Oracle® Database

Advanced Security Administrator's Guide

10*g* Release 2 (10.2)

B14268-02

November 2005

ORACLE®

Oracle Database Advanced Security Administrator's Guide 10*g* Release 2 (10.2)

B14268-02

Primary Author: Sumit Jeloka

Contributor: Rajbir Chahal, Min-Hank Ho, Michael Hwa, Sudha Iyer, Adam Lindsey Jacobs, Supriya Kalyanasundaram, Lakshmi Kethana, Andrew Koyfman, Van Le, Nina Lewis, Stella Li, Janaki Narasinghanallur, Vikram Pesati, Andy Philips, Richard Smith, Deborah Steiner, Philip Thornton, Ramana Turlapati

Contents

2 Configuration and Administration Tools Overview

Part II Data Encryption and Integrity

3 Transparent Data Encryption

4 Configuring Network Data Encryption and Integrity for Oracle Servers and Clients

5 Configuring Network Data Encryption and Integrity for Thin JDBC Clients

Part III   Oracle Advanced Security Strong Authentication

6 Configuring RADIUS Authentication

7   Configuring Kerberos Authentication

8   Configuring Secure Sockets Layer Authentication

9   Using Oracle Wallet Manager

10 Configuring Multiple Authentication Methods and Disabling Oracle Advanced Security

Part IV   Appendixes

A   Data Encryption and Integrity Parameters

B   Authentication Parameters

F orapki Utility

G Entrust-Enabled SSL Authentication

Glossary

Index

List of Figures

List of Tables

Preface

Welcome to the *Oracle Database Advanced Security Administrator's Guide* for the 10*g* Release 2 (10.2) of Oracle Advanced Security.

Oracle Advanced Security contains a comprehensive suite of security features that protect enterprise networks and securely extend them to the Internet. It provides a single source of integration with multiple network encryption and authentication solutions, single sign-on services, and security protocols.

The *Oracle Database Advanced Security Administrator's Guide* describes how to implement, configure and administer Oracle Advanced Security.

This preface contains these topics:

- Audience
- Documentation Accessibility
- Organization
- Related Documentation
- Conventions

Audience

The *Oracle Database Advanced Security Administrator's Guide* is intended for users and systems professionals involved with the implementation, configuration, and administration of Oracle Advanced Security including:

- Implementation consultants
- System administrators
- Security administrators
- Database administrators (DBAs)

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be

accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

http://www.oracle.com/accessibility/

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Organization

This document contains the following chapters:

Part I, "Getting Started with Oracle Advanced Security"

Chapter 1, "Introduction to Oracle Advanced Security"

This chapter provides an overview of Oracle Advanced Security features provided with this release.

Chapter 2, "Configuration and Administration Tools Overview"

This chapter provides an introduction and overview of Oracle Advanced Security GUI and command-line tools.

Part II, "Data Encryption and Integrity"

Chapter 3, "Transparent Data Encryption"

This chapter provides an overview of the transparent data encryption feature introduced in Oracle Advanced Security 10*g* Release 2 (10.2). It describes how to configure and use transparent data encryption services.

Chapter 4, "Configuring Network Data Encryption and Integrity for Oracle Servers and Clients"

This chapter describes how to configure data encryption and integrity within an existing Oracle Net Services 10*g* Release 2 (10.2) network.

Chapter 5, "Configuring Network Data Encryption and Integrity for Thin JDBC Clients"

This chapter provides an overview of the Java implementation of Oracle Advanced Security, which lets Thin Java Database Connectivity (JDBC) clients securely connect to Oracle Database databases.

Part III, "Oracle Advanced Security Strong Authentication"

Chapter 6, "Configuring RADIUS Authentication"

This chapter describes how to configure Oracle for use with RADIUS (Remote Authentication Dial-In User Service). It provides an overview of how RADIUS works within an Oracle environment, and describes how to enable RADIUS authentication and accounting. It also introduces the challenge-response user interface that third party vendors can customize to integrate with third party authentication devices.

Chapter 7, "Configuring Kerberos Authentication"

This chapter describes how to configure Oracle for use with MIT Kerberos and provides a brief overview of steps to configure Kerberos to authenticate Oracle users. It also includes a brief section that discusses interoperability between the Oracle Advanced Security Kerberos adapter and a Microsoft KDC.

Chapter 8, "Configuring Secure Sockets Layer Authentication"

This chapter describes how Oracle Advanced Security supports a public key infrastructure (PKI). It includes a discussion of configuring and using the Secure Sockets Layer (SSL), certificate validation, and hardware security module support features of Oracle Advanced Security.

Chapter 9, "Using Oracle Wallet Manager"

This chapter describes how to use Oracle Wallet Manager to manage Oracle wallets and PKI credentials.

Chapter 10, "Configuring Multiple Authentication Methods and Disabling Oracle Advanced Security"

This chapter describes the authentication methods that can be used with Oracle Advanced Security, and how to use conventional user name and password authentication. It also describes how to configure the network so that Oracle clients can use a specific authentication method, and Oracle servers can accept any method specified.

Part IV, "Appendixes"

Appendix A, "Data Encryption and Integrity Parameters"

This appendix describes Oracle Advanced Security data encryption and integrity configuration parameters.

Appendix B, "Authentication Parameters"

This appendix describes Oracle Advanced Security authentication configuration file parameters.

Appendix C, "Integrating Authentication Devices Using RADIUS"

This appendix explains how third party authentication device vendors can integrate their devices and customize the graphical user interface used in RADIUS challenge-response authentication.

Appendix D, "Oracle Advanced Security FIPS 140-1 Settings"

This appendix describes the `sqlnet.ora` configuration parameters required to comply with the FIPS 140-1 Level 2 evaluated configuration.

Appendix E, "Oracle Advanced Security FIPS 140-2 Settings"

This appendix describes the configuration parameters required to comply with the FIPS 140-2 Level 2 evaluated configuration.

Appendix F, "orapki Utility"

This appendix provides the syntax for the `orapki` command line utility. This utility must be used to manage certificate revocation lists (CRLs). You can also use this utility to create and manage Oracle wallets; create certificate requests, signed certificates, and user certificates for testing purposes; and to export certificates and certificate requests from Oracle wallets.

Appendix G, "Entrust-Enabled SSL Authentication"

This appendix describes how to configure and use Entrust-enabled Oracle Advanced Security for Secure Sockets Layer (SSL) authentication.

Glossary

Related Documentation

For more information, refer to these Oracle resources:

- *Oracle Database Net Services Administrator's Guide*

- *Oracle Database Heterogeneous Connectivity Administrator's Guide*

- *Oracle Database JDBC Developer's Guide and Reference*

- *Oracle Internet Directory Administrator's Guide*

- *Oracle Database Administrator's Guide*

- *Oracle Database Security Guide*

Many books in the documentation set use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Printed documentation is available for sale in the Oracle Store at

http://oraclestore.oracle.com/

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

http://www.oracle.com/technology/membership/index.html

If you already have a user name and password for OTN, then you can go directly to the documentation section of the OTN Web site at

http://www.oracle.com/technology/documentation/index.html

For information from third-party vendors, refer to:

- *ACE/Server Administration Manual, from Security Dynamics*

- *ACE/Server Client for UNIX, from Security Dynamics*

- *ACE/Server Installation Manual, from Security Dynamics*

- *RADIUS Administrator's Guide*

- Notes about building and installing Kerberos from Kerberos version 5 source distribution

- *Entrust/PKI for Oracle*

- *Administering Entrust/PKI on UNIX*

- *Application Environment Specification/Distributed Computing*

For conceptual information about the network security technologies supported by Oracle Advanced Security, you can refer to the following third-party publications:

- *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C* by Bruce Schneier. New York: John Wiley & Sons, 1996.

- *SSL & TLS Essentials: Securing the Web* by Stephen A. Thomas. New York: John Wiley & Sons, 2000.

- *Understanding and Deploying LDAP Directory Services* by Timothy A. Howes, Ph.D., Mark C. Smith, and Gordon S. Good . Indianapolis: New Riders Publishing, 1999.

- *Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations* by Carlisle Adams and Steve Lloyd. Indianapolis: New Riders Publishing, 1999.

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples
- Conventions for Windows Operating Systems

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle Database Concepts*
		Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, data types, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, user names, and roles.	You can specify this clause only for a `NUMBER` column.
		You can back up the database by using the `BACKUP` command.
		Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view.
		Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
lowercase monospace (fixed-width) font	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, user names and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the data files and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
lowercase italic monospace (fixed-width) font	Lowercase italic monospace font represents placeholders or variables.	You can specify the *parallel_clause*. Run U*old_release*.SQL where *old_release* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	DECIMAL (*digits* [, *precision*])
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	{ENABLE | DISABLE}
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	CREATE TABLE ... AS *subquery*; SELECT *col1*, *col2*, ... , *coln* FROM employees;

Convention	Meaning	Example

.
 .
 .
	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	`SQL> SELECT NAME FROM V$DATAFILE;`
		 `NAME`
		 `------------------------------------`
		 `/fsl/dbs/tbs_01.dbf`
		 `/fs1/dbs/tbs_02.dbf`
		 `.`
		 `.`
		 `.`
		 `/fsl/dbs/tbs_09.dbf`
		 `9 rows selected.`

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct    CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*

UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM`
		 `employees;`
		 `SELECT * FROM USER_TABLES;`
		 `DROP TABLE hr.employees;`

lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	
	 Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM`
		 `employees;`
		 `sqlplus hr/hr`
		 `CREATE USER mjones IDENTIFIED BY ty3MU9;`

Conventions for Windows Operating Systems

The following table describes conventions for Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Select Start	How to start a program.	To start the Database Configuration Assistant, Select Start, Programs, Oracle - *HOME_NAME*, Configuration and Migration Tools, Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32 is the same as` `C:\WINNT\SYSTEM32`

Convention	Meaning	Example
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`
Special characters	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"` `C:\>imp SYSTEM/`*`password`* `FROMUSER=scott TABLES=(emp, dept)`
HOME_NAME	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`
ORACLE_HOME and *ORACLE_BASE*	In releases prior to Oracle8i release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory. For Windows, the default location was `C:\orant`. This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install the latest Oracle release on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora`*`nn`*, where *nn* is the latest release number. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle Database Platform Guide for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE\ORACLE_HOME*\rdbms\admin directory.

What's New in Oracle Advanced Security?

This section describes new features of Oracle Advanced Security 10*g* Release 2 (10.2) and provides pointers to additional information. New features information from the previous release is also retained to help those users migrating to the current release.

The following sections describe the new features in Oracle Advanced Security:

- Oracle Database 10g Release 2 (10.2) New Features in Oracle Advanced Security
- Oracle Database 10g Release 1 (10.1) New Features in Oracle Advanced Security

Oracle Database 10*g* Release 2 (10.2) New Features in Oracle Advanced Security

This release includes the following new feature:

- Transparent Data Encryption and Built-in Key Management

  Transparent Data Encryption enables you to encrypt data in columns without having to manage the encryption key. Businesses can protect sensitive data in their databases without having to make changes to their applications.

  Oracle Advanced Security uses industry standard encryption algorithms including AES and 3DES to encrypt columns that have been marked for encryption. Key Management is handled by the database. SQL interfaces to Key Management hide the complexity of encryption.

  > **See Also:** "Supported Encryption Algorithms" on page 1-4 for more information on the encryption algorithms that are supported.
  >
  > Chapter 3, "Transparent Data Encryption" for more information on implementing and using Transparent Data Encryption.

  > **Note:** In this release, the features of Multiplexing and Connection Pooling do not work with SSL transport. Refer to *Oracle Database JDBC Developer's Guide and Reference* for details of encryption support available in JDBC.

Oracle Database 10*g* Release 1 (10.1) New Features in Oracle Advanced Security

This release provides the following new features for strong authentication:

- Support for TLS (Transport Layer Security), version 1.0

TLS is an industry-standard protocol which provides effective security for transactions conducted on the Web. It has been developed by the Internet Engineering Task Force (IETF) to be the successor to SSL version 3.0. TLS is a configurable option provided in Oracle Net Manager.

> **See Also:** Chapter 8, "Configuring Secure Sockets Layer Authentication" for configuration details

- Support for Hardware Security Modules, including Oracle Wallet Manager Integration

  In this release, Oracle Advanced Security supports hardware security modules which use APIs that conform to the RSA Security, Inc., Public-Key Cryptography Standards (PKCS) #11. In addition, it is now possible to create Oracle Wallets that can store credentials on a hardware security module for servers, or private keys on tokens for clients. This provides roaming authentication to the database.

  Hardware security modules can be used for the following functions:

  - Store cryptographic information, such as private keys, which provides stronger security.

  - Perform cryptographic operations to off load RSA operations from the server, freeing the CPU to respond to other transactions.

    > **See Also:**
    >
    > - "Configuring Your System to Use Hardware Security Modules" on page 8-33 for configuration details
    >
    > - "Creating a Wallet to Store Hardware Security Module Credentials" on page 9-8

- Certificate Revocation Lists (CRL) and CRL Distribution Point (CRLDP) Support for Certificate Validation

  In the current release, you have the option to configure certificate revocation status checking for both the client and the server. Certificate revocation status is checked against **CRLs** which are located in file system directories, Oracle Internet Directory, or downloaded from the location specified in the **CRL Distribution Point** (CRL DP) extension on the certificate. The `orapki` utility has also been added for CRL management and for managing Oracle wallets and certificates.

    > **See Also:**
    >
    > - "Certificate Validation with Certificate Revocation Lists" on page 8-24 for details
    >
    > - Appendix F, "orapki Utility" for details about orapki command line utility

Part I

Getting Started with Oracle Advanced Security

This part introduces Oracle Advanced Security, describing security solutions it provides, its features, and its tools.

Part I contains the following chapters:

- Chapter 1, "Introduction to Oracle Advanced Security"
- Chapter 2, "Configuration and Administration Tools Overview"

1
Introduction to Oracle Advanced Security

This chapter introduces Oracle Advanced Security, summarizes the security risks it addresses, and describes its features. These features are available to database and related products that interface with Oracle Net Services, including Oracle Database, Oracle Application Server, and Oracle Identity Management infrastructure.

This chapter contains the following topics:

- Security Challenges in an Enterprise Environment
- Solving Security Challenges with Oracle Advanced Security
- Oracle Advanced Security Architecture
- Secure Data Transfer Across Network Protocol Boundaries
- System Requirements
- Oracle Advanced Security Restrictions

Security Challenges in an Enterprise Environment

To increase efficiency and lower costs, companies adopt strategies to automate business processes. One such strategy is to conduct more business on the Web, but that requires greater computing power, translating to higher IT costs. In response to rising IT costs, more and more businesses are considering enterprise **grid computing** architecture where inexpensive computers act as one powerful system. While such strategies improve the bottom line, they introduce risks, which are associated with securing data in motion and managing an ever increasing number of user identities.

This section examines the security challenges of today's enterprise computing environments in the following topics:

- Security in Enterprise Grid Computing Environments
- Security in an Intranet or Internet Environment
- Common Security Threats

Security in Enterprise Grid Computing Environments

Grid computing is a computing architecture that coordinates large numbers of servers and storage to act as a single large computer. It provides flexibility, lower costs, and IT investment protection because inexpensive, off-the-shelf components can be added to the grid as business needs change. While providing significant benefits, grid computing environments present unique security requirements because their computing resources are distributed and often heterogeneous. The following sections discuss these requirements:

Distributed Environment Security Requirements

Enterprise grid computing pools distributed business computing resources to cost effectively harness the power of clustered servers and storage. A distributed environment requires secure network connections. Even more critical in grid environments, it is necessary to have a uniform definition of "who is the user" and "what is the user allowed to do." Without such uniform definitions, administrators frequently must assign, manage, and revoke authorizations for every user on different software applications to protect employee, customer, and partner information. This is expensive because it takes time, which drives up costs. Consequently, the cost savings gained with grid computing are lost.

Heterogeneous Environment Security Requirements

Because grid computing environments often grow as business needs change, computing resources are added over time, resulting in diverse collections of hardware and software. Such heterogeneous environments require support for different types of authentication mechanisms which adhere to industry standards. Without strict adherence to industry standards, integrating heterogeneous components becomes costly and time consuming. Once again the benefits of grid computing are squandered when the appropriate infrastructure is not present.

Security in an Intranet or Internet Environment

Oracle databases power the largest and most popular Web sites on the Internet. In record numbers, organizations throughout the world are deploying distributed databases and client/server applications based on Oracle Database and Oracle Net Services. This proliferation of distributed computing is matched by an increase in the amount of information that organizations place on computers. Employee and financial records, customer orders, product information, and other sensitive data have moved from filing cabinets to file structures. The volume of sensitive information on the Web has thus increased the value of data that can be compromised.

Common Security Threats

The increased volume of data in distributed, heterogeneous environments exposes users to a variety of security threats, including the following:

- Eavesdropping and Data Theft
- Data Tampering
- Falsifying User Identities
- Password-Related Threats

Eavesdropping and Data Theft

Over the Internet and in wide area network environments, both public carriers and private networks route portions of their network through insecure land lines, vulnerable microwave and satellite links, or a number of servers— exposing valuable data to interested third parties. In local area network environments within a building or campus, the potential exists for insiders with access to the physical wiring to view data not intended for them, and network **sniffer**s can be installed to eavesdrop on network traffic.

Data Tampering

Distributed environments bring with them the possibility that a malicious third party can compromise integrity by tampering with data as it moves between sites.

Falsifying User Identities

In a distributed environment, it is more feasible for a user to falsify an identity to gain access to sensitive information. How can you be sure that user Pat connecting to Server A from Client B really is user Pat?

Moreover, in distributed environments, malefactors can hijack connections. How can you be sure that Client B and Server A are what they claim to be? A transaction that should go from the Personnel system on Server A to the Payroll system on Server B could be intercepted in transit and re-routed to a terminal masquerading as Server B.

Password-Related Threats

In large systems, users typically must remember multiple passwords for the different applications and services that they use. For example, a developer can have access to a development application on a workstation, a PC for sending e-mail, and several computers or intranet sites for testing, reporting bugs, and managing configurations.

Users typically respond to the problem of managing multiple passwords in several ways:

- They may select easy-to-guess passwords, such as a name, a fictional character, or a word found in a dictionary. All of these passwords are vulnerable to **dictionary attack**s.

- They may also choose to standardize passwords so that they are the same on all systems or Web sites. This results in a potentially large exposure in the event of a compromised password. They can also use passwords with slight variations that can be easily derived from known passwords.

- Users with complex passwords may write them down where an attacker can easily find them, or they may just forget them, requiring costly administration and support efforts.

All of these strategies compromise password secrecy and service availability. Moreover, administration of multiple user accounts and passwords is complex, time-consuming, and expensive.

Solving Security Challenges with Oracle Advanced Security

To solve enterprise computing security problems, Oracle Advanced Security provides industry standards-based data privacy, integrity, authentication, single sign-on, and access authorization in a variety of ways. For example, you can configure either Oracle Net native encryption or Secure Sockets Layer (SSL) for data privacy. Oracle Advanced Security also provides the choice of several strong authentication methods, including Kerberos, smart cards, and digital certificates.

Oracle Advanced Security provides the following security features:

- Data Encryption
- Strong Authentication

Data Encryption

Sensitive information that is stored in your database or that travels over enterprise networks and the Internet can be protected by encryption algorithms. An encryption algorithm transforms information into a form that cannot be deciphered without a decryption key.

Figure 1–1 shows how encryption works to ensure the security of a transaction sent over the network. For example, if a manager approves a bonus, this data should be encrypted when sent over the network to avoid eavesdropping. If all communication between the client, the database, and the application server is encrypted, then when the manager sends the bonus amount to the database, it is protected.

Figure 1–1   Encryption
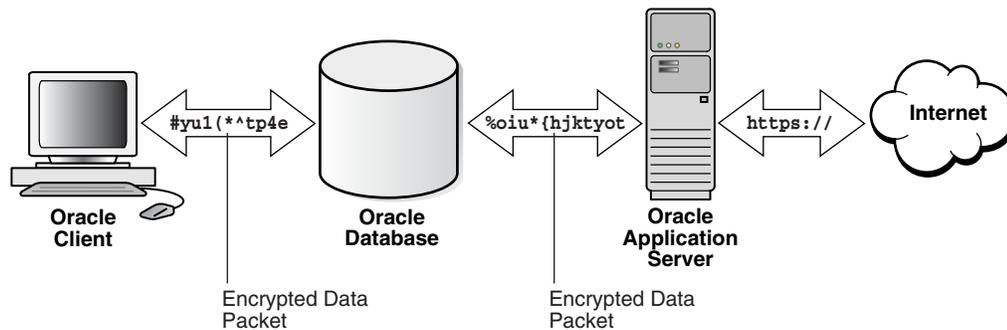
This section discusses the following topics:

- Supported Encryption Algorithms
- Data Integrity
- Federal Information Processing Standard

Supported Encryption Algorithms

Oracle Advanced Security provides the following encryption algorithms to protect the privacy of network data transmissions:

- RC4 Encryption:
- DES Encryption :
- Triple-DES Encryption :
- Advanced Encryption Standard:

Selecting the network encryption algorithm is a user configuration option, providing varying levels of security and performance for different types of data transfers.

Prior versions of Oracle Advanced Security provided three editions: Domestic, Upgrade, and Export, each with different key lengths. Oracle Advanced Security 10*g* Release 2 (10.2) contains a complete complement of the available encryption algorithms and key lengths, previously only available in the Domestic edition. Users deploying prior versions of the product can obtain the Domestic edition for a specific product release.

> **Note:**   The U.S. government has relaxed its export guidelines for encryption products. Accordingly, Oracle can ship Oracle Advanced Security with its strongest encryption features to all of its customers.

RC4 Encryption: The RC4 encryption module uses the RSA Security, Inc., RC4 encryption algorithm. Using a secret, randomly-generated key unique to each session, all network traffic is fully safeguarded including all data values, SQL statements, and stored procedure calls and results. The client, server, or both, can request or require

the use of the encryption module to guarantee that data is protected. Oracle's optimized implementation provides a high degree of security for a minimal performance penalty. For the RC4 algorithm, Oracle provides encryption key lengths of 40-bits, 56-bits, 128-bits, and 256-bits.

DES Encryption : Oracle Advanced Security implements the U.S. Data Encryption Standard algorithm (DES) with a standard, optimized 56-bit key encryption algorithm and also provides DES40, a 40-bit version, for backward compatibility.

Triple-DES Encryption : Oracle Advanced Security also supports Triple-DES encryption (3DES), which encrypts message data with three passes of the DES algorithm. 3DES provides a high degree of message security, but with a performance penalty. The magnitude of penalty depends on the speed of the processor performing the encryption. 3DES typically takes three times as long to encrypt a data block as compared with the standard DES algorithm.

3DES is available in two-key and three-key versions, with effective key lengths of 112-bits and 168-bits, respectively. Both versions operate in outer **Cipher Block Chaining (CBC)** mode.

Advanced Encryption Standard: Approved by the National Institute of Standards and Technology (NIST) in Federal Information Processing Standards (FIPS) Publication 197, Advanced Encryption Standard (AES) is a new cryptographic algorithm standard developed to replace DES. AES is a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits, which are referred to as AES-128, AES-192, and AES-256, respectively. All three versions operate in outer-CBC mode.

> **See Also:**
>
> - Chapter 4, "Configuring Network Data Encryption and Integrity for Oracle Servers and Clients"
>
> - Appendix A, "Data Encryption and Integrity Parameters"

Data Integrity

To ensure the **integrity** of data packets during transmission, Oracle Advanced Security can generate a cryptographically secure message digest using MD5 or SHA-1 hashing algorithms and include it with each message sent across a network.

Data integrity algorithms add little overhead and protect against the following attacks:

- Data modification

- Deleted packets

- Replay attacks

> **Note:**   SHA-1 is slightly slower than MD5 but produces a larger message digest, making it more secure against brute-force collision and inversion attacks.

> **See Also:**   Chapter 4, "Configuring Network Data Encryption and Integrity for Oracle Servers and Clients", for information about MD5 and SHA-1

Federal Information Processing Standard

Oracle Advanced Security Release 8.1.6 has been validated under U.S. Federal Information Processing Standard 140-1 (FIPS) at the Level 2 security level. This provides independent confirmation that Oracle Advanced Security conforms to federal government standards. FIPS 140-1 related configuration settings are described in Appendix D, "Oracle Advanced Security FIPS 140-1 Settings".

The cryptographic libraries for SSL included in Oracle Database 10g have been validated under FIPS 140-2 at the Level 2 security level. FIPS 140-2 related configuration settings are described in Appendix E, "Oracle Advanced Security FIPS 140-2 Settings".

Strong Authentication

Authentication is used to prove the identity of the user. Authenticating user identity is imperative in distributed environments, without which there can be little confidence in network security. Passwords are the most common means of authentication. Oracle Advanced Security enables strong authentication with Oracle authentication adapters that support various third-party authentication services, including SSL with digital certificates.

Figure 1–2 shows user authentication with an Oracle database instance configured to use a third-party authentication server. Having a central facility to authenticate all members of the network (clients to servers, servers to servers, users to both clients and servers) is one effective way to address the threat of network nodes falsifying their identities.

Figure 1–2   Strong Authentication with Oracle Authentication Adapters
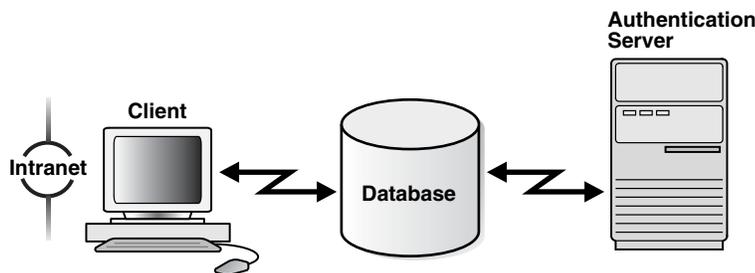
This section contains the following topics:

- Centralized Authentication and Single Sign-On
- Supported Authentication Methods

Centralized Authentication and Single Sign-On

Centralized authentication also provides the benefit of **single sign-on (SSO)** for users. Single sign-on enables users to access multiple accounts and applications with a single password. A user only needs to login once and can then automatically connect to any other service without having to giving user name and password again. Single sign-on eliminates the need for the user to remember and administer multiple passwords, reducing the time spent logging into multiple services.

How Centralized Network Authentication Works Figure 1–3 shows how a centralized network authentication service typically operates.

Figure 1–3   How a Network Authentication Service Authenticates a User
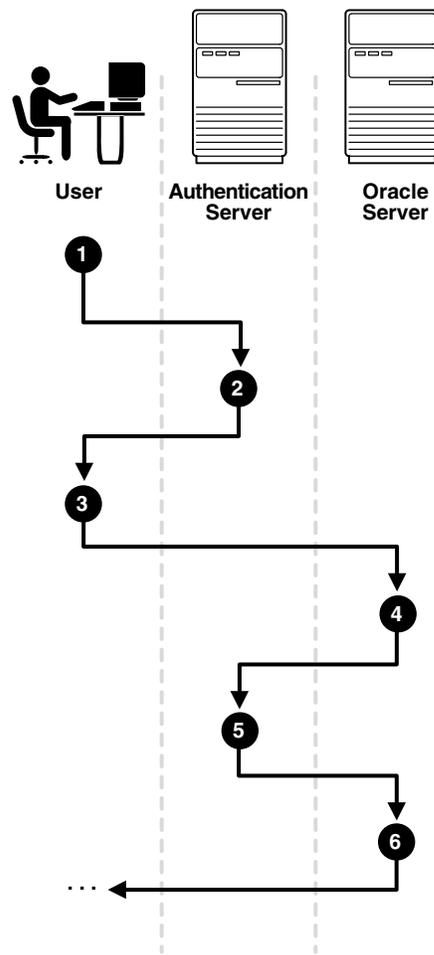
The following steps describe how centralized Network Authentication Process works.

1. A user (client) requests authentication services and provides identifying information, such as a token or password.

2. The authentication server validates the user's identity and passes a ticket or credentials back to the client, which may include an expiration time.

3. The client passes these credentials to the Oracle server concurrent with a service request, such as connection to a database.

4. The server sends the credentials back to the authentication server for authentication.

5. If the authentication server accepts the credentials, then it notifies the Oracle Server and the authenticates the user.

6. If the authentication server does not accept the credentials, then authentication fails, and the service request is denied.

Supported Authentication Methods

Oracle Advanced Security supports the following industry-standard authentication methods:

- Kerberos

- Remote Authentication Dial-In User Service (RADIUS) :

- Secure Sockets Layer (with digital certificates)

- Entrust/PKI :

Kerberos Oracle Advanced Security support for Kerberos provides the benefits of single sign-on and centralized authentication of Oracle users. Kerberos is a trusted third-party authentication system that relies on shared secrets. It presumes that the third party is secure, and provides single sign-on capabilities, centralized password storage, database link authentication, and enhanced PC security. It does this through a Kerberos authentication server. Refer to Chapter 7, "Configuring Kerberos Authentication" for information about configuring and using this adapter.

> **Note:** Oracle authentication for Kerberos provides database link authentication (also called proxy authentication). Kerberos is also an authentication method that is supported with Enterprise User Security.

Remote Authentication Dial-In User Service (RADIUS) : RADIUS is a client/server security protocol that is most widely known for enabling remote authentication and access. Oracle Advanced Security uses this standard in a client/server network environment to enable use of any authentication method that supports the RADIUS protocol. RADIUS can be used with a variety of authentication mechanisms, including token cards and smart cards.

> **See Also:** Chapter 6, "Configuring RADIUS Authentication" for information about configuring and using RADIUS

- **Smart Cards**

  A RADIUS-compliant smart card is a credit card-like hardware device which has memory and a processor. It is read by a smart card reader located at the client workstation.

- **Token Cards**

  Token cards (Secure ID or RADIUS-compliant) can improve ease of use through several different mechanisms. Some token cards dynamically display one-time passwords that are synchronized with an authentication service. The server can verify the password provided by the token card at any given time by contacting the authentication service. Other token cards have a keypad and operate on a challenge-response basis. In this case, the server offers a challenge (a number) that the user enters into a token card. The token card provides a response (another number cryptographically derived from the challenge) that the user enters and sends to the server.

  You can use SecurID tokens through the RADIUS adapter.

Secure Sockets Layer Secure Sockets Layer (SSL) is an industry standard protocol for securing network connections. SSL provides **authentication**, data **encryption**, and data **integrity**.

The SSL protocol is the foundation of a **public key infrastructure (PKI)**. For authentication, SSL uses digital certificates that comply with the X.509v3 standard and a **public and private key pair**.

Oracle Advanced Security SSL can be used to secure communications between any client and any server. You can configure SSL to provide authentication for the server only, the client only, or both client and server. You can also configure SSL features in combination with other authentication methods supported by Oracle Advanced Security (database user names and passwords, RADIUS, and Kerberos).

To support your PKI implementation, Oracle Advanced Security includes the following features in addition to SSL:

- Oracle wallets, where you can store PKI credentials
- Oracle Wallet Manager, which you can use to manage your Oracle wallets
- Certificate validation with certificate revocation lists (CRLs)
- Hardware security module support

> **See Also:**
>
> - Chapter 8, "Configuring Secure Sockets Layer Authentication" for conceptual, configuration, and usage information about SSL, certificate validation, and hardware security modules
>
> - Chapter 9, "Using Oracle Wallet Manager" for information about using this tool to manage Oracle wallets
>
> - Chapter 10, "Configuring Multiple Authentication Methods and Disabling Oracle Advanced Security" for information about configuring SSL in combination with other authentication methods

Entrust/PKI : Oracle Advanced Security supports the public key infrastructure provided by the Entrust/PKI software from Entrust Technologies, Inc. Entrust-enabled Oracle Advanced Security lets Entrust users incorporate Entrust single sign-on into their Oracle applications, and it lets Oracle users incorporate Entrust-based single sign-on into Oracle applications.

> **See Also:** Appendix G, "Entrust-Enabled SSL Authentication" for more information about this feature

Oracle Advanced Security Architecture

Oracle Advanced Security complements an Oracle server or client installation with advanced security features. Figure 1–4 shows the Oracle Advanced Security architecture within an Oracle networking environment.

Figure 1–4   Oracle Advanced Security in an Oracle Networking Environment
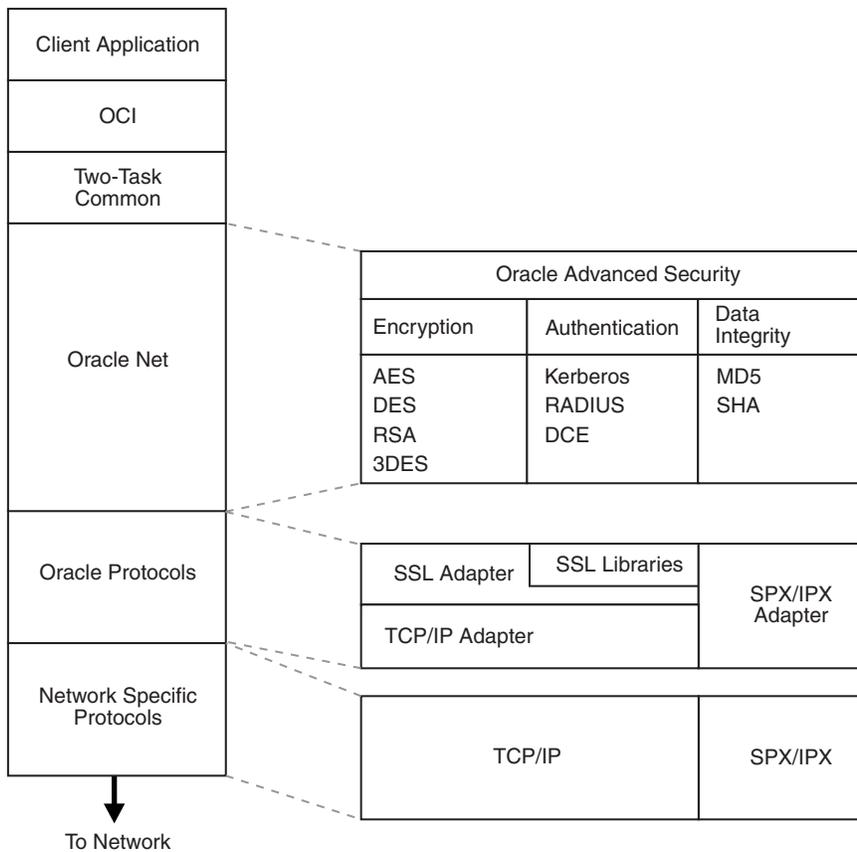
Oracle Advanced Security supports authentication through adapters that are similar to the existing Oracle protocol adapters. As shown in Figure 1–5, authentication adapters integrate the Oracle Net interface, and allow existing applications to take advantage of new authentication systems transparently, without any changes to the application.

Figure 1–5   Oracle Net Services with Authentication Adapters
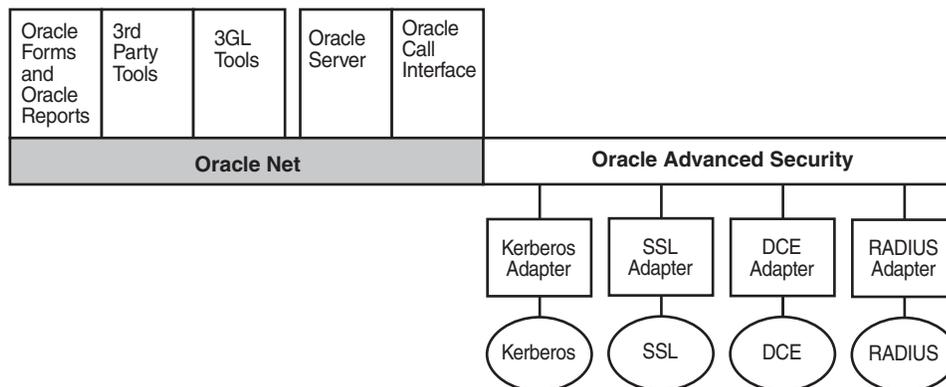
> **See Also:** *Oracle Database Net Services Administrator's Guide* for more information about stack communications in an Oracle networking environment

Secure Data Transfer Across Network Protocol Boundaries

Oracle Advanced Security is fully supported by Oracle Connection Manager, making secure data transfer a reality across network protocol boundaries. Clients using LAN protocols such as NetWare (SPX/IPX), for example, can securely share data with large servers using different network protocols such as LU6.2, TCP/IP, or DECnet. To eliminate potential weak points in the network infrastructure and to maximize performance, Connection Manager passes encrypted data from protocol to protocol without the cost and exposure of decryption and re-encryption.

System Requirements

Oracle Advanced Security is an add-on product bundled with the Oracle Net Server or Oracle Net Client. It must be purchased and installed on both the client and the server.

Oracle Advanced Security 10*g* Release 2 (10.2) requires Oracle Net 10*g* Release 2 (10.2) and supports Oracle Database Enterprise Edition. Table 1–1 lists additional system requirements.

> **Note:** Oracle Advanced Security is not available with Oracle Database Standard Edition.

Table 1–1   Authentication Methods and System Requirements

Authentication Method	System Requirements
Kerberos	■ MIT Kerberos Version 5, release 1.1 ■ The Kerberos authentication server must be installed on a physically secure system.
RADIUS	■ A RADIUS server that is compliant with the standards in the Internet Engineering Task Force (IETF) RFC #2138, *Remote Authentication Dial In User Service (RADIUS)* and RFC #2139 *RADIUS Accounting.* ■ To enable challenge-response authentication, you must run RADIUS on an operating system that supports the Java Native Interface as specified in release 1.1 of the Java Development Kit from JavaSoft.
SSL	■ A wallet that is compatible with the Oracle Wallet Manager 10*g release.* Wallets created in earlier releases of the Oracle Wallet Manager are not forward compatible.
Entrust/PKI	■ Entrust IPSEC Negotiator Toolkit Release 6.0 ■ Entrust/PKI 6.0

Oracle Advanced Security Restrictions

Oracle Applications support Oracle Advanced Security encryption and data integrity. However, because Oracle Advanced Security requires Oracle Net Services to transmit data securely, Oracle Advanced Security external authentication features are not supported by some parts of Oracle Financial, Human Resource, and Manufacturing Applications when they are running on Microsoft Windows. The portions of these products that use Oracle Display Manager (ODM) do not take advantage of Oracle Advanced Security, because ODM does not use Oracle Net Services.

2

Configuration and Administration Tools Overview

Configuring advanced security features for an Oracle database instance includes configuring encryption, integrity (checksumming), and strong authentication methods for Oracle Net Services. Strong authentication method configuration can include third-party software, as is the case for Kerberos or RADIUS, or it may entail configuring and managing a public key infrastructure for using digital certificates with Secure Sockets Layer (SSL).

Such diverse advanced security features require a diverse set of tools with which to configure and administer them. This chapter introduces the tools used to configure and administer advanced security features for an Oracle database in the following topics:

- Network Encryption and Strong Authentication Configuration Tools
- Public Key Infrastructure Credentials Management Tools
- Duties of a Security Administrator/DBA

Network Encryption and Strong Authentication Configuration Tools

Oracle Net Services can be configured to encrypt data using standard encryption algorithms, and for strong authentication methods, such as Kerberos, RADIUS, and SSL. The following sections introduce the Oracle tools you can use to configure these advanced security features for an Oracle Database:

- Oracle Net Manager
- Oracle Advanced Security Kerberos Adapter Command-Line Utilities

Oracle Net Manager

Oracle Net Manager is a graphical user interface tool, primarily used to configure Oracle Net Services for an Oracle home on a local client or server host.

Although you can use Oracle Net Manager to configure Oracle Net Services, such as naming, listeners, and general network settings, it also enables you to configure the following Oracle Advanced Security features, which use the Oracle Net protocol:

- Strong authentication (Kerberos, RADIUS, and Secure Sockets Layer)
- Network encryption (RC4, DES, Triple-DES, and AES)
- Checksumming for data integrity (MD5, SHA-1)

This section introduces you to the features of Oracle Net Manager that are used to configure Oracle Advanced Security. It contains the following topics:

- Starting Oracle Net Manager

- Navigating to the Oracle Advanced Security Profile

> **See Also:**
>
> - "Duties of a Security Administrator/DBA" on page 2-9 for information about the tasks you can perform with this tool that configure advanced security features
>
> - *Oracle Database Net Services Administrator's Guide* and Oracle Net Manager online Help for complete documentation of this tool

Starting Oracle Net Manager

You can start Oracle Net Manager by using Oracle Enterprise Manager Console or as a standalone application. However, you must use the standalone application to access the Oracle Advanced Security Profile where you can configure Oracle Advanced Security features.

To start Oracle Net Manager as a standalone application:

- (UNIX) From *$ORACLE_HOME*/bin, enter the following at the command line:

  ```
  netmgr
  ```

- (Windows) Select **Start**, **Programs**, **Oracle - HOME_NAME**, **Configuration and Migration Tools**, **Net Manager**

Navigating to the Oracle Advanced Security Profile

The Oracle Net Manager interface window contains two panes: the navigator pane and the right pane.The interface displays various property sheets that enable you to configure network components. When you select a network object in the navigator pane, its associated property sheets displays in the right pane. To configure Oracle Advanced Security features, select the **Profile** object in the navigator pane, and then select **Oracle Advanced Security** from the list in the right pane, as shown in Figure 2–1.

Figure 2–1   Oracle Advanced Security Profile in Oracle Net Manager
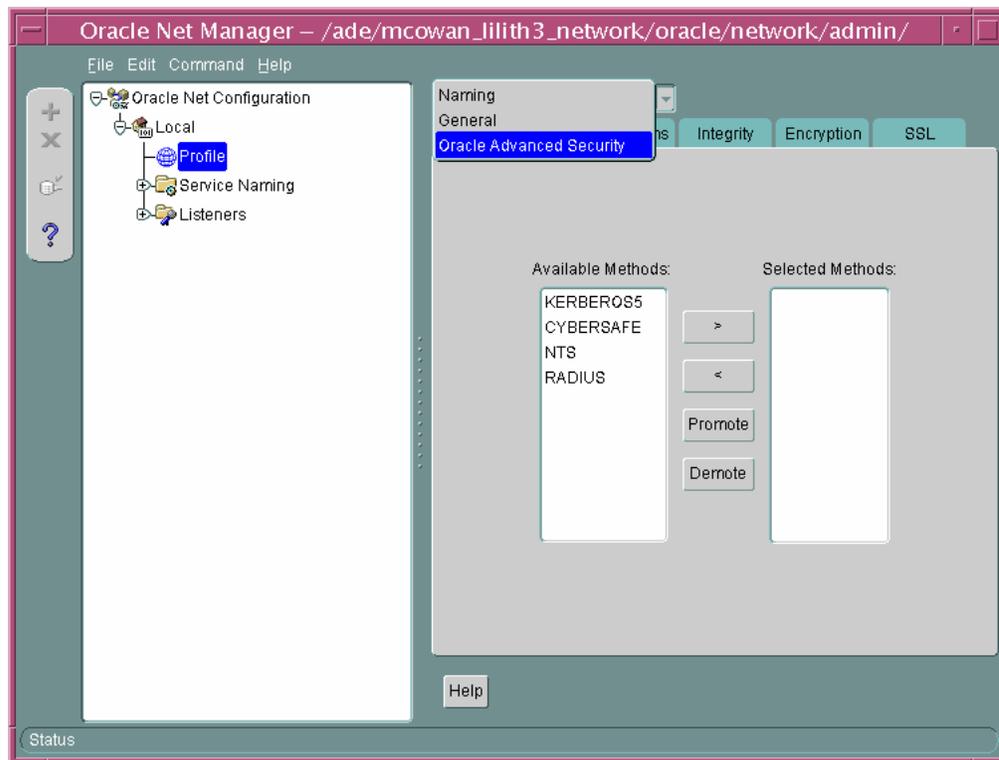
Oracle Advanced Security Profile Property Sheets

The Oracle Advanced Security Profile contains the following property sheets:

- Authentication Property Sheet

- Other Params Property Sheet

- Integrity Property Sheet

- Encryption Property Sheet

- SSL Property Sheet

Authentication Property Sheet Use this property sheet to select a strong authentication method, such as Kerberos Version 5 (KERBEROS5), Windows native authentication (NTS), or RADIUS.

Other Params Property Sheet Use this property sheet to set other parameters for the authentication method you selected on the Authentication property sheet.

Integrity Property Sheet Use this property sheet to enable checksumming on the client or the server and to select an encryption algorithm for generating secure message digests.

Encryption Property Sheet Use this property sheet to select one or more **cipher suite**s to encrypt client or server connections with native encryption algorithms.

SSL Property Sheet Use this property sheet to configure Secure Sockets Layer (SSL), including the **wallet** location and **cipher suite**, on a client or server.

Oracle Advanced Security Kerberos Adapter Command-Line Utilities

The Oracle Advanced Security Kerberos adapter provides three command-line utilities that enable you to obtain, cache, display, and remove Kerberos credentials. The following table briefly describes these utilities:

Utility Name	Description
okinit	Obtains Kerberos tickets from the key distribution center (**KDC**) and caches them in the user's credential cache
oklist	Displays a list of Kerberos tickets in the specified credential cache
okdstry	Removes Kerberos credentials from the specified credential cache

> **See Also:** "Utilities for the Kerberos Authentication Adapter" on page 7-8 for complete descriptions of these utilities, their syntax, and available options

> **Note:** The Cybersafe adapter is not supported beginning with this release. You should use Oracle's Kerberos adapter in its place. Kerberos authentication with the Cybersafe KDC (Trust Broker) continues to be supported when using the Kerberos adapter.

Public Key Infrastructure Credentials Management Tools

The security provided by a public key infrastructure (PKI) depends on how effectively you store, manage, and validate your PKI credentials. The following Oracle tools are used to manage certificates, wallets, and certificate revocation lists so your PKI credentials can be stored securely and your certificate validation mechanisms kept current:

- Oracle Wallet Manager
- orapki Utility

Oracle Wallet Manager

Oracle Wallet Manager is an application that wallet owners and security administrators use to manage and edit the security credentials in their Oracle wallets. A wallet is a password-protected container that is used to store authentication and signing credentials, including private keys, certificates, and trusted certificates needed by SSL. You can use Oracle Wallet Manager to perform the following tasks:

- Create **public and private key pair**s
- Store and manage user credentials
- Generate certificate requests
- Store and manage **certificate authority** certificates (**root key certificate** and **certificate chain**)
- Upload and download wallets to and from an LDAP directory
- Create wallets to store hardware security module credentials

The following topics introduce the Oracle Wallet Manager user interface:

- Starting Oracle Wallet Manager
- Navigating the Oracle Wallet Manager User Interface
- Toolbar
- Menus

> **See Also:** Chapter 9, "Using Oracle Wallet Manager" for detailed
> information about using this application

Starting Oracle Wallet Manager

To start Oracle Wallet Manager:

- (UNIX) From *$ORACLE_HOME*/bin, enter the following at the command line:

  owm
- (Windows) Select **Start**, **Programs**, **Oracle HOME_NAME**, **Integrated Management Tools**, **Wallet Manager**

Navigating the Oracle Wallet Manager User Interface

The Oracle Wallet Manager interface includes two panes, a toolbar, and various menu items as shown in Figure 2–2.

Figure 2–2   Oracle Wallet Manager User Interface
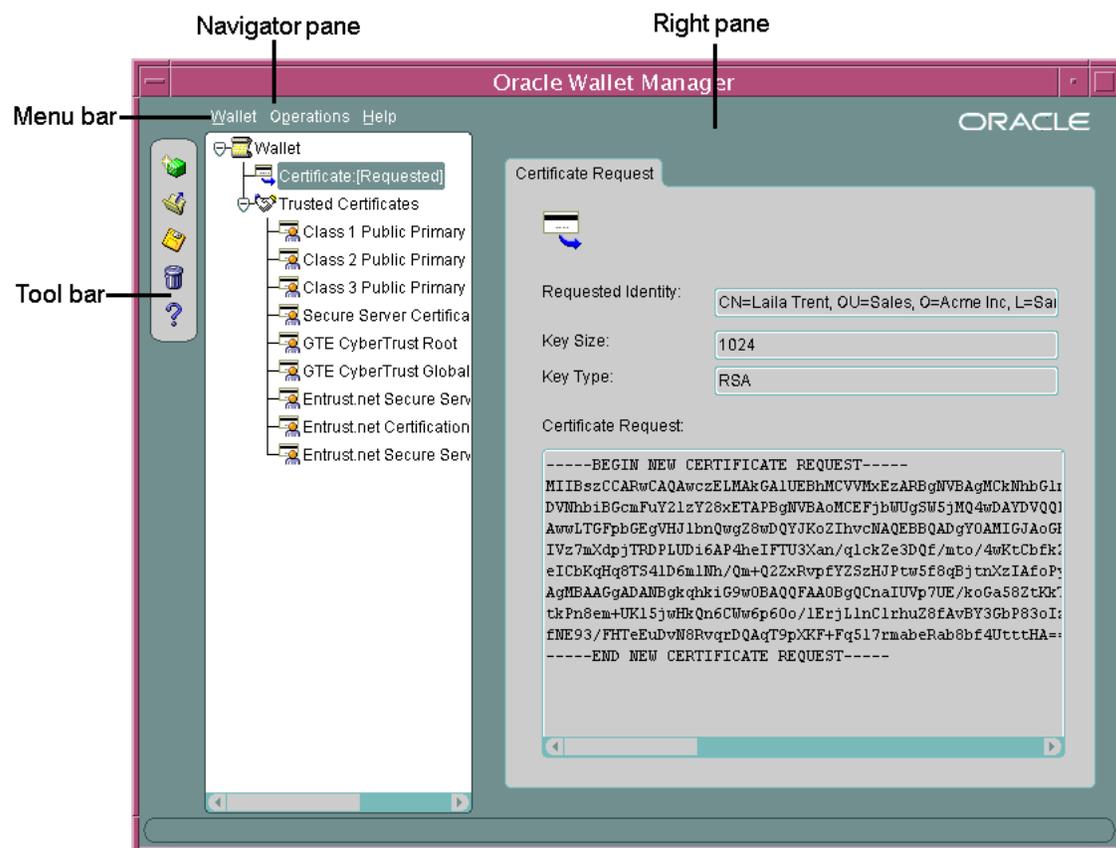
Navigator Pane The navigator pane provides a graphical navigation tree view of the certificate requests and certificates stored in the Oracle home where Oracle Wallet

Manager is installed. You can use the navigator pane to view, modify, add, or delete certificates and certificate requests.

The navigator pane functions the same way as it does in other Oracle graphical user interface tools, enabling you to

- Expand and contract wallet objects so that you can manage the user and trusted certificates they contain.

- Right-click a wallet, certificate, or certificate request to perform operations on it such as add, remove, import, or export.

When you expand a wallet, you see a nested list of user and trusted certificates. When you select a wallet or certificate in the navigator pane, details about your selection display in the adjacent right pane of Oracle Wallet Manager. Table 2–1 lists the main objects that display in the navigator pane.

Table 2–1   Oracle Wallet Manager Navigator Pane Objects

Object	Description
Wallet	Password-protected container that is used to store authentication and signing credentials
Certificate Request[1]	A **PKCS #10**-encoded message containing the requester's **distinguished name (DN)**, a public key, the key size, and key type.
Certificate[1]	An **X.509** data structure containing the entity's DN, public key, and is signed by a trusted identity (**certificate authority**).
Trusted Certificates[1]	Sometimes called a root key certificate, is a certificate from a third party identity that is qualified with a level of trust.

[1] These objects display only after you create a wallet, generate a certificate request, and import a certificate into the wallet.

Right Pane The right pane displays information about an object that is selected in the navigator pane. The right pane is read-only.

Figure 2–3 shows what is displayed in the right pane when a certificate request object is selected in the navigator pane. Information about the request and the requester's identity display in the **Requested Identity**, **Key Size**, and **Key Type** fields. The PKCS #10-encoded certificate request displays in the **Certificate Request** text box. To request a certificate from a certificate authority, you can copy this request into an e-mail or export it into a file.

> **Note:** Figure 2–3 shows a certificate request for a user. A certificate can also be requested for a server in which case the CN attribute will contain the name of the server in place of the user name.

Figure 2–3   Certificate Request Information Displayed in Oracle Wallet Manager Right Pane
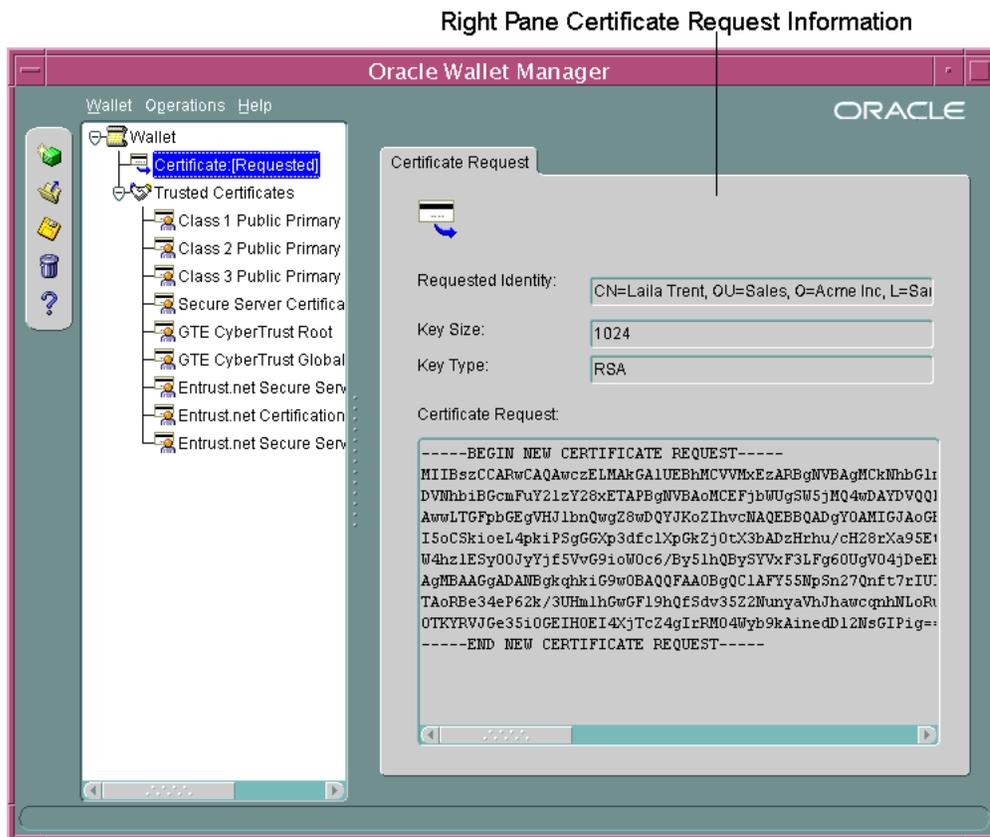
Toolbar

The toolbar contains buttons that enable you to manage your wallets. Move the mouse cursor over a toolbar button to display a description of the button's function. The toolbar buttons are listed and described in Table 2–2.

Table 2–2   Oracle Wallet Manager Toolbar Buttons

Toolbar Button	Description
New	Creates a new wallet
Open Wallet	Enables you to browse your file system to locate and open an existing wallet
Save Wallet	Saves the currently open wallet
Delete Wallet	Deletes the wallet that is currently selected in the navigator pane
Help	Opens the Oracle Wallet Manager online Help

Menus

You use Oracle Wallet Manager menus to manage your wallets and the credentials they contain. The following sections describe the options that are available under each menu.

Wallet Menu Table 2–3 describes the contents of the **Wallet** menu.

Table 2–3   Oracle Wallet Manager Wallet Menu Options

Option	Description
New	Creates a new wallet
Open	Opens an existing wallet
Close	Closes the currently open wallet
Upload Into The Directory Service	Uploads a wallet to a specified LDAP directory server. You must supply a directory password, host name, and port information.
Download From The Directory Service	Downloads a wallet from a specified LDAP directory server. You must supply a directory password, host name, and port information.
Save	Saves the currently open wallet in the current working directory
Save As	Enables you to browse your file system to choose a directory location in which to save the currently open wallet
Save In System Default	Saves the currently open wallet in the system default location: ■ (UNIX) `/etc/ORACLE/WALLETS/username` ■ (Windows) `%USERPROFILE%\ORACLE\WALLETS`
Delete	Deletes the wallet in the current working directory. You must supply the wallet password.
Change Password	Changes the password for the currently open wallet. You must supply the old password before you can create a new one.
Auto Login	Sets the auto login feature for the currently open wallet.
Exit	Exits the Oracle Wallet Manager application

Operations Menu Table 2–4 describes the contents of the **Operations** menu.

Table 2–4   Oracle Wallet Manager Operations Menu Options

Option	Description
Add Certificate Request	Generates a certificate request for the currently open wallet that you can use to request a certificate from a **certificate authority** (CA)
Import User Certificate	Imports the user certificate issued to you from the CA. You must import the issuing CA's certificate as a trusted certificate before you can import the user certificate.
Import Trusted Certificate	Imports the CA's trusted certificate
Remove Certificate Request	Deletes the certificate request in the currently open wallet. You must remove the associated user certificate before you can delete a certificate request.
Remove User Certificate	Deletes the user certificate from the currently open wallet.
Remove Trusted Certificate	Removes the trusted certificate that is selected in the navigator pane from the currently open wallet. You must remove all user certificates that the trusted certificate signs before you can remove it.
Export User Certificate	Exports the user certificate in the currently open wallet to save in a file system directory
Export Certificate Request	Exports the certificate request in the currently open wallet to save in a file

Table 2–4   (Cont.) Oracle Wallet Manager Operations Menu Options

Option	Description
Export Trusted Certificate	Exports the trusted certificate that is selected in the navigator pane to save in another location in your file system
Export All Trusted Certificates	Exports all trusted certificates in the currently open wallet to save in another location in your file system
Export Wallet	Exports the currently open wallet to save as a text file

Help Menu Table 2–5 describes the contents of the **Help** menu.

Table 2–5   Oracle Wallet Manager Help Menu Options

Option	Description
Contents	Opens Oracle Wallet Manager online Help
Search for Help on	Opens Oracle Wallet Manager online Help and displays the Search tab
About Oracle Wallet Manager	Opens a window that displays the Oracle Wallet Manager version number and copyright information

orapki Utility

The orapki utility is a command line tool that you can use to manage certificate revocation lists (**CRLs**), create and manage Oracle wallets, and to create signed certificates for testing purposes.

The basic syntax for this utility is as follows:

```
orapki module command -option_1 argument ... -option_n argument
```

For example, the following command lists all CRLs in the CRL subtree in an instance of Oracle Internet Directory that is installed on machine1.us.acme.com and that uses port 389:

```
orapki crl list -ldap machine1.us.acme.com:389
```

> **See Also:**
>
> ■ "Certificate Revocation List Management" on page 8-27 for information about how to use orapki to manage CRLs in the directory
>
> ■ Appendix F, "orapki Utility" for reference information on all available orapki commands

Duties of a Security Administrator/DBA

Most of the tasks of a security administrator involve ensuring that the connections to and from Oracle databases are secure. Table 2–6 lists the primary tasks of security administrators, the tools used to perform the tasks, and links to where the tasks are documented.

Table 2–6   Common Security Administrator/DBA Configuration and Administrative Tasks

Task	Tools Used	See Also
Configure encrypted Oracle Net connections between database servers and clients	Oracle Net Manager	"Configuring Encryption on the Client and the Server" on page 4-7
Configure checksumming on Oracle Net connections between database servers and clients	Oracle Net Manager	"Configuring Integrity on the Client and the Server" on page 4-8
Configure database clients to accept RADIUS authentication	Oracle Net	"Step 1: Configure RADIUS on the Oracle Client" on page 6-7
Configure a database to accept RADIUS authentication	Oracle Net	"Step 2: Configure RADIUS on the Oracle Database Server" on page 6-8
Create a RADIUS user and grant them access to a database session	SQL*Plus	"Task 3: Create a User and Grant Access" on page 6-13
Configure Kerberos authentication on a database client and server	Oracle Net Manager	"Task 7: Configure Kerberos Authentication" on page 7-4
Create a Kerberos database user	■ `kadmin.local` ■ Oracle Net Manager	■ "Task 8: Create a Kerberos User" on page 7-7 ■ "Task 9: Create an Externally Authenticated Oracle User" on page 7-7
Manage Kerberos credentials in the credential cache	■ `okinit` ■ `oklist` ■ `okdstry`	■ "Obtaining the Initial Ticket with the okinit Utility" on page 7-8 ■ "Displaying Credentials with the oklist Utility" on page 7-9 ■ "Removing Credentials from the Cache File with the okdstry Utility" on page 7-9
Create a wallet for a database client or server	■ Oracle Wallet Manager	"Creating a New Wallet" on page 9-8
Request a user certificate from a **certificate authority** (CA) for SSL authentication	■ Oracle Wallet Manager	■ "Adding a Certificate Request" on page 9-15 ■ "Importing the User Certificate into the Wallet" on page 9-17
Import a user certificate and its associated trusted certificate (CA certificate) into a wallet	■ Oracle Wallet Manager	■ "Importing a Trusted Certificate" on page 9-21 ■ "Importing the User Certificate into the Wallet" on page 9-17
Configuring SSL connections for a database client	■ Oracle Net Manager	"Task 3: Configure SSL on the Client" on page 8-15
Configuring SSL connections for a database server	■ Oracle Net Manager	"Task 2: Configure SSL on the Server" on page 8-9
Enabling certificate validation with **certificate revocation lists**	■ Oracle Net Manager	■ "Configuring Certificate Validation with Certificate Revocation Lists" on page 8-25

Part II

Data Encryption and Integrity

This part describes how to implement and manage transparent data encryption in your Oracle databases. It also describes how to configure Oracle Advanced Security data encryption and integrity for your Oracle network and for thin JDBC connections to the database.

Part II contains the following chapters:

- Chapter 3, "Transparent Data Encryption"

- Chapter 4, "Configuring Network Data Encryption and Integrity for Oracle Servers and Clients"

- Chapter 5, "Configuring Network Data Encryption and Integrity for Thin JDBC Clients"

3

Transparent Data Encryption

This chapter describes how to secure sensitive data within an Oracle database by using transparent data encryption, the feature that enables you to encrypt database columns and manage encryption keys. This chapter contains the following topics:

- About Transparent Data Encryption
- Using Transparent Data Encryption
- Managing Transparent Data Encryption

About Transparent Data Encryption

Oracle Database 10*g* uses authentication, authorization, and auditing mechanisms to secure data in the database, but not in the operating system files where the data is stored. To protect those files, Oracle Database 10*g* provides transparent data encryption. This feature enables you to protect sensitive data in database columns stored in operating system files by encrypting it. Then, to prevent unauthorized decryption, it stores encryption keys in a security module external to the database.

This section contains the following topics:

- Benefits of Using Transparent Data Encryption
- When to Use Transparent Data Encryption
- How Transparent Data Encryption Works
- Overview of Basic Transparent Data Encryption Operations

Benefits of Using Transparent Data Encryption

Transparent data encryption enables simple and easy encryption for sensitive data in columns without requiring users or applications to manage the encryption key. This freedom can be extremely important when addressing, for example, regulatory compliance issues. No need to use views to decrypt data, because the data is transparently decrypted once a user has passed necessary access control checks. Security administrators have the assurance that the data on disk is encrypted, yet handling encrypted data becomes transparent to applications.

When to Use Transparent Data Encryption

Use transparent data encryption to protect confidential data such as credit card and social security numbers without having to manage key storage or create auxiliary tables, views, and triggers. An application that processes sensitive data can use this feature to provide strong data encryption with little or no change to the application.

Do not, however, use transparent data encryption with these database features:

- Index types other than B-tree

- Range scan search through an index

- Large object datatypes such as `BLOB` and `CLOB`

- Original import/export utilities

- Other database tools and utilities that directly access data files

Applications that need to use these unsupported features can use the `DBMS_CRYPTO` package for their encryption needs.

> **Note:** Enabling encryption on an existing table results in a full table update like any other ALTER TABLE operation that modifies table characteristics. Administrators should keep in mind the potential performance and redo log impact on the database server before enabling encryption on a large existing table.

How Transparent Data Encryption Works

Transparent data encryption is a key-based access control system. Even if the encrypted data is retrieved, it cannot be understood until authorized decryption occurs, which is automatic for users authorized to access the table.

When a table contains encrypted columns, a single key is used regardless of the number of encrypted columns. The keys for all tables containing encrypted columns are encrypted with the database server master key and stored in a dictionary table in the database. No keys are stored in the clear.

As shown in Figure 3–1, the master key of the server is stored in an external security module that is outside the database and accessible only to the security administrator. For this external security module, Oracle uses an Oracle wallet as described in this chapter. Storing the master key in this way prevents its unauthorized use. In addition to storing the master key, the Oracle wallet is also used to generate encryption keys and perform encryption and decryption.

Figure 3–1   Transparent Data Encryption Overview
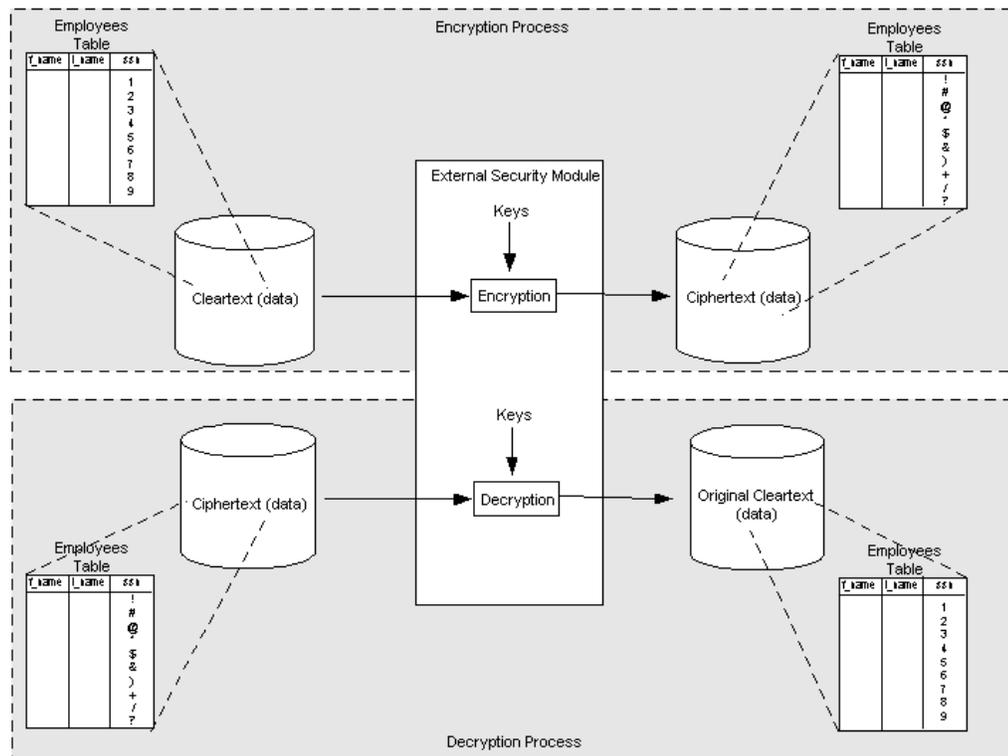
Using an external security module separates ordinary program functions from encryption operations, making it possible to divide duties between database administrators and security administrators. Security is enhanced because no single administrator is granted complete access to all data.

> **See Also:**
>
> - "Using Transparent Data Encryption" on page 3-4 for detailed information about the SQL syntax used for transparent data encryption.
>
> - "Supported Encryption and Integrity Algorithms" on page 3-11 for a list of the algorithms this feature supports.

Overview of Basic Transparent Data Encryption Operations

To use transparent data encryption, you must have the `ALTER SYSTEM` privilege and a valid password to the Oracle wallet. If an Oracle wallet does not exist, then a new one is created using the password specified in the SQL command.

> **See Also:** "Oracle Wallet Management for Transparent Data Encryption" on page 3-12

To create a new master key and begin using transparent data encryption, issue the following command:

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY password
```

This command generates the database server master encryption key, which the server uses to encrypt the column encryption key for each table. No table columns in the database can be encrypted until the master key of the server has been set.

A master key that has been set remains accessible to the database until the database instance is shutdown. To load the master key after the database is restarted, use the following command:

```
ALTER SYSTEM SET WALLET OPEN IDENTIFIED BY password
```

To create a new table with encrypted columns, use the CREATE TABLE command in the following form:

```
CREATE TABLE table_name ( column_name column_type ENCRYPT,....);
```

The ENCRYPT keyword against a column specifies that the column should be encrypted.

If an existing table has columns that require encryption, then use the ALTER TABLE command in the following form:

```
ALTER TABLE table_name MODIFY ( column_name column_type ENCRYPT,...);
```

The ENCRYPT keyword against a column specifies that the column should be encrypted.

To disable access to all encrypted columns in the database, use the following command:

```
ALTER SYSTEM SET WALLET CLOSE
```

The command disables access to the master key in the wallet and prevents access to data in the encrypted columns.

> **See Also:**
>
> - Examples of ALTER SYSTEM commands in Example 3–1 on page 3-5 and Example 3–2 on page 3-7.
>
> - Examples of ALTER TABLE commands in Example 3–8 on page 3-9, Example 3–9 on page 3-9, and Example 3–10 on page 3-10.
>
> - Examples using SQL for encryption include Example 3–5 on page 3-8, Example 3–6 on page 3-8, and Example 3–7 on page 3-9.

Using Transparent Data Encryption

Only a small number of steps are needed to configure and use this feature. This section contains these topics:

- Enabling Transparent Data Encryption

- Opening the Encrypted Wallet for Database Access to Encryption Keys

- Setting and Resetting the Master Key

- Creating Tables That Contain Encrypted Columns

- Specifying Columns for Encryption in Existing Tables

- Creating an Index on an Encrypted Column

- Adding or Removing Salt from an Encrypted Column

- Changing the Encryption Key or Algorithm on Tables Containing Encrypted Columns

Enabling Transparent Data Encryption

Before transparent data encryption can be enabled, the compatibility level for the database must be 10.2. To start using transparent data encryption, the security administrator must create a wallet and set a master key.

The wallet can be the default database wallet shared with other Oracle Database components or a separate wallet specifically used by transparent data encryption. Oracle recommends that, for greater security, a separate wallet be used to store transparent data encryption master keys.

> **See Also:** "Oracle Wallet Management for Transparent Data Encryption" on page 3-12 for more information on configuring transparent data encryption to use a separate wallet

Opening the Encrypted Wallet for Database Access to Encryption Keys

The external security module stores the encryption keys in an Oracle wallet. The database must load the master and column encryption keys into memory from the wallet before it can encrypt or decrypt columns. Use the following `ALTER SYSTEM` command to explicitly open the wallet:

```
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY password
```

where `password` is a string value specified in `sqlnet.ora` as the password to the wallet.

Once the wallet has been opened, it remains open until you shut down the database instance (or close it explicitly by issuing an `ALTER SYSTEM SET ENCRYPTION WALLET CLOSE` command). When you restart the instance, you must issue the `ALTER SYSTEM SET ENCRYPTION WALLET OPEN` command again.

If the schema does not have the `ALTER SYSTEM` privilege, or the wallet is unavailable, or an incorrect password is given, then the command returns an error and exits. If the wallet is already open, the command returns an error and takes no action. Example 3–1 shows an example of each usage case.

Example 3–1   Opening the External Security Module Wallet with ALTER SYSTEM

```
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "U83j10LLt8v";
Wallet opened.
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "U83j10LLt8v";
ORA-XXXXX: Wallet already opened.
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "U93j10LLt8v";
ORA-XXXXX: Incorrect wallet password given.
```

Using Wallets with Automatic Login Enabled for Transparent Data Encryption

The external security module can use wallets with the automatic login feature enabled. These wallets remain open all the time. The security administrator does not have to reopen the wallet after a database instance has been restarted. If your environment does not require the extra security provided by a wallet that must be explicitly opened for use, then you may use an autologin wallet.

> **See Also:** "Using Auto Login" on page 9-14 for information about autologin wallets and how to use them.

Specifying an Additional Wallet Location in SQLNET.ORA

By default, the external security module stores encryption keys in the Oracle wallet specified in the `sqlnet.ora` configuration file. If no wallet location is specified in the `sqlnet.ora` file, then the default database wallet is used.

If you wish to use a wallet specifically for transparent data encryption, then you must specify a second wallet location in `sqlnet.ora` by using the `ENCRYPTION_WALLET_LOCATION` parameter.

> **See Also:** "Sample sqlnet.ora File" on page A-1for an example of the syntax used to set this parameter

Setting and Resetting the Master Key

The master key is stored in the external security module and is used to protect column encryption keys. By default, the master key is a random key generated by transparent data encryption. It can also be an existing key pair from a PKI certificate designated for encryption. To use transparent data encryption with PKI key pairs, the issuing certificate authority must be able to issue X.509v3 certificates with the key usage field marked for encryption.

Neither key type is more secure, but if you have already deployed PKI within your organization, then you can leverage such PKI services as key escrow and recovery. However, encryption using current PKI algorithms requires significantly more system resources than symmetric key encryption. Using a PKI key pair as a master key may result in greater performance degradation when accessing encrypted columns in the database.

To set or reset the master key, you use the same `ALTER SYSTEM` command. The following sections explain how and why to perform each of these tasks.

Setting the Master Key for First Time Use of Transparent Data Encryption

Before encrypting any database columns, you must generate or set a master key. This master key is used to encrypt the column encryption key that is generated automatically when you issue a SQL command with the `ENCRYPT` clause on a database column.

To set the master key, use the following SQL syntax:

```
ALTER SYSTEM SET ENCRYPTION KEY certificate_ID IDENTIFIED BY password
```

where

- `certificate_ID` is an optional string containing the unique identifier of a certificate stored in the security module. Use this parameter if you intend to use your PKI private key as your master key. This parameter has no default setting.

  You can search for a `certificate_ID` by querying the `V$WALLET` fixed view when the wallet is open. Only certificates that can be used as master keys by transparent data encryption are shown.

- `password` is the mandatory wallet password for the security module, with no default setting. It is case sensitive; enclose it in double-quotation marks.

  > **See Also:** *Oracle Database SQL Reference* for the SQL rules for supplying passwords.

If no wallet exists, then this command creates a new one at the wallet location specified in the `sqlnet.ora` parameter file. If no wallet location is specified in the

`sqlnet.ora` file, then the default database wallet location is used. If an existing autologin wallet is present at the expected wallet location, then a new wallet is not created.

Resetting the Master Key

Regenerate the master key only if it has been compromised. Frequent master key regeneration does not necessarily enhance system security. Security modules can store a large, but not infinite, number of keys, and frequent master key regeneration can exhaust all the available storage space.

To reset the master key, use the SQL syntax as shown in "Setting the Master Key for First Time Use of Transparent Data Encryption" on page 3-6.

Note that the `ALTER SYSTEM SET ENCRYPTION KEY` command is a DDL command requiring the `ALTER SYSTEM` privilege, and it automatically commits any pending transactions. Example 3–2 shows a sample usage of this command.

Example 3–2   Setting or Resetting the Master Key To Use a PKI-Based Private Key

```
ALTER SYSTEM SET ENCRYPTION KEY "j23lm781098dhb345sm" IDENTIFIED BY "p3812dH9E";
```

Note that for PKI-based keys, **certificate revocation lists** are not enforced because enforcing certificate revocation may lead to losing access to all encrypted information in the database.

Adding or Removing Salt from an Encrypted Column

Salt is a way to strengthen the security of encrypted data. It is a random string added to the data before it is encrypted, causing repetition of text in the clear to appear different when encrypted. Salt thus removes one method attackers use to steal data, namely, matching patterns of encrypted text.

To add or remove salt from encrypted columns, you again use the `ALTER TABLE MODIFY` command with either the `SALT` or `NO SALT` parameter specified with the `ENCRYPT` clause. Example 3–3 and Example 3–4 illustrate such commands.

Example 3–3   Adding Salt to an Encrypted Column

```
ALTER TABLE employee MODIFY (first_name ENCRYPT SALT);
```

Example 3–4   Removing Salt from an Encrypted Column

```
ALTER TABLE employee MODIFY (first_name ENCRYPT NO SALT);
```

To remove salt from an encrypted column before indexing it, use the syntax shown in Example 3–4. This command also re-encrypts the data.

Creating Tables That Contain Encrypted Columns

To create relational tables with encrypted columns, specify the SQL `ENCRYPT` clause when you define database columns with the `CREATE TABLE` statement.

This section contains the following topics:

- Creating a Table with an Encrypted Column That Uses the Default Algorithm
- Creating a Table with an Encrypted Column Using a Non-Default Algorithm and No Salt
- Creating an Encrypted Column on an External Table

Creating a Table with an Encrypted Column That Uses the Default Algorithm

By default, transparent data encryption uses AES with a 192-bit length key (AES192). If you use the ENCRYPT clause with no other modifiers, the column you specify will be encrypted by using AES192 as shown in Example 3–5.

Example 3–5   Creating a New Table with an Encrypted Column Using the Default Algorithm (AES192)

```
CREATE TABLE employee (
     first_name VARCHAR2(128),
     last_name VARCHAR2(128),
     empID NUMBER,
     salary NUMBER(6) ENCRYPT
);
```

Creating a Table with an Encrypted Column Using a Non-Default Algorithm and No Salt

By default, transparent data encryption adds **salt** to cleartext before encrypting it. This makes it harder for attackers to steal the data through a brute force attack.

However, if you plan to index the encrypted column, you must use NO SALT. Example 3–6 shows how to specify the NO SALT parameter with the SQL ENCRYPT clause (empID NUMBER ENCRYPT NO SALT). It also shows the syntax for specifying a different encryption algorithm (salary NUMBER(6) ENCRYPT USING '3DES168'). Note that the string which specifies the algorithm must be enclosed in single quotation marks.

Example 3–6   Creating a New Table with an Encrypted Column Using 3DES168 and NO SALT

```
CREATE TABLE employee (
     first_name VARCHAR2(128),
     last_name VARCHAR2(128),
     empID NUMBER ENCRYPT NO SALT,
     salary NUMBER(6) ENCRYPT USING '3DES168'
);
```

Creating an Encrypted Column on an External Table

The external table database feature enables you to access data in external sources as if it were in a table in the database. External tables can be updated using the ORACLE_DATAPUMP access driver. Otherwise, they are read-only.

> **See Also:**   *Oracle Database Concepts* for discussions of Schema Objects and Tables.

To encrypt specific columns in an external table, you specify the ENCRYPT clause when you define those columns. That specification causes a randomly generated key to be used to encrypt the columns.

However, if you intend to move your external table, that key will not be available in the new location. For such a table, you should specify your own password to encrypt the columns. Then, after you move the data, use the same password to regenerate the key so you can access encrypted column data in the new location.

Table partition exchange also requires a password-based column encryption key.

Example 3–7   Creating a New External Table with a Password-Generated Column Key

```
CREATE TABLE emp_ext (
    first_name,
    last_name,
    empID,
    salary,
    ssn ENCRYPT IDENTIFIED BY "xIcf3T9u"
)  ORGANIZATION EXTERNAL
   (
    TYPE ORACLE_DATAPUMP
    DEFAULT DIRECTORY "D_DIR"
    LOCATION('emp_ext.dat')
    )
    REJECT LIMIT UNLIMITED
as select * from employee;
```

> **See Also:**   *Oracle Database SQL Reference* about CREATE TABLE, ENCRYPT, and the rules for passwords.

Specifying Columns for Encryption in Existing Tables

To add an encrypted column to an existing table, or to encrypt or decrypt an existing column, you use the ALTER TABLE SQL command with the ADD or MODIFY clause.

This section contains the following topics:

- Adding Encrypted Columns to Existing Tables
- Encrypting Unencrypted Columns
- Disabling Encryption on a Column

Adding Encrypted Columns to Existing Tables

To add an encrypted column to an existing table, you use the ALTER TABLE ADD command, specifying the new column with the ENCRYPT clause as shown in Example 3–8.

Example 3–8   Adding Encrypted Columns to Existing Tables

```
ALTER TABLE employee ADD (ssn VARCHAR2(11) ENCRYPT);
```

This command encrypts the new column with the default AES encryption algorithm, using a 192-bit key length (AES192) and adding salt to the cleartext data before encryption. However, if you plan to index this column use the ENCRYPT clause with the NO SALT parameter. For an example of this syntax, see "Adding or Removing Salt from an Encrypted Column" on page 3-7.

Encrypting Unencrypted Columns

To encrypt unencrypted columns, use the ALTER TABLE MODIFY command, specifying the unencrypted column with the ENCRYPT clause as shown in Example 3–9.

Example 3–9   Encrypting Unencrypted Columns

```
ALTER TABLE employee MODIFY (first_name ENCRYPT);
```

This command encrypts the column using the default AES192 algorithm, because no other encryption algorithm was specified. Salt is added by default because the NO

SALT parameter was not specified with ENCRYPT. In this respect, this example is similar to Example 3–8.

> **See Also:** "Adding or Removing Salt from an Encrypted Column" on page 3-7which shows how to specify a different encryption algorithm and no salt

If a column must be indexed, you must specify that no salt be added.

Disabling Encryption on a Column

It may be necessary to disable encryption for reasons of compatibility or performance. To disable column encryption, use the ALTER TABLE MODIFY command with the DECRYPT clause as shown in Example 3–10.

Example 3–10   Turning Off Column Encryption

```
ALTER TABLE employee MODIFY (first_name DECRYPT);
```

Creating an Index on an Encrypted Column

To create an index on an encrypted column, you use the standard CREATE INDEX command. The column being indexed must have been encrypted without salt. Example 3–11 shows how to create an index on a column that has been encrypted without salt.

Example 3–11   Creating Index on a Column Encrypted Without Salt

```
CREATE TABLE employee (
   first_name VARCHAR2(128),
   last_name VARCHAR2(128),
   empID NUMBER ENCRYPT NO SALT,
   salary NUMBER(6) ENCRYPT USING '3DES168'
);
CREATE INDEX employee_idx on employee (empID);
```

> **Note:**   You cannot create an index on a column that has been encrypted with salt. If you try to do this, an error (ORA-28338) is raised.

Changing the Encryption Key or Algorithm on Tables Containing Encrypted Columns

Each table can have at most one encryption key for its columns. This key can be changed by using either the original encryption algorithm or a different algorithm specified in the REKEY command phrase. The examples in this section illustrate these capabilities. Good security practices include backing up the wallet before and after such changes.

Example 3–12   Changing the Encryption Key on Tables Containing Encrypted Columns

```
ALTER TABLE employee REKEY;
```

Example 3–13   Changing the Encryption Key and Algorithm on Tables Containing Encrypted Columns

```
ALTER TABLE employee REKEY USING '3DES168';
```

See Also: *Oracle Database SQL Reference* about `ALTER TABLE`

Supported Encryption and Integrity Algorithms

By default, transparent data encryption uses the Advanced Encryption Standard with a 192-bit length cipher key (AES192). In addition, **salt** is added by default to cleartext before encryption unless specified otherwise. Note that salt cannot be added to indexed columns that you want to encrypt. For indexed columns, choose the `NO SALT` parameter for the SQL `ENCRYPT` clause.

You can change encryption algorithms and encryption keys on existing encrypted columns by setting a different algorithm with the SQL `ENCRYPT` clause.

See Also:

- Example 3–6 on page 3-8 for the correct syntax when choosing the `NO SALT` parameter for the SQL `ENCRYPT` clause

- "Changing the Encryption Key or Algorithm on Tables Containing Encrypted Columns" on page 3-10 for syntax examples when setting a different algorithm with the SQL `ENCRYPT` clause

Table 3–1 lists the supported encryption algorithms.

Table 3–1   Supported Encryption Algorithms for Transparent Data Encryption

Algorithm	Key Size	Parameter Name
Triple DES (Data Encryption Standard)	168 bits	`3DES168`
AES (Advanced Encryption Standard)	128 bits	`AES128`
AES	192 bits (default)	`AES192`
AES	256 bits	`AES256`

For integrity protection, the `SHA-1` hashing algorithm is used.

Datatypes That Can Be Encrypted with the Transparent Data Encryption Feature

The following datatypes can be encrypted using this feature:

- `CHAR`
- `DATE`
- `INTERVAL DAY TO SECOND`
- `INTERVAL YEAR TO MONTH`
- `NCHAR`
- `NUMBER`
- `NVARCHAR2`
- `RAW`
- `TIMESTAMP` (includes `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP WITH LOCAL TIME ZONE`)
- `VARCHAR2`

Quick Reference: Transparent Data Encryption SQL Commands

Table 3–2 provides a summary of the SQL commands you can use to implement and manage transparent data encryption.

Table 3–2   Transparent Data Encryption SQL Commands Quick Reference

Task	SQL Command
Add encrypted column to existing table	`ALTER TABLE table_name ADD (column_name datatype ENCRYPT);`
Create table and encrypt column	`CREATE TABLE <table_name> (column_name datatype ENCRYPT);`
Encrypt unencrypted existing column	`ALTER TABLE table_name MODIFY (column_name ENCRYPT);`
Master key: set or reset	`ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY <password>;`
Master key: set or reset to use PKI certificate	`ALTER SYSTEM SET ENCRYPTION KEY <certificate_ID> IDENTIFIED BY <password>;`
Wallet: open to access master keys	`ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY <password>;`

Managing Transparent Data Encryption

This section contains these topics:

- Oracle Wallet Management for Transparent Data Encryption

- Backup and Recovery of Master Keys

- Export and Import of Tables with Encrypted Columns

- Performance Effects of Transparent Data Encryption

- Security Considerations for Using Transparent Data Encryption

- Transparent Data Encryption and Replication in Distributed Environments

- Transparent Data Encryption with OCI

- Transparent Data Encryption Data Dictionary Views

Oracle Wallet Management for Transparent Data Encryption

To store the master keys, transparent data encryption can use either the default database wallet shared by all Oracle components or a separate wallet. The wallet can be an autologin wallet that allows access to encrypted data without requiring a security administrator to explicitly open the wallet.

Creating Wallets

To create wallets used by transparent data encryption, you use the `ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY password` command. If no wallet exists in the default or specified locations, transparent data encryption creates a wallet when setting the master key for the first time. A wallet is not be created if the `WALLET_LOCATION` parameter in the `sqlnet.ora` file does not specify a valid path. The password specified in the SQL command for setting the master key becomes the password to open the wallet.

In addition to the SQL command, you can also use the `mkwallet` command-line utility and Oracle Wallet Manager to create wallets. These are full-featured tools that allow you to create wallets and to view and modify their content.

> **See Also:** Chapter 9, "Using Oracle Wallet Manager" for more information about Oracle Wallet Manager

Using an Autologin Wallet

You can create an autologin wallet with the `mkwallet` utility or Oracle Wallet Manager. The autologin wallet allows convenient access to encrypted data across database instance restarts.

Transparent data encryption uses an autologin wallet only if it is available at the correct location and the SQL command to open an encrypted wallet has not already been executed. If an autologin wallet is being used, you must not use the `ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY password` command.

Specifying a Separate Wallet for Transparent Data Encryption

By default, transparent database encryption uses the default database wallet or the wallet specified by the `WALLET_LOCATION` parameter in the `sqlnet.ora` configuration file. Because this wallet can be shared by other Oracle components, Oracle recommends that a separate wallet be used exclusively for storing master keys used by transparent data encryption. To designate a separate wallet, set the `ENCRYPTION_WALLET_LOCATION` parameter in the `sqlnet.ora` file to point to the wallet used exclusively by transparent data encryption.

When determining which wallet to use, transparent data encryption first attempts to use the wallet specified by the parameter `ENCRYPTION_WALLET_LOCATION`. If the parameter is not set, or no wallet is found at the specified location, then it attempts to use the wallet specified by the parameter `WALLET_LOCATION`. If this fails as well, then transparent data encryption looks for a wallet at the default database location.

Backup and Recovery of Master Keys

This section contains the following topics:

- Backup and Recovery of Oracle Wallet
- Backup and Recovery of PKI Key Pair

Backup and Recovery of Oracle Wallet

Because the master keys are required to access any encrypted data, they must be properly backed up. This means that, because master keys reside in an Oracle wallet, the wallet should be periodically backed up in a secure location along with the database data files. You must back up a copy of the wallet whenever a new master key is set.

If you loose the wallet that stores the master key, you can restore access to encrypted data by copying the backed-up version of the wallet to the appropriate location. If the restored wallet was archived after the last time the master key was reset, then no additional action needs to be taken.

If the restored wallet does not contain the most recent master key, then you can recover old data up to the point when the master key was reset by rolling back the state of the database to that point in time. All modifications to encrypted columns after the master key was reset are lost.

Backup and Recovery of PKI Key Pair

Transparent data encryption supports the use of PKI asymmetric key pairs as master keys. This enables it to leverage existing key backup, escrow, and recovery facilities from leading certificate authority vendors.

In current key escrow or recovery systems, the certificate authority with key recovery capabilities typically stores a version of the private key or a piece of information that helps recover the private key. If the private key is lost, the user can recover the original key and certificate by contacting the certificate authority and initiating a key recovery process.

Typically, the key recovery process is automated and requires the user to present certain authenticating credentials to the certificate authority. Transparent data encryption puts no restrictions on the key recovery process other than that the recovered key and its associated certificate be a PKCS#12 file that can be imported into an Oracle wallet. This requirement is consistent with the key recovery mechanisms of leading certificate authorities.

After obtaining the PKCS#12 file with the original certificate and private key, you need to create a new empty wallet in the same location as the previous wallet. To do this, you can use the `mkwallet` command line utility or Oracle Wallet Manager. You can then import the PKCS#12 file into the wallet by using the same utility. You should choose a strong password to protect the wallet.

After the wallet has been created and the correct certificates imported, log onto the database and execute the following command at the SQL prompt to complete the recovery process:

```
ALTER SYSTEM SET ENCRYPTION KEY certificate_id IDENTIFIED BY wallet_password
```

To retrieve the `certificate_id` of the certificate in the wallet, query the `V$WALLET` fixed view after the wallet has been opened.

Export and Import of Tables with Encrypted Columns

When exporting tables containing encrypted columns, it is important that:

- The sensitive data remain unintelligible during transport

- Authorized users can decrypt that data after it is imported at the destination

Because the key for decryption is local to the server where the tables originally reside, decryption at the destination will not be possible using that key. Consequently, prior to exporting, the administrator re-keys the table(s) with a password key, which he then securely provides to the destination administrator.

Upon import, the destination administrator specifies the same password. The affected columns being imported are decrypted, enabling the receiving server to immediately re-encrypt those columns with a local server key. They are then ready for standard authorized use in their new location.

Performance Effects of Transparent Data Encryption

This feature affects performance *only* when data is retrieved from or inserted into an encrypted column. *No* reduction of performance occurs for such operations on other columns, even in a table containing encrypted columns.

The total performance effect depends on the number of encrypted columns and their frequency of access. The columns most appropriate for encryption are those containing the most sensitive data, including regulatory mandates.

Enabling transparent data encryption on columns in an existing table results in a full table update on all rows in the column as it encrypts all data stored within the column. This may cause the table to be inaccessible while encryption is being enabled. Using Online Redefinition to enable transparent data encryption allows the table to be available while it is being encrypted.

The redo log impact of a full table update on a large table should also be kept in mind. If transparent data encryption is being enabled on a very large table, then the redo log size might need to be increased to accommodate the operation.

Security Considerations for Using Transparent Data Encryption

Access to encrypted columns can be:

- Public

- Controlled by standard authentication and authorization procedures and policies

- Controlled by individual users who can grant access to others

Security considerations for transparent data encryption operate within the broader arena of total system security. Security administrators must identify the levels of risk to be addressed and the degrees of sensitivity in data maintained by the site. Costs and benefits must be evaluated for the alternative methods of achieving acceptable protections. In many cases, it makes sense to have separate security administrators, separate wallets, and protected backup procedures for encrypted materials. Having separate wallets permits auto-login for other Oracle components but preserves password protection for the wallet enabling transparent data encryption.

Additional security considerations apply to normal database and network operations when transparent data encryption is in use. Encrypted column data stays encrypted in the data files, undo logs, and redo logs, as well as in the buffer cache of the system global area (SGA). However, during expression evaluation it is decrypted, making it possible for decrypted data to appear in the swap file on the disk, potentially visible to privileged operating system users.

Transparent Data Encryption and Replication in Distributed Environments

In general, the methods and protections vary with the choice of the tool by which replication is accomplished. SQL*Loader, Dataguard, and import/export are methods to enable transporting tables with encrypted columns. They rely on using a password created precisely to protect the data during transmission. When the data is reconstituted at the receipt-point, the columns marked as encrypted are immediately re-keyed with a master key local to the new table location.

> **See Also:** Manuals for SQL*Loader, Dataguard, and import/export.

When asynchronous SQL-based replication is used, information describing the update to be performed is encoded (but not encrypted) in a LOB and can remain in the deferred queue for some time. While in the deferred queue, the data is not protected. To minimize this risk, configure replication so that encrypted table information does not remain in the deferred queue for a long time.

Transparent Data Encryption with OCI

"Row-shipping" cannot be used, because the key to make the row usable is not available at the receipt-point.

Transparent Data Encryption Data Dictionary Views

The following three data dictionary views maintain information about the encryption algorithms used to encrypt columns:

- `DBA_ENCRYPTED_COLUMNS`

- `ALL_ENCRYPTED_COLUMNS`

  Shows the algorithm used to encrypt columns for all tables that are accessible to a particular user.

- `USER_ENCRYPTED_COLUMNS`

  Shows the algorithm used to encrypt columns for all tables in a particular user's schema.

> **See Also:** *Oracle Database Reference* for a full description of these data dictionary views.

4

Configuring Network Data Encryption and Integrity for Oracle Servers and Clients

This chapter describes how to configure native Oracle Net Services data **encryption** and **integrity** for Oracle Advanced Security. It contains the following topics:

- Oracle Advanced Security Encryption
- Oracle Advanced Security Data Integrity
- Diffie-Hellman Based Key Management
- How To Configure Data Encryption and Integrity

Oracle Advanced Security Encryption

This section describes data encryption algorithms available in the current release of Oracle Advanced Security:

- About Encryption
- Advanced Encryption Standard
- DES Algorithm Support
- Triple-DES Support
- RSA RC4 Algorithm for High Speed Encryption

Note: Prior to Release 8.1.7, Oracle Advanced Security provided three editions: Domestic, Upgrade, and Export each with different key lengths. This release now contains a complete complement of the available encryption algorithms and key lengths, previously only available in the Domestic edition. Users deploying prior versions of the product can obtain the Domestic edition for a specific product release.

About Encryption

The purpose of a secure cryptosystem is to convert **plaintext** data into unintelligible **ciphertext** based on a key, in such a way that it is very hard (computationally infeasible) to convert ciphertext back into its corresponding plaintext without knowledge of the correct key. In a symmetric cryptosystem, the same key is used both for encryption and decryption of the same data. Oracle Advanced Security provides the Advanced Encryption Standard (AES), DES, 3DES, and RC4 symmetric cryptosystems for protecting the confidentiality of Oracle Net Services traffic.

Advanced Encryption Standard

In this release, the new Federal Information Processing Standard (FIPS) encryption algorithm, Advanced Encryption Standard (AES), is supported. AES can be used by all U.S. government organizations and businesses to protect sensitive data over a network. This encryption algorithm defines three standard key lengths, which are 128-bit, 192-bit, and 256-bit. All versions operate in outer **Cipher Block Chaining (CBC)** mode.

DES Algorithm Support

Oracle Advanced Security provides the Data Encryption Standard (DES) algorithm. DES has been a U.S. government standard for many years and is sometimes mandated in the financial services industry. Because it has been a standard for so long, DES is deployed throughout the world for use in a wide variety of applications.

Triple-DES Support

Oracle Advanced Security supports Triple-DES encryption (3DES), which encrypts message data with three passes of the DES algorithm. 3DES provides a high degree of message security, but with a performance penalty. The magnitude of the performance penalty depends on the speed of the processor performing the encryption. 3DES typically takes three times as long to encrypt a data block when compared to the standard DES algorithm.

3DES is available in two-key and three-key versions, with effective key lengths of 112-bits and 168-bits, respectively. Both versions operate in outer **Cipher Block Chaining (CBC)** mode.

DES40 Algorithm

The DES40 algorithm, available in every release of Oracle Advanced Security, Oracle Advanced Networking Option, and Secure Network Services, is a variant of DES in which the secret key is preprocessed to provide 40 effective key bits. It was designed to provide DES-based encryption to customers outside the U.S. and Canada at a time when the U.S. export laws were more restrictive. Now, in Oracle Advanced Security 10*g* Release 2 (10.2), DES40, DES, and 3DES are all available for export. DES40 is still supported to provide backward-compatibility for international customers.

RSA RC4 Algorithm for High Speed Encryption

The RC4 algorithm, developed by RSA Data Security Inc., has become the international standard for high-speed data encryption. RC4 is a variable key-length stream cipher that operates at several times the speed of DES, making it possible to encrypt large, bulk data transfers with minimal performance consequences.

Oracle Advanced Security 10*g* Release 2 (10.2) provides an RC4 implementation with 40-bit, 56-bit, 128-bit, and 256-bit key lengths. This provides backward-compatibility and strong encryption, with no material performance compromise.

> **See Also:**
>
> - "Configuring Encryption on the Client and the Server" on page 4-7
>
> - Table 4–3, " Valid Encryption Algorithms" on page 4-8

Oracle Advanced Security Data Integrity

Encryption of network data provides data privacy so that unauthorized parties are not able to view plaintext data as it passes over the network. Oracle Advanced Security also provides protection against two forms of active attack. Table 4–1 provides information about these attacks.

Table 4–1   Two Forms of Attack

Type of Attack	Explanation
Data modification attack	An unauthorized party intercepting data in transit, altering it, and retransmitting it is a data modification attack. For example, intercepting a $100 bank deposit, changing the amount to $10,000, and retransmitting the higher amount is a data modification attack.
Replay attack	Repetitively retransmitting an entire set of valid data is a replay attack, such as intercepting a $100 bank withdrawal and retransmitting it ten times, thereby receiving $1,000.

Data Integrity Algorithms Supported

Oracle Advanced Security lets you select a keyed, sequenced implementation of the Message Digest 5 (MD5) algorithm or the Secure Hash Algorithm (SHA-1) to protect against both of these forms of attack. Both of these hash algorithms create a checksum that changes if the data is altered in any way. This protection operates independently from the encryption process so you can enable data integrity with or without enabling encryption.

> **See Also:**
>
> - "Configuring Integrity on the Client and the Server" on page 4-8
>
> - Table 4–4, " Valid Integrity Algorithms" on page 4-10

Diffie-Hellman Based Key Management

The secrecy of encrypted data depends upon the existence of a secret key shared between the communicating parties. A key is a secret exclusively shared by parties on both sides of a connection. Without the key, it is extremely difficult (computationally infeasible) to decrypt an encrypted message or to alter a cryptographic, checksummed message without detection. Providing and maintaining such secret keys is referred to as key management.

Secure key distribution is difficult in a multiuser environment. Oracle Advanced Security uses the well known **Diffie-Hellman key negotiation algorithm** to perform secure key distribution for both encryption and data integrity.

When encryption is used to protect the security of encrypted data, keys must be changed frequently to minimize the effects of a compromised key. Accordingly, the Oracle Advanced Security key management function changes the session key with every session.

Authentication Key Fold-in

The purpose of Authentication Key Fold-in is to defeat a possible third-party attack (historically called the *man-in-the-middle attack*) on the Diffie-Hellman key negotiation.

It strengthens the session key significantly by combining a shared secret, known only to the client and the server, with the original session key negotiated by Diffie-Hellman.

The client and the server begin communicating using the session key generated by Diffie-Hellman. When the client authenticates to the server, they establish a shared secret that is only known to both parties. Oracle Advanced Security combines the shared secret and the Diffie-Hellman session key to generate a stronger session key designed to defeat a man-in-the-middle attack.

Note: The authentication key fold-in function is an imbedded feature of Oracle Advanced Security and requires no configuration by the system or network administrator.

How To Configure Data Encryption and Integrity

This section describes how to configure Oracle Advanced Security native Oracle Net Services encryption and integrity and presumes the prior installation of Oracle Net Services.

The network or security administrator sets up the encryption and integrity configuration parameters. The profile on client and server systems using data encryption and integrity (`sqlnet.ora` file) must contain some or all of the parameters listed in this section, under the following topics:

- About Activating Encryption and Integrity
- About Negotiating Encryption and Integrity
- Setting the Encryption Seed (Optional)
- Configuring Encryption and Integrity Parameters Using Oracle Net Manager

See Also: Chapter 8, "Configuring Secure Sockets Layer Authentication", to configure the SSL feature for encryption, integrity, and authentication

About Activating Encryption and Integrity

In any network connection, it is possible for both the client and server to support more than one encryption algorithm and more than one integrity algorithm. When a connection is made, the server selects which algorithm to use, if any, from those algorithms specified in the `sqlnet.ora` files.

The server searches for a match between the algorithms available on both the client and the server, and picks the first algorithm in its own list that also appears in the client list. If one side of the connection does not specify an algorithm list, all the algorithms installed on that side are acceptable. The connection fails with error message `ORA-12650` if either side specifies an algorithm that is not installed.

Encryption and integrity parameters are defined by modifying a `sqlnet.ora` file on the clients and the servers on the network.

You can choose to configure any or all of the available Oracle Advanced Security encryption algorithms (Table 4–3), and either or both of the available integrity algorithms (Table 4–4). Only one encryption algorithm and one integrity algorithm are used for each connect session.

> **Note:** Oracle Advanced Security selects the first encryption algorithm and the first integrity algorithm enabled on the client and the server. Oracle Corporation recommends that you select algorithms and key lengths in the order in which you prefer negotiation, choosing the strongest key length first.

> **See Also:** Appendix A, "Data Encryption and Integrity Parameters"

About Negotiating Encryption and Integrity

To negotiate whether to turn on encryption or integrity, you can specify four possible values for the Oracle Advanced Security encryption and integrity configuration parameters. The four values are listed in the order of increasing security. The value `REJECTED` provides the *minimum* amount of security between client and server communications, and the value `REQUIRED` provides the *maximum* amount of network security:

- REJECTED
- ACCEPTED
- REQUESTED
- REQUIRED

The default value for each of the parameters is `ACCEPTED`.

REJECTED

Select this value if you do not elect to enable the security service, even if required by the other side.

In this scenario, this side of the connection specifies that the security service is not permitted. If the other side is set to REQUIRED, the connection *terminates* with error message `ORA-12650`. If the other side is set to REQUESTED, ACCEPTED, or REJECTED, the connection continues without error and without the security service enabled.

ACCEPTED

Select this value to enable the security service if required or requested by the other side.

In this scenario, this side of the connection does not require the security service, but it is enabled if the other side is set to REQUIRED or REQUESTED. If the other side is set to REQUIRED or REQUESTED, and an encryption or integrity algorithm match is found, the connection continues without error and with the security service enabled. If the other side is set to REQUIRED and no algorithm match is found, the connection terminates with error message `ORA-12650`.

If the other side is set to REQUESTED and no algorithm match is found, or if the other side is set to ACCEPTED or REJECTED, the connection continues without error and without the security service enabled.

REQUESTED

Select this value to enable the security service if the other side permits it.

In this scenario, this side of the connection specifies that the security service is desired but not required. The security service is enabled if the other side specifies ACCEPTED, REQUESTED, or REQUIRED. There must be a matching algorithm available on the other side, otherwise the service is not enabled. If the other side specifies REQUIRED and there is no matching algorithm, *the connection fails.*

REQUIRED

Select this value to enable the security service or preclude the connection.

In this scenario, this side of the connection specifies that the security service *must be enabled.* The connection *fails* if the other side specifies REJECTED or if there is no compatible algorithm on the other side.

Table 4–2 shows whether the security service is enabled, based on a combination of client and server configuration parameters. If either the server or client has specified REQUIRED, the lack of a common algorithm *causes the connection to fail.* Otherwise, if the service is enabled, lack of a common service algorithm results in the service being *disabled*.

Table 4–2   Encryption and Data Integrity Negotiations

Client Setting	Server Setting	Encryption and Data Negotiation
REJECTED	REJECTED	OFF
ACCEPTED	REJECTED	OFF
REQUESTED	REJECTED	OFF
REQUIRED	REJECTED	Connection fails
REJECTED	ACCEPTED	OFF
ACCEPTED	ACCEPTED	OFF[1]
REQUESTED	ACCEPTED	ON
REQUIRED	ACCEPTED	ON
REJECTED	REQUESTED	OFF
ACCEPTED	REQUESTED	ON
REQUESTED	REQUESTED	ON
REQUIRED	REQUESTED	ON
REJECTED	REQUIRED	Connection fails
ACCEPTED	REQUIRED	ON
REQUESTED	REQUIRED	ON
REQUIRED	REQUIRED	ON

[1]   This value defaults to OFF. Cryptography and data integrity are not enabled until the user changes this parameter by using Oracle Net Manager or by modifying the `sqlnet.ora` file.

Setting the Encryption Seed (Optional)

Several seeds are used to generate a random number on the client and on the server. One of the seeds that can be used is a user-defined encryption seed. This is set with the `sqlnet.crypto_seed` parameter in the `sqlnet.ora` file. It can be 10 to 70 characters in length and changed at any time. The Diffie-Hellman key exchange uses the random numbers to generate unique session keys for every connect session.

Configuring Encryption and Integrity Parameters Using Oracle Net Manager

You can set up or change encryption and integrity parameter settings using Oracle Net Manager. This section describes the following topics:

- Configuring Encryption on the Client and the Server
- Configuring Integrity on the Client and the Server

> **See Also:**
>
> - Appendix A, "Data Encryption and Integrity Parameters", for valid encryption algorithms
> - Oracle Net Manager online help, for more detailed configuration information

Configuring Encryption on the Client and the Server

Use Oracle Net Manager to configure encryption on the client and on the server (See Also "Starting Oracle Net Manager" on page 2-2). The steps to configure Oracle Net Manager are:

1. Navigate to the Oracle Advanced Security profile (For details, refer to "Navigating to the Oracle Advanced Security Profile" on page 2-2) The Oracle Advanced Security tabbed window is displayed. (Figure 4–1):

Figure 4–1   Oracle Advanced Security Encryption Window
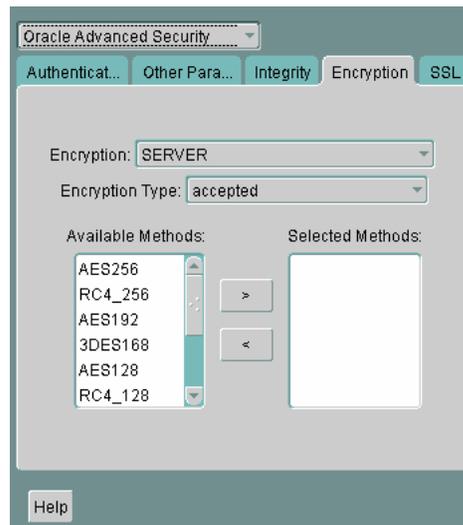
2. Click the **Encryption** tab.

3. Select **CLIENT** or **SERVER** option from the Integrity box.

4. From the Encryption Type list, select one of the following:

   - REQUESTED
   - REQUIRED
   - ACCEPTED
   - REJECTED

5. (Optional) In the **Encryption Seed** field, enter between 10 and 70 random characters. The encryption seed for the client should not be the same as that for the server.

6. Select an encryption algorithm in the **Available Methods** list. Move it to the **Selected Methods** list by choosing the right arrow (**>**). Repeat for each additional method you want to use.

7. Select **File**, **Save Network Configuration**. The `sqlnet.ora` file is updated.

8. Repeat this procedure to configure encryption on the other system. The `sqlnet.ora` file on the two systems should contain the following entries:

   - On the server:

     ```
     SQLNET.ENCRYPTION_SERVER = [accepted | rejected | requested | required]
     SQLNET.ENCRYPTION_TYPES_SERVER = (valid_encryption_algorithm [,valid_
     encryption_algorithm])
     ```

   - On the client:

     ```
     SQLNET.ENCRYPTION_CLIENT = [accepted | rejected | requested | required]
     SQLNET.ENCRYPTION_TYPES_CLIENT = (valid_encryption_algorithm [,valid_
     encryption_algorithm])
     ```

Valid encryption algorithms and their associated legal values are summarized by Table 4–3:

Table 4–3   Valid Encryption Algorithms

Algorithm Name	Legal Value
RC4 256-bit key	RC4_256
RC4 128-bit key	RC4_128
RC4 56-bit key	RC4_56
RC4 40-bit key	RC4_40
AES 256-bit key	AES256
AES 192-bit key	AES192
AES 128-bit key	AES128
3-key 3DES	3DES168
2-key 3DES	3DES112
DES 56-bit key	DES
DES 40-bit key	DES40

Configuring Integrity on the Client and the Server

Use Oracle Net Manager to configure data integrity on the client and on the server (

> **See Also:** "Starting Oracle Net Manager" on page 2-2

1. Navigate to the Oracle Advanced Security profile. (For details, refer to "Navigating to the Oracle Advanced Security Profile" on page 2-2) The Oracle Advanced Security tabbed window is displayed. (Figure 4–2):

Figure 4–2   Oracle Advanced Security Integrity Window
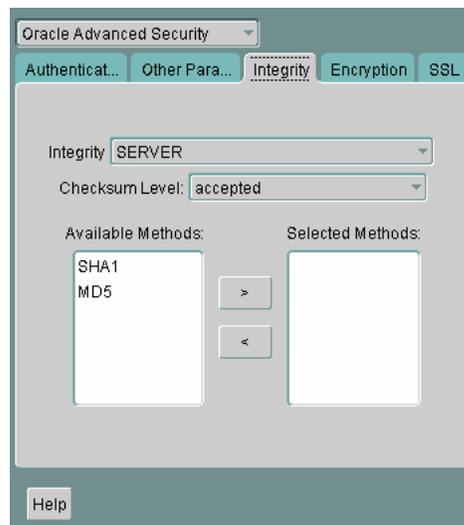
2. Click the **Integrity** tab.

3. Depending upon which system you are configuring, select the **Server** or **Client** from the **Integrity** box.

4. From the **Checksum Level** list, select one of the following checksum level values:

   - REQUESTED

   - REQUIRED

   - ACCEPTED

   - REJECTED

5. Select an integrity algorithm in the **Available Methods** list. Move it to the **Selected Methods** list by choosing the right arrow (**>**). Repeat for each additional method you want to use.

6. Select **File**, **Save Network Configuration**. The sqlnet.ora file is updated.

7. Repeat this procedure to configure integrity on the other system. The sqlnet.ora file on the two systems should contain the following entries:

   - On the server:

     ```
     SQLNET.CRYPTO_CHECKSUM_SERVER = [accepted | rejected | requested |
     required]
     SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER = (valid_crypto_checksum_algorithm
     [,valid_crypto_checksum_algorithm])
     ```

   - On the client:

     ```
     SQLNET.CRYPTO_CHECKSUM_CLIENT = [accepted | rejected | requested |
     required]
     SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT = (valid_crypto_checksum_algorithm
     [,valid_crypto_checksum_algorithm])
     ```

Valid integrity algorithms and their associated legal values are displayed in Table 4–4:

Table 4–4   Valid Integrity Algorithms

Algorithm Name	Legal Values
MD5	MD5
SHA-1	SHA1

5

Configuring Network Data Encryption and Integrity for Thin JDBC Clients

This chapter describes the Java implementation of Oracle Advanced Security, which lets thin Java Database Connectivity (JDBC) clients securely connect to Oracle Databases. This chapter contains the following topics:

- About the Java Implementation
- Configuration Parameters

> **See Also:** *Oracle Database JDBC Developer's Guide and Reference*, for information about JDBC, including examples

About the Java Implementation

The Java implementation of Oracle Advanced Security provides network encryption and integrity protection for Thin JDBC clients communicating with Oracle Databases that have Oracle Advanced Security enabled.

This section contains the following topics:

- Java Database Connectivity Support
- Securing Thin JDBC
- Implementation Overview
- Obfuscation

Java Database Connectivity Support

Java Database Connectivity (JDBC), an industry-standard Java interface, is a Java standard for connecting to a relational database from a Java program. Sun Microsystems defined the JDBC standard and Oracle Corporation implements and extends the standard with its own JDBC drivers.

Oracle JDBC drivers are used to create JDBC applications to communicate with Oracle databases. Oracle implements two types of JDBC drivers: Thick JDBC drivers built on top of the C-based Oracle Net client, as well as a Thin (Pure Java) JDBC driver to support downloadable applets. Oracle extensions to JDBC include the following features:

- Data access and manipulation
- LOB access and manipulation
- Oracle object type mapping

- Object reference access and manipulation

- Array access and manipulation

- Application performance enhancement

Securing Thin JDBC

Because the Thin JDBC driver is designed to be used with downloadable applets used over the Internet, Oracle designed a 100% Java implementation of Oracle Advanced Security encryption and integrity algorithms for use with thin clients. Oracle Advanced Security provides the following features for Thin JDBC:

- Data encryption

- Data integrity checking

- Secure connections from Thin JDBC clients to the Oracle RDBMS

- Ability for developers to build applets that transmit data over a secure communication channel

- Secure connections from middle tier servers with Java Server Pages (JSP) to the Oracle RDBMS

- Secure connections from Oracle Database 10*g* to older versions of Oracle databases with Oracle Advanced Security installed

The Oracle JDBC Thin driver implements the Oracle O3LOGON protocol for authentication. It does not support Oracle Advanced Security SSL implementation, nor does it support third party authentication features such as RADIUS, Kerberos, and SecurID. However, the Oracle JDBC OCI (thick) driver support is the same as thick client support, where all Oracle Advanced Security features are implemented.

> **Note:** Use the `jsse.jar` file provided by your platform vendors for Java SSL requests. Oracle JavaSSL is not supported starting with this release.

Oracle Advanced Security continues to encrypt and provide integrity checking of Oracle Net Services traffic between Oracle Net clients and Oracle servers using algorithms written in C. The Oracle Advanced Security Java implementation provides Java versions of the following encryption algorithms:

- RC4_256

- RC4_128

- RC4_56

- RC4_40

- 3DES112

- 3DES168

> **Note:** In Oracle Advanced Security, DES runs in Cipher Block Chaining (CBC) mode.

In addition, this implementation provides data integrity checking for Thin JDBC using Message Digest 5 (MD5), a cryptographically secure message digest.

Implementation Overview

On the server side, the negotiation of algorithms and the generation of keys function exactly the same as Oracle Advanced Security native encryption. This enables backward and forward compatibility of clients and servers.

On the client side, the algorithm negotiation and key generation occur in exactly the same manner as C-based Oracle Advanced Security encryption. The client and server negotiate encryption algorithms, generate random numbers, use Diffie-Hellman to exchange session keys, and use the Oracle Password Protocol (O3LOGON key fold-in), in the same manner as the traditional Oracle Net clients. Thin JDBC contains a complete implementation of a Oracle Net client in pure Java.

Obfuscation

Java cryptography code is *obfuscated* in this release. Obfuscation protects Java classes and methods that contain encryption and decryption capabilities with obfuscation software.

Java byte code **obfuscation** is a process frequently used to protect intellectual property written in the form of Java programs. It mixes up Java symbols found in the code. The process leaves the original program structure intact, letting the program run correctly while changing the names of the classes, methods, and variables in order to hide the intended behavior. Although it is possible to decompile and read non-obfuscated Java code, obfuscated Java code is sufficiently difficult to decompile to satisfy U.S. government export controls.

Configuration Parameters

A properties class object containing several configuration parameters is passed to the Oracle Advanced Security interface. This chapter lists the configuration parameters for the following:

- Client Encryption Level: ORACLE.NET.ENCRYPTION_CLIENT
- Client Encryption Selected List: ORACLE.NET.ENCRYPTION_TYPES_CLIENT
- Client Integrity Level: ORACLE.NET.CRYPTO_CHECKSUM_CLIENT
- Client Integrity Selected List: ORACLE.NET.CRYPTO_CHEKSUM_TYPES_CLIENT

Client Encryption Level: ORACLE.NET.ENCRYPTION_CLIENT

This parameter defines the level of security that the client wants to negotiate with the server. Table 5–1 describes this parameter's attributes.

Table 5–1   ORACLE.NET.ENCRYPTION_CLIENT Parameter Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	REJECTED; ACCEPTED; REQUESTED; REQUIRED
Default Value	ACCEPTED
Syntax	up.put("oracle.net.encryption_client",level)

Table 5–1   (Cont.) ORACLE.NET.ENCRYPTION_CLIENT Parameter Attributes

Attribute	Description
Example	`up.put("oracle.net.encryption_client", "REQUIRED"), where up is defined as Properties up=new properties()`

Client Encryption Selected List: ORACLE.NET.ENCRYPTION_TYPES_CLIENT

This parameter defines the encryption algorithm to be used. Table 5–2 describes this parameter's attributes.

Table 5–2   ORACLE.NET.ENCRYPTION_TYPES_CLIENT Parameter Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	RC4_256; RC4_128; RC4_56; RC4_40; 3DES112; 3DES168
Syntax	`up.put("oracle.net.encryption_types_ client",alg)`
Example	`up.put("oracle.net.encryption_types_client", "3DES112"), where up is defined as Properties up=new Properties()`

Note: In this context, "C" refers to CBC (Cipher Block Chaining) mode.

Client Integrity Level: ORACLE.NET.CRYPTO_CHECKSUM_CLIENT

This parameter defines the level of security that it wants to negotiate with the server for data integrity. Table 5–3 describes this parameters attributes.

Table 5–3   ORACLE.NET.CRYPTO_CHECKSUM_CLIENT Parameter Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	REJECTED; ACCEPTED; REQUESTED; REQUIRED
Default Value	ACCEPTED
Syntax	`up.put("oracle.net.crypto_checksum_ client",level)`
Example	`up.put("oracle.net.crypto_checksum_client", "REQUIRED"), where up is defined as Properties up=new Properties()`

Client Integrity Selected List: ORACLE.NET.CRYPTO_CHEKSUM_TYPES_CLIENT

This parameter defines the data integrity algorithm to be used. Table 5–4 describes this parameter's attributes.

Table 5–4   ORACLE.NET.CRYPTO_CHEKSUM_TYPES_CLIENT Parameter Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	MD5
Syntax	`up.put("oracle.net.crypto_checksum_types_` `client",alg)`

Example	`up.put("oracle.net.crypto_checksum_types_`
	`client","MD5"), where up is defined as`
	`Properties up=new Properties()`

Part III

Oracle Advanced Security Strong Authentication

This part describes how to configure strong authentication methods for the Oracle network.

Part III contains the following chapters:

6

Configuring RADIUS Authentication

This chapter describes how to configure an Oracle Database server for use with RADIUS (Remote Authentication Dial-In User Service). It contains the following topics:

- RADIUS Overview
- RADIUS Authentication Modes
- Enabling RADIUS Authentication, Authorization, and Accounting
- Using RADIUS to Log In to a Database
- RSA ACE/Server Configuration Checklist

> **Note:** SecurID, an authentication product of RSA Security, Inc., though not directly supported by Oracle Advanced Security, has been certified as RADIUS-compliant. You can therefore, run SecurID under RADIUS.
>
> Refer to the RSA Security SecurID documentation for further information.

RADIUS Overview

RADIUS is a client/server security protocol widely used to enable remote authentication and access. Oracle Advanced Security uses this industry standard in a client/server network environment.

You can enable the network to use any authentication method that supports the RADIUS standard, including token cards and smart cards, by installing and configuring the RADIUS protocol. Moreover, when you use RADIUS, you can change the authentication method without modifying either the Oracle client or the Oracle database server.

From the user's perspective, the entire authentication process is transparent. When the user seeks access to an Oracle database server, the Oracle database server, acting as the RADIUS client, notifies the RADIUS server. The RADIUS server:

- Looks up the user's security information
- Passes authentication and authorization information between the appropriate authentication server or servers and the Oracle database server
- Grants the user access to the Oracle database server
- Logs session information, including when, how often, and for how long the user was connected to the Oracle database server

> **Note:** Oracle Advanced Security does not support RADIUS authentication over database links.

The Oracle/RADIUS environment is displayed in Figure 6–1:

Figure 6–1   RADIUS in an Oracle Environment

Oracle Client   Oracle Server

Radius Client   **Radius Server**
or
RSA ACE / Server
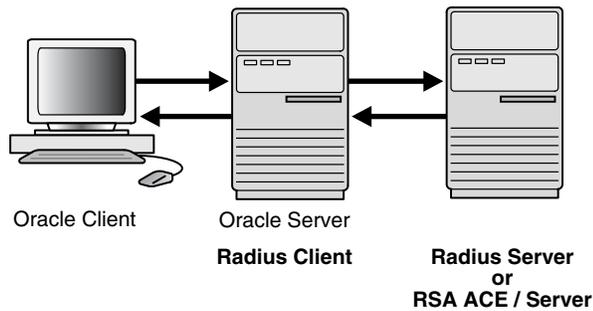
The Oracle database server acts as the RADIUS client, passing information between the Oracle client and the RADIUS server. Similarly, the RADIUS server passes information between the Oracle database server and the appropriate authentication servers. The authentication components are listed in Table 6–1:

Table 6–1   RADIUS Authentication Components

Component	Stored Information
Oracle client	Configuration setting for communicating through RADIUS.
Oracle database server/RADIUS client	Configuration settings for passing information between the Oracle client and the RADIUS server. The secret key file.
RADIUS server	Authentication and authorization information for all users. Each client's name or IP address. Each client's shared secret. Unlimited number of menu files enabling users already authenticated to select different login options without reconnecting.
Authentication server or servers	User authentication information such as pass codes and PINs, depending on the authentication method in use. **Note:** The RADIUS server can also be the authentication server.

A RADIUS server vendor is often the authentication server vendor as well. In this case authentication can be processed on the RADIUS server. For example, the RSA ACE/Server is both a RADIUS server and an authentication server. It thus authenticates the user's pass code.

> **See Also:**   *Oracle Database Net Services Administrator's Guide,* for information about the `sqlnet.ora` file

RADIUS Authentication Modes

User authentication can take place in the following ways:

■   Synchronous Authentication Mode

- Challenge-Response (Asynchronous) Authentication Mode

Synchronous Authentication Mode

In the synchronous mode, RADIUS lets you use various authentication methods, including passwords and SecurID token cards. Figure 6–2 shows the sequence in which synchronous authentication occurs:

Figure 6–2   Synchronous Authentication Sequence
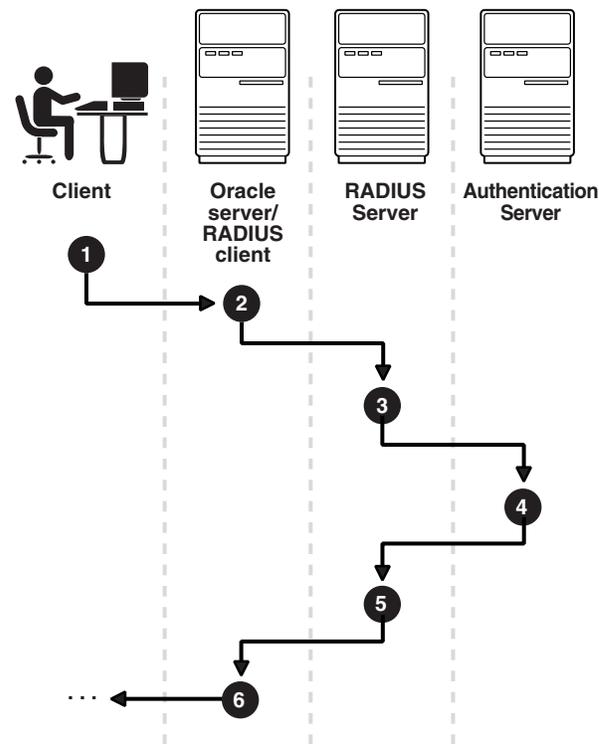
Following steps describe the Synchronous Authentication Sequence:

1. A user logs in by entering a connect string, pass code, or other value. The client system passes this data to the Oracle database server.

2. The Oracle database server, acting as the RADIUS client, passes the data from the Oracle client to the RADIUS server.

3. The RADIUS server passes the data to the appropriate authentication server, such as Smart Card or SecurID ACE for validation.

4. The authentication server sends either an Access Accept or an Access Reject message back to the RADIUS server.

5. The RADIUS server passes this response to the Oracle database server/RADIUS client.

6. The Oracle database server/RADIUS client passes the response back to the Oracle client.

Example: Synchronous Authentication with SecurID Token Cards

With SecurID authentication, each user has a token card that displays a dynamic number that changes every sixty seconds. To gain access to the Oracle database

server/RADIUS client, the user enters a valid pass code that includes both a personal identification number (PIN) and the dynamic number currently displayed on the user's SecurID card. The Oracle database server passes this authentication information from the Oracle client to the RADIUS server, which in this case is the authentication server for validation. Once the authentication server (RSA ACE/Server) validates the user, it sends an *accept* packet to the Oracle database server, which, in turn, passes it to the Oracle client. The user is now authenticated and able to access the appropriate tables and applications.

> **See Also:**
>
> - Chapter 1, "Introduction to Oracle Advanced Security"
> - "Token Cards" on page 1-8
> - Documentation provided by RSA Security, Inc.

Challenge-Response (Asynchronous) Authentication Mode

When the system uses the asynchronous mode, the user does not need to enter a user name and password at the SQL*Plus CONNECT string. Instead, a graphical user interface asks the user for this information later in the process.

Figure 6–3 shows the sequence in which challenge-response (asynchronous) authentication occurs.

> **Note:** If the RADIUS server is the authentication server, Steps 3, 4, and 5, and Steps 9, 10, and 11 in Figure 6–3 are combined.

Figure 6–3   Asynchronous Authentication Sequence
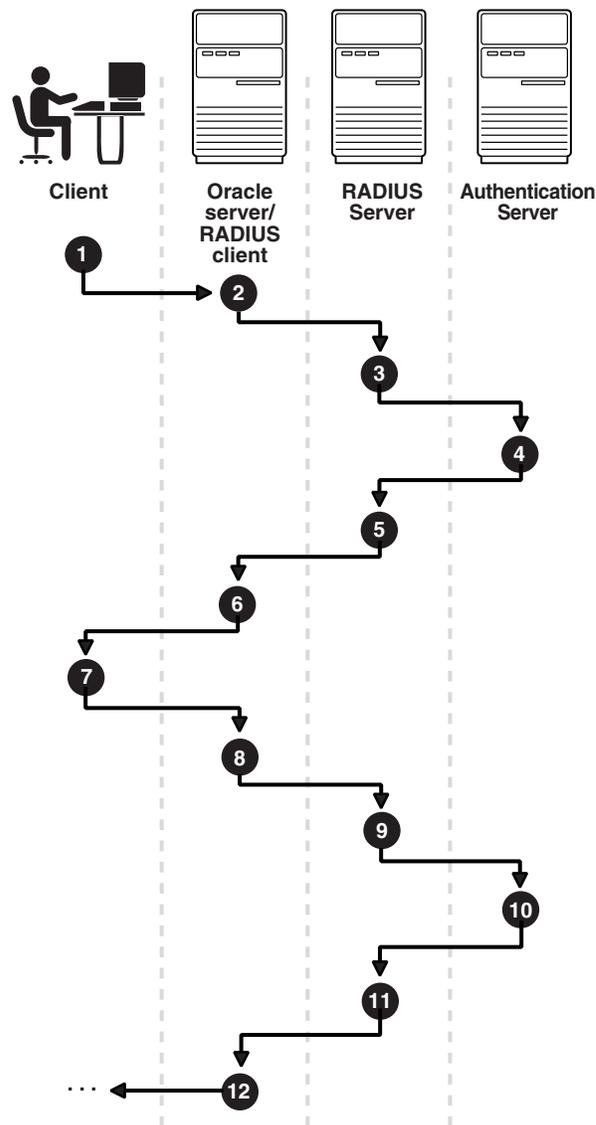
Following steps describe the Asynchronous Authentication Sequence:

1. A user seeks a connection to an Oracle database server. The client system passes the data to the Oracle database server.

2. The Oracle database server, acting as the RADIUS client, passes the data from the Oracle client to the RADIUS server.

3. The RADIUS server passes the data to the appropriate authentication server, such as a Smart Card, SecurID ACE, or token card server.

4. The authentication server sends a challenge, such as a random number, to the RADIUS server.

5. The RADIUS server passes the challenge to the Oracle database server/RADIUS client.

6. The Oracle database server/RADIUS client, in turn, passes it to the Oracle client. A graphical user interface presents the challenge to the user.

7. The user provides a response to the challenge. To formulate a response, the user can, for example, enter the received challenge into the token card. The token card provides a dynamic password that is entered into the graphical user interface. The Oracle client passes the user's response to the Oracle database server/RADIUS client.

8. The Oracle database server/RADIUS client sends the user's response to the RADIUS server.

9. The RADIUS server passes the user's response to the appropriate authentication server for validation.

10. The authentication server sends either an Access Accept or an Access Reject message back to the RADIUS server.

11. The RADIUS server passes the response to the Oracle database server/RADIUS client.

12. The Oracle database server/RADIUS client passes the response to the Oracle client.

Example: Asynchronous Authentication with Smart Cards

With smart card authentication, the user logs in by inserting the smart card, a plastic card (like a credit card) with an embedded integrated circuit for storing information into a hardware device which reads the card. The Oracle client sends the login information contained in the smart card to the authentication server by way of the Oracle database server/RADIUS client and the RADIUS server. The authentication server sends back a challenge to the Oracle client, by way of the RADIUS server and the Oracle database server, prompting the user for authentication information. The information could be, for example, a PIN as well as additional authentication information contained on the smart card.

The Oracle client sends the user's response to the authentication server by way of the Oracle database server and the RADIUS server. If the user has entered a valid number, the authentication server sends an *accept* packet back to the Oracle client by way of the RADIUS server and the Oracle database server. The user is now authenticated and authorized to access the appropriate tables and applications. If the user has entered incorrect information, the authentication server sends back a message rejecting user's access.

Example: Asynchronous Authentication with ActivCard Tokens

One particular ActivCard token is a hand-held device with a keypad and which displays a dynamic password. When the user seeks access to an Oracle database server by entering a password, the information is passed to the appropriate authentication server by way of the Oracle database server/RADIUS client and the RADIUS server. The authentication server sends back a challenge to the client, by way of the RADIUS server and the Oracle database server. The user types that challenge into the token, and the token displays a number for the user to send in response.

The Oracle client then sends the user's response to the authentication server by way of the Oracle database server and the RADIUS server. If the user has typed a valid number, the authentication server sends an *accept* packet back to the Oracle client by way of the RADIUS server and the Oracle database server. The user is now authenticated and authorized to access the appropriate tables and applications. If the user has entered an incorrect response, the authentication server sends back a message rejecting the user's access.

Enabling RADIUS Authentication, Authorization, and Accounting

To enable RADIUS authentication and accounting, perform the following tasks:

- Task 1: Install RADIUS on the Oracle Database Server and on the Oracle Client
- Task 2: Configure RADIUS Authentication
- Task 3: Create a User and Grant Access
- Task 4: Configure External RADIUS Authorization (optional)
- Task 5: Configure RADIUS Accounting
- Task 6: Add the RADIUS Client Name to the RADIUS Server Database
- Task 7: Configure the Authentication Server for Use with RADIUS.
- Task 8: Configure the RADIUS Server for Use with the Authentication Server
- Task 9: Configure Mapping Roles

Task 1: Install RADIUS on the Oracle Database Server and on the Oracle Client

RADIUS is installed with Oracle Advanced Security during a typical installation of Oracle Database.

> **See Also:**   Oracle Database operating system-specific installation documentation, for information about installing Oracle Advanced Security and the RADIUS adapter

Task 2: Configure RADIUS Authentication

This task includes the following steps:

- Step 1: Configure RADIUS on the Oracle Client
- Step 2: Configure RADIUS on the Oracle Database Server
- Step 3: Configure Additional RADIUS Features

Unless otherwise indicated, perform these configuration tasks by using Oracle Net Manager or by using any text editor to modify the `sqlnet.ora` file.

Step 1: Configure RADIUS on the Oracle Client

Use Oracle Net Manager to configure RADIUS on the Oracle client (See "Starting Oracle Net Manager" on page 2-2):

1. Navigate to the Oracle Advanced Security profile (For details, refer to "Navigating to the Oracle Advanced Security Profile" on page 2-2) The Oracle Advanced Security tabbed window is displayed (Figure 6–4):

Figure 6–4   Oracle Advanced Security Authentication Window
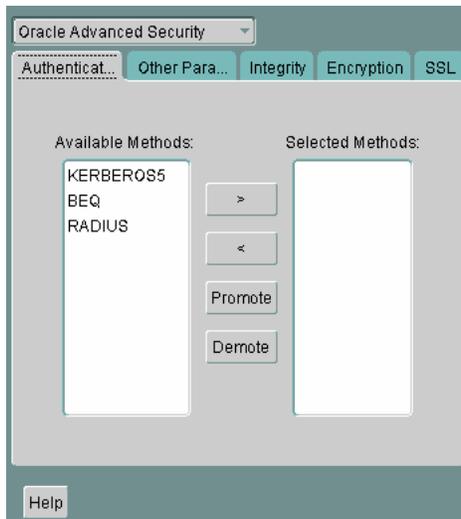
2. Click the **Authentication** tab.

3. From the Available Methods list, select **RADIUS**.

4. Select the right-arrow (**>**) to move RADIUS to the **Selected Methods** list. Move any other methods you want to use in the same way.

5. Arrange the selected methods in order of required usage by selecting a method in the Selected Methods list, and clicking **Promote** or **Demote** to position it in the list. For example, put RADIUS at the top of the list for it to be the first service used.

6. Select **File**, **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry:

   ```
   SQLNET.AUTHENTICATION_SERVICES=(RADIUS)
   ```

Step 2: Configure RADIUS on the Oracle Database Server

Following are the steps to configure RADIUS:

■   Create the RADIUS Secret Key File on the Oracle Database Server

■   Configure RADIUS Parameters on the Server (sqlnet.ora file)

■   Set Oracle Database Server Initialization Parameters

Create the RADIUS Secret Key File on the Oracle Database Server

Following are the steps to create Oracle Database Server:

1. Obtain the RADIUS secret key from the RADIUS server. For each RADIUS client, the administrator of the RADIUS server creates a shared secret key, which must be longer than 16-characters.

2. On the Oracle database server, create a directory:

   ■   (UNIX) *$ORACLE_HOME*`/network/security`

   ■   (Windows) *ORACLE_BASE\ORACLE_HOME*`\network\security`

3. Create the file `radius.key` to hold the shared secret copied from the RADIUS server. Place the file in the directory you created in Step 2.

4. Copy the shared secret key and paste it (and nothing else) into the `radius.key` file created on the Oracle database server.

5. For security purposes, change the file permission of `radius.key` to read only, accessible only by the Oracle owner. Oracle relies on the file system to keep this file secret.

> **See Also:** The RADIUS server administration documentation, for information about obtaining the secret key

Configure RADIUS Parameters on the Server (sqlnet.ora file)

Use Oracle Net Manager to configure RADIUS parameters on the server (Refer to "Starting Oracle Net Manager" on page 2-2):

1. Navigate to the Oracle Advanced Security profile. (Refer to "Navigating to the Oracle Advanced Security Profile" on page 2-2) The Oracle Advanced Security tabbed window is displayed. (Figure 6–4).

2. Click the **Authentication** tab.

3. From the Available Methods list, select **RADIUS**.

4. Move RADIUS to the **Selected Methods** list by choosing the right-arrow (**>**).

5. To arrange the selected methods in order of desired use, select a method in the Selected Methods list, and select **Promote** or **Demote** to position it in the list. For example, if you want RADIUS to be the first service used, put it at the top of the list.

6. Click **Other Params** as shown in (Figure 6–5):

Figure 6–5   Oracle Advanced Security Other Params Window
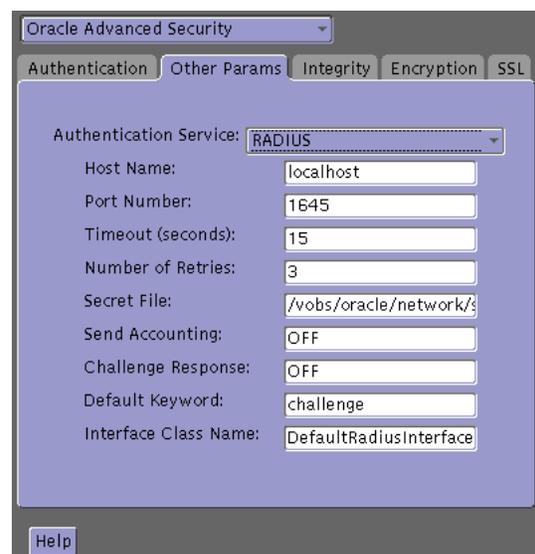
7. From the Authentication Service list, select **RADIUS**.

8. In the **Host Name** field, accept the **localhost** as the default primary RADIUS server, or enter another host name.

9. Ensure that the default value of the **Secret File** field is valid.

10. Select **File**, **Save Network Configuration**.

The `sqlnet.ora` file is updated with the following entries:

```
SQLNET.AUTHENTICATION_SERVICES=RADIUS
SQLNET.RADIUS_AUTHENTICATION=RADIUS_server_{hostname|IP_address}
```

Set Oracle Database Server Initialization Parameters

Configure the initialization parameter file, located in

- (UNIX) *$ORACLE_HOME*/admin/*db_name*/pfile

- (Windows) *ORACLE_BASE**ORACLE_HOME*\admin*db_name*\pfile

with the following values:

```
REMOTE_OS_AUTHENT=FALSE
OS_AUTHENT_PREFIX=""
```

> **Caution:** Setting `REMOTE_OS_AUTHENT` **to TRUE can enable a security breach because it lets someone using a non-secure protocol, such as TCP, perform an operating system-authorized login (formerly called an** `OPS$` **login).**

> **Note:** In addition to setting the `REMOTE_OS_AUTHENT` initialization parameter file to FALSE, you should issue the startup command with a `PFILE` option. This ensures that the parameters from your `initSID.ora` are used.

> **See Also:** *Oracle Database Reference* and the *Oracle Database Administrator's Guide* for information about setting initialization parameters on an Oracle Database server

Step 3: Configure Additional RADIUS Features

- Change Default Settings

- Configure Challenge-Response

- Set Parameters for an Alternate RADIUS Server

Change Default Settings

Use Oracle Net Manager to change default settings (See "Starting Oracle Net Manager" on page 2-2):

1. Navigate to the Oracle Advanced Security profile (See "Navigating to the Oracle Advanced Security Profile" on page 2-2) The Oracle Advanced Security tabbed window is displayed. (Figure 6–5).

2. Click the **Other Params** tab.

3. From the Authentication Service list, select **RADIUS**.

4. Change the default setting for any of the following fields:

Field	Description
Port Number	Specifies the listening port of the primary RADIUS server. The default value is 1645.

Field	Description
Timeout (seconds)	Specifies the time the Oracle database server waits for a response from the primary RADIUS server. The default is 15 seconds.
Number of Retries	Specifies the number of times the Oracle database server resends messages to the primary RADIUS server. The default is three retries.
	For instructions on configuring RADIUS accounting, see: Task 5: Configure RADIUS Accounting on page 6-14.
Secret File	Specifies the location of the secret key on the Oracle database server. The field specifies the location of the secret key file, not the secret key itself.
	For information about specifying the secret key, see: Create the RADIUS Secret Key File on the Oracle Database Server on page 6-8.

5. Select **File**, **Save Network Configuration**.

The `sqlnet.ora` file is updated with the following entries:

```
SQLNET.RADIUS_AUTHENTICATION_PORT=(PORT)
SQLNET.RADIUS_AUTHENTICATION_TIMEOUT=
(NUMBER OF SECONDS TO WAIT FOR response)
SQLNET.RADIUS_AUTHENTICATION_RETRIES=
(NUMBER OF TIMES TO RE-SEND TO RADIUS server)
SQLNET.RADIUS_SECRET=(path/radius.key)
```

Configure Challenge-Response

The challenge-response (asynchronous) mode presents the user with a graphical interface requesting first a password, then additional information, for example, a dynamic password that the user obtains from a token card. With the RADIUS adapter, this interface is Java-based to provide optimal platform independence.

> **Note:** Third party vendors of authentication devices must customize this graphical user interface to fit their particular device. For example, a smart card vendor would customize the Java interface so that the Oracle client reads data, such as a dynamic password, from the smart card. When the smart card receives a challenge, it responds by prompting the user for more information, such as a PIN.

> **See Also:** Appendix C, "Integrating Authentication Devices Using RADIUS", for information about how to customize the challenge-response user interface

To configure challenge-response:

1. If you are using JDK 1.1.7 or JRE 1.1.7, set the `JAVA_HOME` environment variable to the JRE or JDK location on the system where the Oracle client is run:

■ On UNIX, enter this command at the prompt:

```
% setenv JAVA_HOME /usr/local/packages/jre1.1.7B
```

■ On Windows, select **Start**, **Settings**, **Control Panel**, **System**, **Environment**, and set the JAVA_HOME variable as follows:

```
c:\java\jre1.1.7B
```

> **Note:** This step is not required for any other JDK/JRE version.

2. Navigate to the Oracle Advanced Security profile in Oracle Net Manager (See "Navigating to the Oracle Advanced Security Profile" on page 2-2) The Oracle Advanced Security Other Params window is displayed. (Figure 6–5).

3. From the Authentication Service list, select **RADIUS**.

4. In the **Challenge Response** field, enter **ON** to enable challenge-response.

5. In the **Default Keyword** field, accept the default value of the challenge or enter a keyword for requesting a challenge from the RADIUS server.

> **Note:** The keyword feature is provided by Oracle and supported by some, but not all, RADIUS servers. You can use this feature only if your RADIUS server supports it.
>
> By setting a keyword, you let the user avoid using a password to verify identity. If the user does *not* enter a password, the keyword you set here is passed to the RADIUS server which responds with a challenge requesting, for example, a driver's license number or birth date. If the user *does* enter a password, the RADIUS server may or may not respond with a challenge, depending upon the configuration of the RADIUS server.

6. In the **Interface Class Name** field, accept the default value of **DefaultRadiusInterface** or enter the name of the class you have created to handle the challenge-response conversation. If other than the default RADIUS interface is used, you also must edit the sqlnet.ora file to enter SQLNET.RADIUS_CLASSPATH=(location), where location is the complete path name of the jar file. It defaults to $ORACLE_HOME/network/jlib/netradius.jar: $ORACLE_HOME/JRE/lib/vt.jar

7. Select **File**, **Save Network Configuration**.

The sqlnet.ora file is updated with the following entries:

```
SQLNET.RADIUS_CHALLENGE_RESPONSE=([ON | OFF])
SQLNET.RADIUS_CHALLENGE_KEYWORD=(KEYWORD)
SQLNET.RADIUS_AUTHENTICATION_INTERFACE=(name of interface including the package
name delimited by "/" for ".")
```

Set Parameters for an Alternate RADIUS Server

If you are using an alternate RADIUS server, set these parameters in the sqlnet.ora file using any text editor.

```
SQLNET.RADIUS_ALTERNATE=(hostname or ip address of alternate radius server)
SQLNET.RADIUS_ALTERNATE_PORT=(1812)
SQLNET.RADIUS_ALTERNATE_TIMEOUT=(number of seconds to wait for response)
SQLNET.RADIUS_ALTERNATE_RETRIES=(number of times to re-send to radius server)
```

Task 3: Create a User and Grant Access

To grant user access:

1. Launch SQL*Plus and execute these commands to create and grant access to a user identified externally on the Oracle database server.

```
SQL> CONNECT system/manager@database_name;
SQL> CREATE USER username IDENTIFIED EXTERNALLY;
SQL> GRANT CREATE SESSION TO USER username;
SQL> EXIT
```

If you are using Windows, you can use the Security Manager tool in the Oracle Enterprise Manager.

> **See Also:**
>
> - *Oracle Database Administrator's Guide*
>
> - *Oracle Database Heterogeneous Connectivity Administrator's Guide*

2. Enter the same user in the RADIUS server's users file.

> **See Also:** Administration documentation for the RADIUS server

Task 4: Configure External RADIUS Authorization (optional)

If you require external RADIUS authorization for RADIUS users who connect to an Oracle database, then you must perform the following steps to configure the Oracle server, the Oracle client, and the RADIUS server:

To configure the Oracle server (RADIUS client):

Following steps describes how to configure Oracle Server (RADIUS Client):

1. Add the `OS_ROLE` parameter to the `init.ora` file and set this parameter to `TRUE` as follows:

```
OS_ROLE=TRUE
```

Then restart the database so the system can read the change to the `init.ora` file.

2. Set the RADIUS challenge-response mode to `ON` for the server if you have not already done so by following the steps listed in "Configure Challenge-Response" on page 6-11.

3. Add externally identified users and roles.

To configure the Oracle client (where users log in):

Set the RADIUS challenge-response mode to `ON` for the client if you have not already done so by following the steps listed in "Configure Challenge-Response" on page 6-11.

To configure the RADIUS server:

Following steps describe how to configure the RADIUS Server:

1. Add the following attributes to the RADIUS server attribute configuration file:

ATTRIBUTE NAME	CODE	TYPE
VENDOR_SPECIFIC	26	Integer
ORACLE_ROLE	1	String

2. Assign a Vendor ID for Oracle in the RADIUS server attribute configuration file that includes the SMI Network Management Private Enterprise Code of 111.

   For example, enter the following in the RADIUS server attribute configuration file:

   ```
   VALUE      VENDOR_SPECIFIC      ORACLE      111
   ```

3. Using the following syntax, add the ORACLE_ROLE attribute to the user profile of the users who will use external RADIUS authorization:

   ```
   ORA_databaseSID_rolename[_[A]|[D]]
   ```

   where:

   - ORA designates that this role is used for Oracle purposes

   - databaseSID is the Oracle system identifier that is configured in the database server's init.ora file

   - rolename is the name of role as it is defined in the data dictionary. For example, SYSDBA

   - A is an optional character that indicates the user has administrator's privileges for this role

   - D is an optional character that indicates this role is to be enabled by default

   Ensure that RADIUS groups which map to Oracle roles adhere to the ORACLE_ROLE syntax.

   For example:

   ```
   USERNAME      USERPASSWD="user_password",
                 SERVICE_TYPE=login_user,
                 VENDOR_SPECIFIC=ORACLE,
                 ORACLE_ROLE=ORA_ora920_sysdba
   ```

   > **See Also:**   The RADIUS server administration documentation for information about configuring the server.

Task 5: Configure RADIUS Accounting

RADIUS accounting logs information about access to the Oracle database server and stores it in a file on the RADIUS accounting server. Use this feature only if both the RADIUS server and authentication server support it.

Set RADIUS Accounting on the Oracle Database Server

Use Oracle Net Manager to enable or disable RADIUS accounting (See "Starting Oracle Net Manager" on page 2-2):

1. Navigate to the Oracle Advanced Security profile. (See "Navigating to the Oracle Advanced Security Profile" on page 2-2) The Other Params window is displayed (Figure 6–5).

2. From the Authentication Service list, select **RADIUS**.

3. In the Send Accounting field, enter **ON** to enable accounting or **OFF** to disable accounting.

4. Select **File**, **Save Network Configuration**.

   The sqlnet.ora file is updated with the following entry:

   ```
   SQLNET.RADIUS_SEND_ACCOUNTING= ON
   ```

Configure the RADIUS Accounting Server

RADIUS Accounting Server consists of an accounting server residing on either the same host as the RADIUS authentication server or on a separate host.

> **See Also:** Administration documentation for the RADIUS server, for information about configuring RADIUS accounting

Task 6: Add the RADIUS Client Name to the RADIUS Server Database

You can use virtually any RADIUS server that complies with the standards in the Internet Engineering Task Force (IETF) RFC #2138, *Remote Authentication Dial In User Service (RADIUS)* and RFC #2139 *RADIUS Accounting*. Because RADIUS servers vary, consult the documentation for your particular RADIUS server for any unique interoperability requirements.

Perform the following steps to add the RADIUS client name to a Livingston RADIUS server:

1. Open the clients file, which can be found at `/etc/raddb/clients`. The following text and table appear:

```
@ (#) clients 1.1 2/21/96 Copyright 1991 Livingston Enterprises Inc
This file contains a list of clients which are allowed to make authentication
requests and their encryption key. The first field is a valid hostname. The
second field (separated by blanks or tabs) is the encryption key.
Client Name                     Key
```

2. In the `CLIENT NAME` column, enter the host name or IP address of the host on which the Oracle database server is running. In the `KEY` column, type the shared secret.

   The value you enter in the `CLIENT NAME` column, whether it is the client's name or IP address, depends on the RADIUS server.

3. Save and close the clients file.

> **See Also:** Administration documentation for the RADIUS server

Task 7: Configure the Authentication Server for Use with RADIUS

Refer to the authentication server documentation for instructions about configuring the authentication servers.

> **See Also:** "Related Documentation" on page -xxii, which contains a list of possible resources.

Task 8: Configure the RADIUS Server for Use with the Authentication Server

Refer to the RADIUS server documentation.

Task 9: Configure Mapping Roles

If the RADIUS server supports vendor type attributes, you can manage roles by storing them in the RADIUS server. The Oracle database server downloads the roles when there is a `CONNECT` request using RADIUS.

To use this feature, configure roles on both the Oracle database server and the RADIUS server.

Perform these steps to configure roles on the Oracle database server:

1. Use a text editor to set the `OS_ROLES` parameter in the initialization parameters file on the Oracle database server.

2. Stop and restart the Oracle database server.

3. Create each role that the RADIUS server will manage on the Oracle database server with the value IDENTIFIED EXTERNALLY.

   To configure roles on the RADIUS server, refer to Table 6–2 and use the following syntax:

   ```
   ORA_DatabaseName.DatabaseDomainName_RoleName
   ```

   Example:

   ```
   ORA_USERDB.US.ORACLE.COM_MANAGER
   ```

Table 6–2   RADIUS Configuration Parameters

Parameter	Description
DatabaseName	The name of the Oracle database server for which the role is being created. This is the same as the value of the DB_NAME initialization parameter.
DatabaseDomainName	The name of the domain to which the Oracle database server belongs. The value is the same as the value of the DB_DOMAIN initialization parameter.
RoleName	The name of the role created in the Oracle database server.

4. Configure RADIUS challenge-response mode.

   See Also:

   - Challenge-Response (Asynchronous) Authentication Mode on page 6-4

   - Configure Challenge-Response on page 6-11

Using RADIUS to Log In to a Database

If you are using the synchronous authentication mode, launch SQL*Plus and enter the following command at the prompt:

```
CONNECT username/password@database_alias
```

> **Note:** You can log in with this command only when challenge-response is not turned to `ON`.

If you are using the challenge-response mode, launch SQL*Plus and, at the prompt, enter the command that follows:

```
CONNECT /@database_alias
```

> **Note:** you can log in with this command only when challenge-response is turned to `ON`.

> **Note:** The challenge-response mode can be configured for all login cases.

RSA ACE/Server Configuration Checklist

If you are using an RSA ACE/Server as a RADIUS server, check the following items before making your initial connection:

- Ensure that the host agent in the RSA ACE/Server is set up to send a node secret. In version 5.0, this is done by leaving the SENT Node secret box unchecked. If the RSA ACE/Server fails to send a node secret to the agent, then a node verification failure message will be written to the RSA ACE/Server log.

- If you are using RSA SecurID tokens, then ensure that the token is synchronized with the RSA ACE/Server.

    See Also: RSA ACE/Server documentation for specific information about troubleshooting.

7

Configuring Kerberos Authentication

This chapter describes how to configure Oracle Advanced Security for Oracle Database for use with Kerberos authentication, and how to configure Kerberos to authenticate Oracle database users. This chapter contains the following sections:

- Enabling Kerberos Authentication
- Utilities for the Kerberos Authentication Adapter
- Configuring Interoperability with a Windows 2000 Domain Controller KDC
- Troubleshooting

> **See Also:** *Oracle Database Enterprise User Administrator's Guide* for information on migrating Kerberos users to Kerberos-authenticated enterprise users

Enabling Kerberos Authentication

To enable Kerberos authentication:

- Task 1: Install Kerberos
- Task 2: Configure a Service Principal for an Oracle Database Server
- Task 3: Extract a Service Table from Kerberos
- Task 4: Install an Oracle Database Server and an Oracle Client
- Task 5: Install Oracle Net Services and Oracle Advanced Security
- Task 6: Configure Oracle Net Services and Oracle Database
- Task 7: Configure Kerberos Authentication
- Task 8: Create a Kerberos User
- Task 9: Create an Externally Authenticated Oracle User
- Task 10: Get an Initial Ticket for the Kerberos/Oracle User

Task 1: Install Kerberos

Install Kerberos on the system that functions as the authentication server.

> **See Also:** About building and installing Kerberos from Kerberos version 5 source.

Distribution for information about how to install Kerberos.

> **Note:** After upgrading from a 32-bit version of Oracle Database, the first use of the Kerberos authentication adapter causes an error message: `ORA-01637: Packet receive failed`.
>
> Workaround: After upgrading to the 64-bit version of the database and before using Kerberos external authentication method, check for a file named `/usr/tmp/oracle_service_name.RC` on your computer, and remove it.

Task 2: Configure a Service Principal for an Oracle Database Server

To enable the Oracle database server to validate the identity of clients that authenticate themselves using Kerberos, you must create a **service principal** for Oracle Database.

The name of the principal should have the following format:

```
kservice/kinstance@REALM
```

Each of the fields in the service principal specify the following values:

Service Principal Field	Description
kservice	A case-sensitive string that represents the Oracle service. This can be the same as the database service name.
kinstance	This is typically the fully qualified name of the system on which Oracle Database is running.
REALM	The domain name of the database server. REALM must always be uppercase and is typically the DNS domain name.

> **Note:** The utility names in this section are executable programs. However, the Kerberos user name `krbuser` and the realm `SOMECO.COM` are examples only.

For example, suppose kservice is `oracle`, the fully qualified name of the system on which Oracle Database is running is `dbserver.someco.com` and the realm is `SOMECO.COM`. The principal name then is:

```
oracle/dbserver.someco.com@SOMECO.COM
```

It is a convention to use the DNS domain name as the name of the realm. To create the **service principal**, run `kadmin.local`. On UNIX, run this command as the root user, by using the following syntax:

```
# cd /kerberos-install-directory/sbin
# ./kadmin.local
```

To add a **principal** named `oracle/dbserver.someco.com@SOMECO.COM` to the list of server principals known by Kerberos, enter the following:

```
kadmin.local:addprinc -randkey oracle/dbserver.someco.com@SOMECO.COM
```

Task 3: Extract a Service Table from Kerberos

Extract the **service table** from Kerberos and copy it to the Oracle database server/Kerberos client system.

For example, use the following steps to extract a service table for `dbserver.someco.com`:

1. Enter the following to extract the service table:

   ```
   kadmin.local:  ktadd -k /tmp/keytab oracle/dbserver.someco.com

   Entry for principal oracle/dbserver.someco.com with kvno 2, encryption
   DES-CBC-CRC added to the keytab WRFILE: 'WRFILE:/tmp/keytab

   kadmin.local:  exit

   oklist -k -t /tmp/keytab
   ```

2. After the service table has been extracted, verify that the new entries are in the table in addition to the old ones. If they are not, or you need to add more, use `kadmin.local` to append to them.

   If you do not enter a realm when using `ktadd`, it uses the realm of the current host and displays it in the command output, as shown in Step 1.

3. If the Kerberos service table is on the same system as the Kerberos client, you can move it. If the service table is on a different system from the Kerberos client, you must transfer the file with a program such as FTP. If using FTP, transfer the file in binary mode.

   The following example shows how to move the service table on a UNIX platform:

   ```
   # mv /tmp/keytab /etc/v5srvtab
   ```

   The default name of the service file is `/etc/v5srvtab`.

4. Verify that the owner of the Oracle database server executable can read the service table (`/etc/v5srvtab` in the previous example). To do so, set the file owner to the Oracle user, or make the file readable by the group to which Oracle belongs.

   > **Caution:** Do not make the file readable to all users. This can cause a security breach.

Task 4: Install an Oracle Database Server and an Oracle Client

Install the Oracle database server and client software.

> **See Also:** Oracle Database operating system-specific installation documentation

Task 5: Install Oracle Net Services and Oracle Advanced Security

Install Oracle Net Services and Oracle Advanced Security on the Oracle database server and Oracle client systems.

> **See Also:** Oracle Database operating system-specific installation documentation

Task 6: Configure Oracle Net Services and Oracle Database

Configure Oracle Net Services on the Oracle database server and client.

See Also:

- Oracle Database operating system-specific installation documentation

- *Oracle Database Net Services Administrator's Guide*.

Task 7: Configure Kerberos Authentication

Perform these tasks to set required parameters in the Oracle database server and client `sqlnet.ora` files:

- Step 1: Configure Kerberos on the Client and on the Database Server

- Step 2: Set the Initialization Parameters

- Step 3: Set sqlnet.ora Parameters (optional)

Step 1: Configure Kerberos on the Client and on the Database Server

Use Oracle Net Manager to perform the following steps to configure Kerberos authentication service parameters on the client and on the database server (Refer to, "Starting Oracle Net Manager" on page 2-2):

1. Navigate to the Oracle Advanced Security profile. (Refer to "Navigating to the Oracle Advanced Security Profile" on page 2-2) The Oracle Advanced Security window is displayed (Figure 7–1):

Figure 7–1   Oracle Advanced Security Authentication Window (Kerberos)
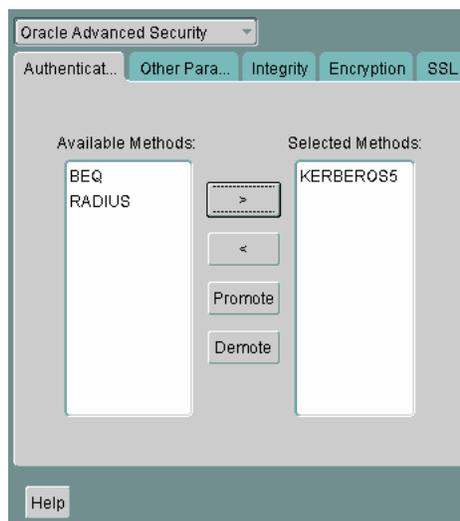
2. Click the **Authentication** tab.

3. From the Available Methods list, select **KERBEROS5**.

4. Move KERBEROS5 to the Selected Methods list by clicking the right arrow (**>**).

5. Arrange the selected methods in order of use. To do this, select a method in the Selected Methods list, then click **Promote** or **Demote** to position it in the list. For example, if you want KERBEROS5 to be the first service used, move it to the top of the list.

6. Click the **Other Params** tab (Figure 7–2).

Figure 7–2   Oracle Advanced Security Other Params Window (Kerberos)
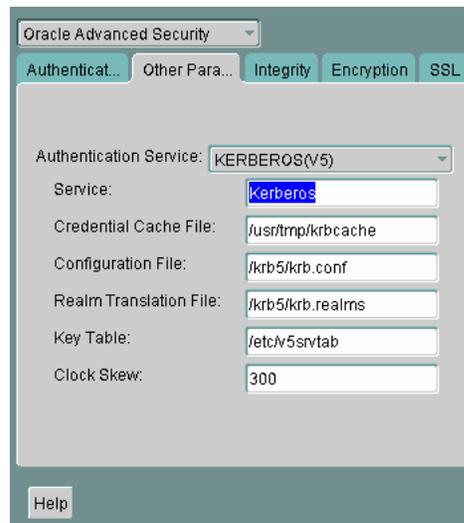
1. From the Authentication Service list, select **KERBEROS(V5)**.

2. Type **Kerberos** into the **Service** field. This field defines the name of the service Oracle Database uses to obtain a Kerberos **service ticket**. When you provide the value for this field, the other fields are enabled.

3. Optionally enter values for the following fields:

    ■   Credential Cache File

    ■   Configuration File

    ■   Realm Translation File

    ■   Key Table

    ■   Clock Skew

        See Also: Oracle Net Manager online Help, and "Step 3: Set sqlnet.ora Parameters (optional)" on page 7-6, for more information about the fields and the parameters they configure

4. Select **File**, **Save Network Configuration**.

    The sqlnet.ora file is updated with the following entries:

    ```
    SQLNET.AUTHENTICATION_SERVICES=(KERBEROS5)
    SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=kservice
    ```

Step 2: Set the Initialization Parameters

To set parameters in the initialization parameter file:

1. Add the following parameter to the initialization parameter file:

    ```
    REMOTE_OS_AUTHENT=FALSE
    ```

> **Caution:** Setting REMOTE_OS_AUTHENT to TRUE can enable a security breach, because it lets someone using a non-secure protocol, such as TCP, perform an operating system-authorized login (formerly called an OPS$ login).

> **Note:** In addition to setting the REMOTE_OS_AUTHENT initialization parameter file to FALSE, you should issue the startup command with a PFILE option. This ensures that the parameters from initSID.ora file are used.

2. Because Kerberos user names can be long, and Oracle user names are limited to 30 characters, Oracle strongly recommends that you set the value of OS_AUTHENT_ PREFIX to null as follows:

```
OS_AUTHENT_PREFIX=""
```

Setting this parameter to null overrides the default value of OPS$.

Step 3: Set sqlnet.ora Parameters (optional)

In addition to the required parameters, you can optionally set the following parameters in the sqlnet.ora file on the client and the Oracle database server:

Attribute	Description
Parameter:	SQLNET.KERBEROS5_CC_NAME=*pathname_to_credentials_ cache_file*
Description:	Specifies the complete path name to the Kerberos credentials cache (CC) file. The default value is operating system-dependent. For UNIX, it is /tmp/krb5cc_userid.
	You can also set this parameter by using the KRB5CCNAME environment variable, but the value set in the sqlnet.ora file takes precedence over the value set in KRB5CCNAME.
Example:	SQLNET.KERBEROS5_CC_NAME=/usr/tmp/krbcache
Parameter:	SQLNET.KERBEROS5_CLOCKSKEW=*number_of_seconds_ accepted_as_network_delay*
Description:	This parameter specifies how many seconds can pass before a Kerberos credential is considered out-of-date. It is used when a credential is actually received by either a client or a database server. An Oracle database server also uses it to decide if a credential needs to be stored to protect against a replay attack. The default is 300 seconds.
Example:	SQLNET.KERBEROS5_CLOCKSKEW=1200
Parameter:	SQLNET.KERBEROS5_CONF=*pathname_to_Kerberos_ configuration_file*
Description:	This parameter specifies the complete path name to the Kerberos configuration file. The configuration file contains the realm for the default KDC (key distribution center) and maps realms to KDC hosts. The default is operating system-dependent. For UNIX, it is /krb5/krb.conf.
Example:	SQLNET.KERBEROS5_CONF=/krb/krb.conf
Parameter:	SQLNET.KERBEROS5_CONF_MIT=[TRUE|FALSE]

Attribute	Description
Description:	This parameter specifies whether the new MIT Kerberos configuration format is used. If the value is set to TRUE, it will parse the file according to the new configuration format rules. When the value is set to FALSE, the default (non-MIT) configuration is used. The default is FALSE.
Example:	SQLNET.KERBEROS5_CONF_MIT=False
Parameter:	SQLNET.KERBEROS5_KEYTAB= *pathname_to_Kerberos_principal/key_table*
Description:	This parameter specifies the complete path name to the Kerberos principal/secret key mapping file. It is used by the Oracle database server to extract its key and decrypt the incoming authentication information from the client. The default is operating system-dependent. For UNIX, it is /etc/v5srvtab.
Example:	SQLNET.KERBEROS5_KEYTAB=/etc/v5srvtab
Parameter:	SQLNET.KERBEROS5_REALMS= *pathname_to_Kerberos_realm_translation_file*
Description:	This parameter specifies the complete path name to the Kerberos realm translation file. The translation file provides a mapping from a host name or domain name to a realm. The default is operating system-dependent. For UNIX, it is /etc/krb.realms.
Example:	SQLNET.KERBEROS5_REALMS=/krb5/krb.realms

Task 8: Create a Kerberos User

To create Oracle users that Kerberos can authenticate, perform this task on the Kerberos authentication server where the administration tools are installed. The realm must already exist.

> **Note:** The utility names in this section are executable programs. However, the Kerberos user name krbuser and realm SOMECO.COM are examples only. They can vary among systems.

Run /krb5/admin/kadmin.local as root to create a new Kerberos user, such as krbuser.

The following example is UNIX-specific:

```
# ./kadmin.local
kadmin.local: addprinc krbuser
Enter password for principal: "krbuser@SOMECO.COM": (password does not display)
Re-enter password for principal: "krbuser@SOMECO.COM": (password does not display)
kadmin.local: exit
```

Task 9: Create an Externally Authenticated Oracle User

Run SQL*Plus on the Oracle database server to create the Oracle user that corresponds to the Kerberos user. In the following example, OS_AUTHENT_PREFIX is set to null (""). The Oracle user name is in uppercase enclosed in double quotation marks as shown in the following example:

```
SQL> CONNECT / AS SYSDBA;
SQL> CREATE USER "KRBUSER@SOMECO.COM" IDENTIFIED EXTERNALLY;
SQL> GRANT CREATE SESSION TO "KRBUSER@SOMECO.COM";
```

Task 10: Get an Initial Ticket for the Kerberos/Oracle User

Before you can connect to the database, you must ask the Key Distribution Center (KDC) for an **initial ticket**. To do so, run the following on the client:

```
% okinit username
```

If, when making a database connection, a reference such as the following follows a database link, you must use the forwardable flag (-f) option:

```
sqlplus /@oracle
```

Executing `okinit -f` enables credentials that can be used across database links. Run the following commands on the Oracle client:

```
% okinit -f
Password for krbuser@SOMECO.COM:password
```

Utilities for the Kerberos Authentication Adapter

Three utilities are shipped with the Oracle Kerberos authentication adapter. These utilities are intended for use on an Oracle client with Oracle Kerberos authentication support installed. Use the following utilities for these specified tasks:

- Obtaining the Initial Ticket with the okinit Utility
- Displaying Credentials with the oklist Utility
- Removing Credentials from the Cache File with the okdstry Utility

Obtaining the Initial Ticket with the okinit Utility

The `okinit` utility obtains and caches Kerberos tickets. This utility is typically used to obtain the ticket-granting ticket, using a password entered by the user to decrypt the credential from the key distribution center (KDC). The ticket-granting ticket is then stored in the user's credential cache.

The options available with `okinit` are listed in Table 7–1:

Table 7–1   Options for the okinit Utility

Option	Description
-f	Ask for a forwardable ticket-granting ticket. This option is necessary to follow database links.
-l	Specify the lifetime of the ticket-granting ticket and all subsequent tickets. By default, the ticket-granting ticket is good for eight (8) hours, but shorter or longer-lived credentials may be desired. Note that the KDC can ignore this option or put site-configured limits on what can be specified. The lifetime value is a string that consists of a number qualified by w (weeks), d (days), h (hours), m (minutes), or s (seconds), as in the following example: `okinit -l 2w1d6h20m30s` The example requests a ticket-granting ticket that has a life time of 2 weeks, 1 day, 6 hours, 20 minutes, and 30 seconds.
-c	Specify an alternative credential cache. For UNIX, the default is `/tmp/krb5cc_`*uid*. You can also specify the alternate credential cache by using the `SQLNET.KERBEROS5_CC_NAME` parameter in the `sqlnet.ora` file.
-?	List command line options.

Displaying Credentials with the oklist Utility

Run the `oklist` utility to display the list of tickets held. Available `oklist` options are listed in Table 7–2:

Table 7–2   Options for the oklist Utility

Option	Description
`-f`	Show flags with credentials. Relevant flags are: ■ `I`, credential is a ticket-granting ticket ■ `F`, credential is forwardable ■ `f`, credential is forwarded.
`-c`	Specify an alternative credential cache. In UNIX, the default is `/tmp/krb5cc_uid`. The alternate credential cache can also be specified by using the `SQLNET.KERBEROS5_CC_NAME` parameter in the `sqlnet.ora` file.
`-k`	List the entries in the service table (default `/etc/v5srvtab`) on UNIX. The alternate service table can also be specified by using the `SQLNET.KERBEROS5_KEYTAB` parameter in the `sqlnet.ora` file.

The show flag option (`-f`) displays additional information, as shown in the following example:

```
% oklist -f
27-Jul-1999 21:57:51   28-Jul-1999 05:58:14
krbtgt/SOMECO.COM@SOMECO.COM
Flags: FI
```

Removing Credentials from the Cache File with the okdstry Utility

Use the `okdstry` utility to remove credentials from the credentials cache file:

```
$ okdstry -f
```

where the `-f` command option lets you specify an alternative credential cache. For UNIX, the default is `/tmp/krb5cc_uid`. You can also specify the alternate credential cache by using the `SQLNET.KRB5_CC_NAME` parameter in the `sqlnet.ora` file.

Connecting to an Oracle Database Server Authenticated by Kerberos

You can now connect to an Oracle database server without using a user name or password. Enter a command similar to the following:

```
$ sqlplus /@net_service_name
```

where *net_service_name* is an Oracle Net Services service name. For example:

```
$ sqlplus /@oracle_dbname
```

> **See Also:**   Chapter 1, "Introduction to Oracle Advanced Security" and *Oracle Database Heterogeneous Connectivity Administrator's Guide* for information about external authentication

Configuring Interoperability with a Windows 2000 Domain Controller KDC

Oracle Advanced Security, which complies with MIT Kerberos, can interoperate with tickets that are issued by a Kerberos Key Distribution Center (KDC) on a Windows

2000 domain controller to enable Kerberos authentication with an Oracle database. To configure Kerberos authentication that uses a Windows 2000 domain controller KDC, perform the following tasks:

- Task 1: Configure an Oracle Kerberos Client to Interoperate with a Windows 2000 Domain Controller KDC

- Task 2: Configure a Windows 2000 Domain Controller KDC to Interoperate with an Oracle Client

- Task 3: Configure an Oracle Database to Interoperate with a Windows 2000 Domain Controller KDC

- Task 4: Obtain an Initial Ticket for the Kerberos/Oracle User

Task 1: Configure an Oracle Kerberos Client to Interoperate with a Windows 2000 Domain Controller KDC

The following steps must be performed on the Oracle Kerberos client.

Step 1: Create the Client Kerberos Configuration Files to Use a Windows Domain Controller KDC

Create the following Kerberos client configuration files that refer to the Windows 2000 domain controller as the Kerberos KDC. In the examples that follow, the Windows 2000 domain controller is running on a node named `sales3854.us.acme.com`.

- **`krb.conf`** file

  For example:

  ```
  SALES3854.US.ACME.COM
  SALES3854.US.ACME.COM sales3854.us.acme.com admin server
  ```

- **`krb5.conf`** file

  For example:

  ```
  [libdefaults]
  default_realm=SALES.US.ACME.COM
  [realms]
  SALES.US.ACME.COM= {
  kdc=sales3854.us.acme.com:88
  }
  [domain_realm]
  .us.acme.com=SALES.US.ACME.COM
  ```

- **`krb5.realms`** file

  For example:

  ```
  us.acme.com SALES.US.ACME.COM
  ```

Step 2: Specify the Oracle Configuration Parameters in the sqlnet.ora File

Configuring an Oracle client to interoperate with a Windows 2000 domain controller KDC uses the same `sqlnet.ora` file parameters that are listed in "Step 1: Configure Kerberos on the Client and on the Database Server" on page 7-4.

Set the following parameters in the `sqlnet.ora` file on the client:

```
SQLNET.KERBEROS5_CONF=pathname_to_Kerberos_configuration_file
SQLNET.KERBEROS5_CONF_MIT=TRUE
```

```
SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=Kerberos_service_name
SQLNET.AUTHENTICATION_SERVICES=(BEQ,KERBEROS5)
```

> **Note:** Ensure that the `SQLNET.KERBEROS5_CONF_MIT`
> parameter is set to `TRUE` because the Windows 2000 operating
> system is designed to interoperate only with security services that
> are based on MIT Kerberos version 5.

Step 3: Specify the Listening Port Number

The Windows 2000 domain controller KDC listens on UDP/TCP port 88. Ensure that the system file entry for `kerberos5` is set to UDP/TCP port 88 as follows:

For the UNIX environ,emt, ensure that the `kerberos5` entry in the `/etc/services` file is set to 88

Task 2: Configure a Windows 2000 Domain Controller KDC to Interoperate with an Oracle Client

The following steps must be performed on the Windows 2000 domain controller.

> **See Also:** Microsoft documentation for information about how to
> create users in Active Directory.

Step 1: Create the User

Create a new user for the Oracle client in Microsoft Active Directory.

Step 2: Create the Oracle Database Principal

1. Create a new user for the Oracle database in Microsoft Active Directory.

   For example, if the Oracle database runs on the host `sales3854.us.acme.com`, then use Active Directory to create a user with the user name `sales3854.us.acme.com` and the password `oracle`.

   > **Note:** Do not create a user as `host/hostname.dns.com`, such as
   > `oracle/sales3854.us.acme.com`, in Active Directory.
   > Microsoft's KDC does not support multipart names like an MIT
   > KDC does. An MIT KDC allows multipart names to be used for
   > service principals because it treats all principals as user names.
   > However, Microsoft's KDC does not.

2. Use the `Ktpass` command line utility to extract the keytab file with the following syntax:

   ```
   Ktpass -princ service/hostname@NT-DNS-REALM-NAME -mapuser account -pass
   password -out keytab.file
   ```

   Using the database user created in the previous step, the following is an example of `Ktpass` usage:

   ```
   C:> Ktpass -princ oracle/sales3854.us.acme.com@SALES.US.COM -mapuser sales3854
   -pass oracle -out C:\temp\v5srvtab
   ```

This utility is part of the Windows 2000 Support Tools and can be found on the Windows 2000 distribution media in the `\support\reskit\netmgmt\security` folder.

3. Copy the extracted keytab file to the host computer where the Oracle database is installed.

For example, the keytab that was created in the previous step can be copied to `/krb5/v5svrtab`.

> **See Also:** Detailed information about Windows 2000 interoperability with Kerberos 5 that is available at the following URL:
>
> http://www.microsoft.com/WINDOWS2000/techinfo/planning/security/kerbsteps.asp

Task 3: Configure an Oracle Database to Interoperate with a Windows 2000 Domain Controller KDC

The following steps must be performed on the host computer where the Oracle database is installed.

Step 1: Set Configuration Parameters in the sqlnet.ora File

Specify values for the following parameters in the `sqlnet.ora` file for the database server:

```
SQLNET.KERBEROS5_CONF=pathname_to_Kerberos_configuration_file
SQLNET.KERBEROS5_KEYTAB=pathname_to_Kerberos_principal/key_table
SQLNET.KERBEROS5_CONF_MIT=TRUE
SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=Kerberos_service_name
SQLNET.AUTHENTICATION_SERVICES=(BEQ,KERBEROS5)
```

> **Note:** Ensure that the `SQLNET.KERBEROS5_CONF_MIT` parameter is set to `TRUE` because the Windows 2000 operating system is designed to interoperate only with security services that are based on MIT Kerberos version 5.

Step 2: Create an Externally Authenticated Oracle User

Follow the task information for "Task 9: Create an Externally Authenticated Oracle User" on page 7-7 to create an externally authenticated Oracle user. Ensure that the username is created in all uppercase characters. For example, `ORAKRB@SALES.US.ACME.COM`.

> **See Also:** "Task 7: Configure Kerberos Authentication" on page 7-4 for information about using Oracle Net Manager to set the `sqlnet.ora` file parameters.

Task 4: Obtain an Initial Ticket for the Kerberos/Oracle User

Before a client can connect to the database, the client must request an **initial ticket**. To request an initial ticket, follow the task information for "Task 10: Get an Initial Ticket for the Kerberos/Oracle User" on page 7-8.

Troubleshooting

This section lists some common configuration problems and explains how to resolve them.

- If you cannot get your ticket-granting ticket using `okinit`:
    - Ensure that the default realm is correct by examining the `krb.conf` file.
    - Ensure that the KDC is running on the host specified for the realm.
    - Ensure that the KDC has an entry for the user principal and that the passwords match.
    - Ensure that the `krb.conf` and `krb.realms` files are readable by Oracle.
- If you have an initial ticket but still cannot connect:
    - After trying to connect, check for a service ticket.
    - Check that the `sqlnet.ora` file on the database server side has a service name that corresponds to a service known by Kerberos.
    - Check that the clocks on all systems involved are set to times that are within a few minutes of each other or change the `SQLNET.KERBEROS5_CLOCKSKEW` parameter in the `sqlnet.ora` file.
- If you have a service ticket and you still cannot connect:
    - Check the clocks on the client and database server.
    - Check that the `v5srvtab` file exists in the correct location and is readable by Oracle. Remember to set the `sqlnet.ora` parameters.
    - Check that the `v5srvtab` file has been generated for the service named in the `sqlnet.ora` file on the database server side.
- If everything seems to work fine, but then you issue another query and it fails:
    - Check that the initial ticket is forwardable. You must have obtained the initial ticket by running the `okinit` utility.
    - Check the expiration date on the credentials. If the credentials have expired, then close the connection and run `okinit` to get a new initial ticket.

8

Configuring Secure Sockets Layer Authentication

This chapter describes how to configure and use the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols which are supported by Oracle Advanced Security. It contains the following topics:

- SSL and TLS in an Oracle Environment
- Public Key Infrastructure in an Oracle Environment
- SSL Combined with Other Authentication Methods
- SSL and Firewalls
- SSL Usage Issues
- Enabling SSL
- Troubleshooting SSL
- Certificate Validation with Certificate Revocation Lists
- Configuring Your System to Use Hardware Security Modules

SSL and TLS in an Oracle Environment

Secure Sockets Layer (SSL) is an industry standard protocol originally designed by Netscape Communications Corporation for securing network connections. SSL uses RSA public key cryptography in conjunction with symmetric key cryptography to provide authentication, encryption, and data integrity.

This section discusses the following topics:

- Difference between SSL and TLS
- Using SSL
- How SSL Works in an Oracle Environment: The SSL Handshake

Difference between SSL and TLS

Although SSL was primarily developed by Netscape Communications Corporation, the Internet Engineering Task Force (IETF) took over development of it, and renamed it Transport Layer Security (TLS). Essentially, TLS is an incremental improvement to SSL version 3.0.

> **See Also:** *The TLS Protocol Version 1.0* [RFC 2246] at the IETF Web site, which can be found at:
>
> http://www.ietf.org

> **Note:** To simplify discussion, this chapter uses the term SSL where either SSL or TLS may be appropriate because SSL is the most widely recognized term. However, where distinctions occur between how you use or configure these protocols, this chapter specifies what is appropriate for either SSL or TLS.

Using SSL

Oracle Advanced Security supports authentication by using digital certificates over SSL in addition to the native encryption and data integrity capabilities of these protocols.

By using Oracle Advanced Security SSL functionality to secure communications between clients and servers, you can

- Use SSL to encrypt the connection between clients and servers

- Authenticate any client or server, such as Oracle Application Server 10g, to any Oracle database server that is configured to communicate over SSL

You can use SSL features by themselves or in combination with other authentication methods supported by Oracle Advanced Security. For example, you can use the encryption provided by SSL in combination with the authentication provided by Kerberos. SSL supports any of the following authentication modes:

- Only the server authenticates itself to the client

- Both client and server authenticate themselves to each other

- Neither the client nor the server authenticates itself to the other, thus using the SSL encryption feature by itself

> **See Also:**
> - *The SSL Protocol*, version 3.0, published by the Internet Engineering Task Force, for a more detailed discussion of SSL
> - Chapter 1, "Introduction to Oracle Advanced Security", for more information about authentication methods

How SSL Works in an Oracle Environment: The SSL Handshake

When a network connection over SSL is initiated, the client and server perform an SSL handshake that includes the following steps:

- The client and server establish which **cipher suite**s to use. This includes which encryption algorithms are used for data transfers.

- The server sends its certificate to the client, and the client verifies that the server's certificate was signed by a trusted CA. This step verifies the identity of the server.

- Similarly, if client authentication is required, the client sends its own certificate to the server, and the server verifies that the client's certificate was signed by a trusted CA.

- The client and server exchange key information using public key cryptography. Based on this information, each generates a **session key**. All subsequent communications between the client and the server is encrypted and decrypted by using this set of session keys and the negotiated cipher suite.

The authentication process consists of the following steps:

1. On a client, the user initiates an Oracle Net connection to the server by using SSL.

2. SSL performs the handshake between the client and the server.

3. If the handshake is successful, the server verifies that the user has the appropriate **authorization** to access the database.

Public Key Infrastructure in an Oracle Environment

A public key infrastructure (PKI) is a substrate of network components that provide a security underpinning, based on trust assertions, for an entire organization. A PKI exists so that disparate network entities can access its security services, which use public-key cryptography on an as-needed basis. Oracle provides a complete PKI that is based on RSA Security, Inc., Public-Key Cryptography Standards, and which interoperates with Oracle servers and clients.

About Public Key Cryptography

Traditional private-key or symmetric-key cryptography requires a single, secret key that is shared by two or more parties to a secure communication. This key is used to both encrypt and decrypt secure messages sent between the parties, requiring prior, secure distribution of the key to each party. The problem with this method is that it is difficult to securely transmit and store the key.

Public-key cryptography provides a solution to this problem, by employing **public and private key pair**s and a secure method for key distribution. The freely available **public key** is used to encrypt messages that can *only* be decrypted by the holder of the associated **private key**. The private key is securely stored, together with other security credentials, in an encrypted container called a **wallet**.

Public-key algorithms can guarantee the secrecy of a message, but they do not necessarily guarantee secure communications because they do not verify the identities of the communicating parties. To establish secure communications, it is important to verify that the public key used to encrypt a message does in fact belong to the target recipient. Otherwise, a third party can potentially eavesdrop on the communication and intercept public key requests, substituting its own public key for a legitimate key (the **man-in-the-middle** attack).

In order to avoid such an attack, it is necessary to verify the owner of the public key, a process called **authentication**. Authentication can be accomplished through a **certificate authority** (CA), which is a third party that is trusted by both of the communicating parties.

The CA issues public key certificates that contain an entity's name, public key, and certain other security credentials. Such credentials typically include the CA name, the CA signature, and the certificate effective dates (From Date, To Date).

The CA uses its private key to encrypt a message, while the public key is used to decrypt it, thus verifying that the message was encrypted by the CA. The CA public key is well known and does not have to be authenticated each time it is accessed. Such CA public keys are stored in wallets.

Public Key Infrastructure Components in an Oracle Environment

Public key infrastructure (PKI) components in an Oracle environment include the following:

- Certificate Authority
- Certificates
- Certificate Revocation Lists
- Wallets
- Hardware Security Modules

Certificate Authority

A certificate authority (CA) is a trusted third party that certifies the identity of entities, such as users, databases, administrators, clients, and servers. When an entity requests certification, the CA verifies its identity and grants a certificate, which is signed with the CA's private key.

Different CAs may have different identification requirements when issuing certificates. Some CAs may verify a requester's identity with a driver's license, some may verify identity with the requester's fingerprints, while others may require that requesters have their certificate request form notarized.

The CA publishes its own certificate, which includes its public key. Each network entity has a list of trusted CA certificates. Before communicating, network entities exchange certificates and check that each other's certificate is signed by one of the CAs on their respective trusted CA certificate lists.

Network entities can obtain their certificates from the same or different CAs. By default, Oracle Advanced Security automatically installs trusted certificates from VeriSign, RSA, Entrust, and GTE CyberTrust when you create a new wallet.

Oracle Application Server Certificate Authority, part of Oracle Identity Management Infrastructure, is a new Oracle PKI component available in Oracle Application Server 10g (9.0.4).

> **See Also:** "Wallets" on page 8-5

Certificates

A certificate is created when an entity's public key is signed by a trusted certificate authority (CA). A certificate ensures that an entity's identification information is correct and that the public key actually belongs to that entity.

A certificate contains the entity's name, public key, and an expiration date, as well as a serial number and **certificate chain** information. It can also contain information about the privileges associated with the certificate.

When a network entity receives a certificate, it verifies that it is a **trusted certificate**, that is, one that has been issued and signed by a **trusted certificate authority**. A certificate remains valid until it expires or until it is revoked.

Certificate Revocation Lists

Typically, when a CA signs a certificate binding a public key pair to a user identity, the certificate is valid for a specified period of time. However, certain events, such as user name changes or compromised private keys, can render a certificate invalid before the validity period expires. When this happens, the CA revokes the certificate and adds its serial number to a Certificate Revocation List (CRL). CAs periodically

publish CRLs to alert the user population when it is no longer acceptable to use a particular public key to verify its associated user identity.

When servers or clients receive user certificates in an Oracle environment, they can validate the certificate by checking its expiration date, signature, and revocation status. Certificate revocation status is checked by validating it against published CRLs. If certificate revocation status checking is turned on, then the server searches for the appropriate CRL depending on how this feature has been configured. The server searches for CRLs in the following locations:

1. Local file system

2. Oracle Internet Directory

3. **CRL Distribution Point**, a location specified in the CRL Distribution Point (CRL DP) X.509, version 3, certificate extension when the certificate is issued.

> **See Also:** "Certificate Validation with Certificate Revocation Lists" on page 8-24 for information about configuring and managing this PKI component

> **Note:** To use CRLs with other Oracle products, refer to the specific product documentation. This implementation of certificate validation with CRLs is only available in the Oracle Database 10*g* Release 2 (10.2) SSL adapter.

Wallets

A wallet is a container that is used to store authentication and signing credentials, including private keys, certificates, and trusted certificates needed by SSL. In an Oracle environment, every entity that communicates over SSL must have a wallet containing an X.509 version 3 certificate, private key, and list of trusted certificates, with the exception of Diffie-Hellman.

Security administrators use Oracle Wallet Manager to manage security credentials on the server. Wallet owners use it to manage security credentials on clients. Specifically, you use Oracle Wallet Manager to do the following:

- Generate a public-private key pair and create a certificate request

- Store a user certificate that matches with the private key

- Configure trusted certificates

> **Note:** Installation of Oracle Advanced Security 10*g* Release 2 (10.2) also installs Oracle Wallet Manager release 10.1.

> **See Also:**
> - Chapter 9, "Using Oracle Wallet Manager"
> - "Creating a New Wallet" on page 9-8
> - "Managing Trusted Certificates" on page 9-20

Hardware Security Modules

Oracle Advanced Security uses these devices for the following functions:

- Store cryptographic information, such as private keys

- Perform cryptographic operations to off load RSA operations from the server, freeing the CPU to respond to other transactions

Cryptographic information can be stored on two types of hardware devices:

- (Server-side) Hardware boxes where keys are stored in the box, but managed by using tokens.

- (Client-side) Smart card readers, which support storing private keys on tokens.

An Oracle environment supports hardware devices using APIs that conform to the RSA Security, Inc., Public-Key Cryptography Standards (PKCS) #11 specification.

> **Note:** Currently only nCipher devices are certified with Oracle Advanced Security. Certificate with other vendors is in progress.

> **See Also:** "Configuring Your System to Use Hardware Security Modules" on page 8-33 for details configuration details.

SSL Combined with Other Authentication Methods

You can configure Oracle Advanced Security to use SSL concurrently with database user names and passwords, RADIUS, and Kerberos, which are discussed in the following sections:

- Architecture: Oracle Advanced Security and SSL

- How SSL Works with Other Authentication Methods

> **See Also:** Appendix A, "Data Encryption and Integrity Parameters" for information about how to configure SSL with other supported authentication methods, including an example of a `sqlnet.ora` file with multiple authentication methods specified.

Architecture: Oracle Advanced Security and SSL

Figure 1–4 on page 1-10, which displays the Oracle Advanced Security implementation architecture, shows that Oracle Advanced Security operates at the **session layer** on top of SSL and uses TCP/IP at the **transport layer**. This separation of functionality lets you employ SSL concurrently with other supported protocols.

> **See Also:** *Oracle Database Net Services Administrator's Guide* for information about stack communications in an Oracle networking environment

How SSL Works with Other Authentication Methods

Figure 8–1 illustrates a configuration in which SSL is used in combination with another authentication method supported by Oracle Advanced Security.

Figure 8–1   SSL in Relation to Other Authentication Methods
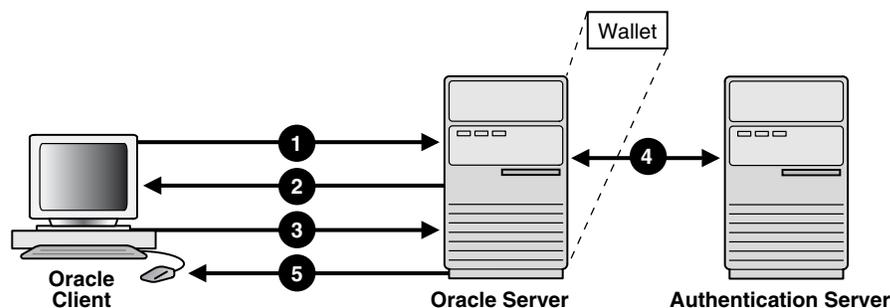
In this example, SSL is used to establish the initial handshake (server authentication), and an alternative authentication method is used to authenticate the client

1. The client seeks to connect to the Oracle database server.

2. SSL performs a handshake during which the server authenticates itself to the client and both the client and server establish which cipher suite to use.

3. Once the SSL handshake is successfully completed, the user seeks access to the database.

4. The Oracle database server authenticates the user with the authentication server using a non-SSL authentication method such as Kerberos or RADIUS.

5. Upon validation by the authentication server, the Oracle database server grants access and authorization to the user, and then the user can access the database securely by using SSL.

> **See Also:**   "How SSL Works in an Oracle Environment: The SSL Handshake" on page 8-2

SSL and Firewalls

Oracle Advanced Security supports two types of firewalls:

-   Application proxy-based firewalls, such as Network Associates Gauntlet, or Axent Raptor.

-   Stateful packet inspection firewalls, such as Check Point Firewall-1, or Cisco PIX Firewall.

When you enable SSL, stateful inspection firewalls behave like application proxy firewalls because they do not decrypt encrypted packets.

Firewalls do not inspect encrypted traffic. When a firewall encounters data addressed to an SSL port on an intranet server, it checks the target IP address against its access rules and lets the SSL packet pass through to permitted SSL ports, rejecting all others.

With the Oracle Net Firewall Proxy kit, a product offered by some firewall vendors, firewall applications can provide specific support for database network traffic. If the proxy kit is implemented in the firewall, then the following processing takes place:

-   The Net Proxy (a component of the Oracle Net Firewall Proxy kit) determines where to route its traffic.

-   The database listener requires access to a **certificate** in order to participate in the SSL handshake. The listener inspects the SSL packet and identifies the target database, returning the port on which the target database listens to the client. This port must be designated as an SSL port.

- The client communicates on this server-designated port in all subsequent connections.

- The number of ports that are open in the firewall increase as a function of the number of database connections requested for different databases. This approach prohibits the database server from using randomly chosen SSL ports, because the SSL ports on the firewall must match those chosen by the database. You can avoid this condition by deploying Oracle Connection Manager, an application included with Oracle Database Enterprise Edition.

Oracle Connection Manager lets you route client connections over multiple Oracle Net protocols. Each client connection request establishes an SSL connection between the client and Oracle Connection Manager, which in turn establishes a TCP/IP connection with the target database. Multiple clients can connect to multiple databases behind the firewall, using a single SSL port through the firewall.

Note: Although Oracle Connection Manager can be used to avoid opening up multiple SSL ports through the firewall, consider the following:

- The internal connection between Oracle Connection Manager and the database is not an SSL connection. You should encrypt such connections using Oracle Advanced Security native encryption.

- Because such connections do not use SSL, clients cannot use certificate-based authentication.

See Also: *Oracle Database Net Services Administrator's Guide* for information about Oracle Connection Manager

SSL Usage Issues

Consider the following issues when using SSL:

- SSL use enables secure communication with other Oracle products, such as Oracle Internet Directory.

- Because SSL supports both authentication and encryption, the client/server connection is somewhat slower than the standard Oracle Net TCP/IP transport (using native encryption).

- Each SSL authentication mode requires configuration settings.

Note:

- U.S. government regulations prohibit double encryption. Accordingly, if you configure Oracle Advanced Security to use SSL encryption and another encryption method concurrently, then the connection fails. You also cannot configure SSL authentication concurrently with non-SSL authentication.

- If you configure SSL encryption, you must disable non-SSL encryption. To disable such encryption, refer to "Disabling Oracle Advanced Security Authentication" on page 10-1.

See Also:

- "Configuring Your System to Use Hardware Security Modules" on page 8-33 for information about improving SSL performance with hardware accelerators

- "Enabling SSL" on page 8-9

Enabling SSL

To enable SSL, perform the following tasks:

- Task 1: Install Oracle Advanced Security and Related Products

- Task 2: Configure SSL on the Server

- Task 3: Configure SSL on the Client

- Task 4: Log on to the Database

Task 1: Install Oracle Advanced Security and Related Products

Install Oracle Advanced Security on both the client and server. When you do this, the Oracle Universal Installer automatically installs SSL libraries and Oracle Wallet Manager on your computer.

See Also: Oracle Database platform-specific installation documentation

Task 2: Configure SSL on the Server

During installation, Oracle sets defaults on both the Oracle database server and on the Oracle client for all SSL parameters except the location of the Oracle wallet. To configure SSL on the server, perform these steps:

- Step 1: Confirm Wallet Creation on the Server

- Step 2: Specify the Database Wallet Location on the Server

- Step 3: Set the SSL Cipher Suites on the Server (Optional)

- Step 4: Set the Required SSL Version on the Server (Optional)

- Step 5: Set SSL Client Authentication on the Server (Optional)

- Step 6: Set SSL as an Authentication Service on the Server (Optional)

- Step 7: Create a Listening Endpoint that Uses TCP/IP with SSL on the Server

See Also: Appendix B, "Authentication Parameters" for the dynamic parameter names

Step 1: Confirm Wallet Creation on the Server

Before proceeding to the next step, you must confirm that a wallet has been created. To confirm that your wallet is ready, open it by using Oracle Wallet Manager. The wallet should contain a certificate with a status of `Ready` and auto login turned on. If auto login is not on, then select it from the Wallet menu and save the wallet again. This turns auto login on.

See Also:

- "Opening an Existing Wallet" on page 9-9

- "Creating a New Wallet" on page 9-8

- "Using Auto Login" on page 9-14

Step 2: Specify the Database Wallet Location on the Server

Use Oracle Net Manager to specify required configuration parameters for the server
(Refer to "Starting Oracle Net Manager" on page 2-2):

1. Navigate to the Oracle Advanced Security profile. (Refer to "Navigating to the
   Oracle Advanced Security Profile" on page 2-2) The Oracle Advanced Security SSL
   window is displayed. (Figure 8–5).

2. Click the SSL tab and select **Configure SSL for: Server**.

3. In the **Wallet Directory** box, enter the directory in which the Oracle wallet is
   located or click **Browse** to find it by searching the file system.

   Note that if you are configuring the database-to-directory SSL connection for
   Enterprise User Security, then Database Configuration Assistant automatically
   creates a database wallet while registering the database with the directory. You
   must use that wallet to store the database PKI credentials for SSL-authenticated
   Enterprise User Security.

   Important:

   - Use Oracle Wallet Manager to create the wallet. Refer to
     "Creating a New Wallet" on page 9-8.

   - Use Oracle Net Manager to set the wallet location in the
     `sqlnet.ora` file.

   Ensure that you enter the same wallet location when you create it
   and when you set the location in the `sqlnet.ora` file.

   ---

1. Select **File**, **Save Network Configuration**.

   The `sqlnet.ora` and `listener.ora` files are updated with the following
   entries:

   ```
   wallet_location =
    (SOURCE=
     (METHOD=File)
     (METHOD_DATA=
      (DIRECTORY=wallet_location)))
   ```

   Note: The listener uses the wallet defined in the `listener.ora`
   file. It can use any database wallet. When SSL is configured for a
   server using Net Manager, the wallet location is entered into the
   `listener.ora` and the `sqlnet.ora` files. The `listener.ora`
   file is not relevant to the Oracle client.

   To change the listener wallet location so that the listener has its
   own wallet, you can edit `listener.ora` to enter the new location.

Step 3: Set the SSL Cipher Suites on the Server (Optional)

A cipher suite is a set of authentication, encryption, and data integrity algorithms used for exchanging messages between network entities. During an SSL handshake, two entities negotiate to see which cipher suite they will use when transmitting messages back and forth.

When you install Oracle Advanced Security, the SSL cipher suites listed in Table 8–1 are set for you by default and negotiated in the order they are listed. You can override the default order by setting the SSL_CIPHER_SUITES parameter. For example, if you use Oracle Net Manager to add the cipher suite SSL_RSA_WITH_RC4_128_SHA, all other cipher suites in the default setting are ignored.

You can prioritize the cipher suites. When the client negotiates with servers regarding which cipher suite to use, it follows the prioritization you set. When you prioritize the cipher suites, consider the following:

- Compatibility. Server and client must be configured to use compatible cipher suites for a successful connection.

- Cipher priority and strength. Prioritize cipher suites starting with the strongest and moving to the weakest to ensure the highest level of security possible.

- The level of security you want to use. For example, triple-DES encryption is stronger than DES

- The impact on performance. For example, triple-DES encryption is slower than DES.

> **Notes:** Regarding Diffie-Hellman anonymous authentication:
>
> 1. If you set the server to employ this cipher suite, then you must also set the same cipher suite on the client. Otherwise, the connection fails.
>
> 2. If you use a cipher suite employing Diffie-Hellman anonymous, then you must set the SSL_CLIENT_AUTHENTICATION parameter to FALSE. For more information, refer to "Step 5: Set SSL Client Authentication on the Server (Optional)" on page 8-13.
>
> 3. There is a known bug in which an OCI client requires a wallet even when using a cipher suite with DH_ANON, which does not authenticate the client.

Table 8–1 lists the SSL cipher suites supported in the current release of Oracle Advanced Security. These cipher suites are set by default when you install Oracle Advanced Security. The following table also lists the authentication, encryption, and data integrity types each cipher suite uses.

Table 8–1   Oracle Advanced Security Cipher Suites

Cipher Suites	Authentication	Encryption	Data Integrity
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES EDE CBC	SHA-1
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4 128	SHA-1
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4 128	MD5
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES CBC	SHA-1
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH anon	3DES EDE CBC	SHA-1
SSL_DH_anon_WITH_RC4_128_MD5	DH anon	RC4 128	MD5
SSL_DH_anon_WITH_DES_CBC_SHA	DH anon	DES CBC	SHA-1

Table 8–1   (Cont.) Oracle Advanced Security Cipher Suites

Cipher Suites	Authentication	Encryption	Data Integrity
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RSA	RC4 40	MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA	DES40 CBC	SHA-1
SSL_RSA_WITH_AES_128_CBC_SHA[1]	RSA	AES 128 CBC	SHA-1
SSL_RSA_WITH_AES_256_CBC_SHA[1]	RSA	AES 256 CBC	SHA-1

[1]   AES ciphers work with Transport Layer Security (TLS 1.0) only

To specify cipher suites for the server:

1. Click **Add**. A dialog box displays available cipher suites (Figure 8–2).

Figure 8–2   SSL Cipher Suites Window
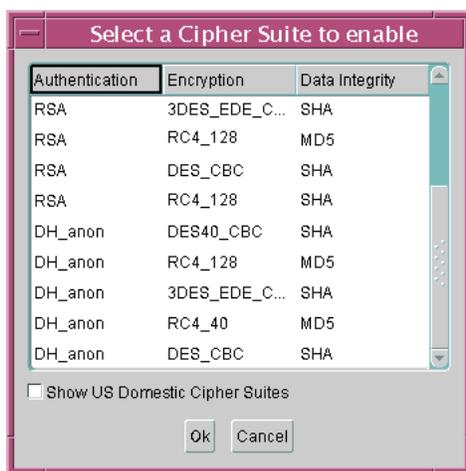
2. Select a suite and click **OK**. The **Cipher Suite Configuration** list is updated (Figure 8–3):

Figure 8–3   Oracle Advanced Security SSL Window (Server)
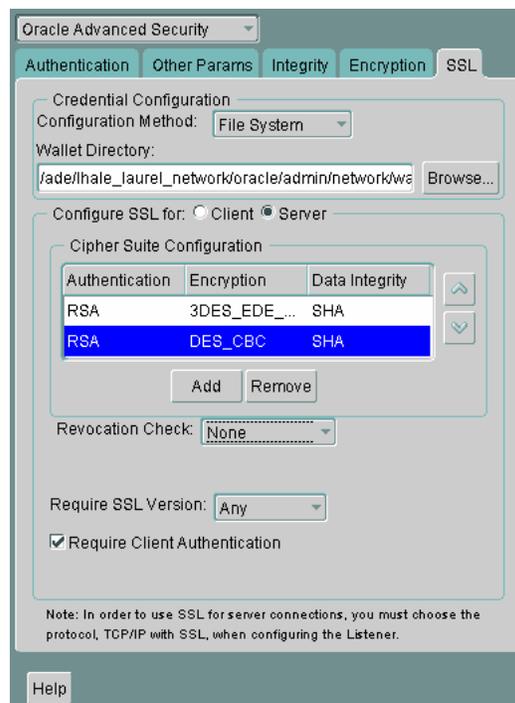
3. Use the up and down arrows to prioritize the cipher suites.

4. Select **File**, **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry:

   ```
   SSL_CIPHER_SUITES= (SSL_cipher_suite1 [,SSL_cipher_suite2])
   ```

Step 4: Set the Required SSL Version on the Server (Optional)

You can set the `SSL_VERSION` parameter in the `sqlnet.ora` file. This parameter defines the version of SSL that must run on the systems with which the server communicates. You can require these systems to use any valid version. The default setting for this parameter in `sqlnet.ora` is `undetermined`, which is set by selecting **Any** from the list in the SSL tab of the Oracle Advanced Security window.

To set the SSL version for the server:

1. In the Require SSL Version list, the default is **Any**. Accept this default or select the SSL version you want to use.

2. Select **File**, **Save Network Configuration**.

   If you chose **Any**, then the `sqlnet.ora` file is updated with the following entry:

   ```
   SSL_VERSION=UNDETERMINED
   ```

Note: SSL 2.0 is not supported on the server side.

Step 5: Set SSL Client Authentication on the Server (Optional)

The `SSL_CLIENT_AUTHENTICATION` parameter in the `sqlnet.ora` file controls whether the client is authenticated using SSL. The default value is `TRUE`.

You must set this parameter to `FALSE` if you are using a cipher suite that contains Diffie-Hellman anonymous authentication (`DH_anon`). Also, you can set this parameter to `FALSE` for the client to authenticate itself to the server by using any of the non-SSL authentication methods supported by Oracle Advanced Security, such as Kerberos or RADIUS.

> **Note:** There is a known bug in which an OCI client requires a wallet even when using a cipher suite with DH_ANON, which does not authenticate the client.

To set SSL_CLIENT_AUTHENTICATION to FALSE on the server:

Figure 8–4   Oracle Advanced Security SSL Window (Server)
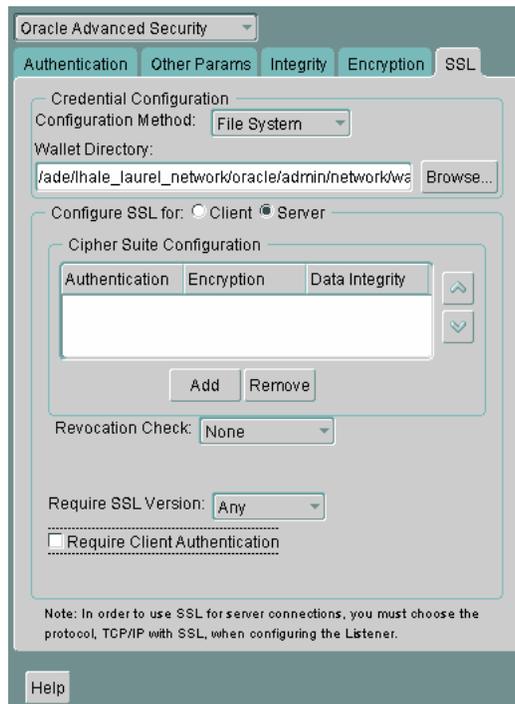
1. Deselect **Require Client Authentication**.

2. Select **File**, **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry:

   ```
   SSL_CLIENT_AUTHENTICATION=FALSE
   ```

Step 6: Set SSL as an Authentication Service on the Server (Optional)

The `SQLNET.AUTHENTICATION_SERVICES` parameter in the `sqlnet.ora` file sets the SSL authentication service.

Set this parameter if you want to use SSL authentication in conjunction with another authentication method supported by Oracle Advanced Security. For example, use this parameter if you want the server to authenticate itself to the client by using SSL and the client to authenticate itself to the server by using Kerberos.

To set the SQLNET.AUTHENTICATION_SERVICES parameter on the server:
Add TCP/IP with SSL (TCPS) to this parameter in the `sqlnet.ora` file by using a text editor. For example, if you want to use SSL authentication in conjunction with RADIUS authentication, set this parameter as follows:

```
SQLNET.AUTHENTICATION_SERVICES = (TCPS, radius)
```

If you do not want to use SSL authentication in conjunction with another authentication method, then do not set this parameter.

Step 7: Create a Listening Endpoint that Uses TCP/IP with SSL on the Server

Configure the listener with a TCP/IP with SSL listening endpoint in the `listener.ora` file. Oracle Corporation recommends using port number 2484 for typical Oracle Net clients.

> **See Also:**
>
> - *Oracle Database Net Services Administrator's Guide* for detailed information about configuring the `listener.ora` file
>
> - "Certificate Validation with Certificate Revocation Lists" on page 8-24 for information about configuring your system to validate certificates with certificate revocation lists

Task 3: Configure SSL on the Client

To configure SSL on the client:

- Step 1: Confirm Client Wallet Creation

- Step 2: Configure Oracle Net Service Name to Include Server DNs and Use TCP/IP with SSL on the Client

- Step 3: Specify Required Client SSL Configuration (Wallet Location)

- Step 4: Set the Client SSL Cipher Suites (Optional)

- Step 5: Set the Required SSL Version on the Client (Optional)

- Step 6: Set SSL as an Authentication Service on the Client (Optional)

> **See Also:** Appendix B, "Authentication Parameters" for the dynamic parameter names

Step 1: Confirm Client Wallet Creation

Before proceeding to the next step, you must confirm that a wallet has been created on the client and that the client has a valid certificate.

> **Note:** Oracle Corporation recommends that you use Oracle Wallet Manager to remove the **trusted certificate** in your Oracle wallet associated with each **certificate authority** that you do not use.

See Also:

- Chapter 9, "Using Oracle Wallet Manager", for general information about wallets

- "Opening an Existing Wallet" on page 9-9, for information about opening an existing wallet

- "Creating a New Wallet" on page 9-8, for information about creating a new wallet

Step 2: Configure Oracle Net Service Name to Include Server DNs and Use TCP/IP with SSL on the Client

You must specify the server's **distinguished name (DN)** and `TCPS` as the protocol in the client network configuration files to enable server DN matching and TCP/IP with SSL connections. Server DN matching prevents the database server from faking its identity to the client during connections by matching the server's global database name against the DN from the server certificate.

You must manually edit the client network configuration files, `tnsnames.ora` and `listener.ora`, to specify the server's DN and the TCP/IP with SSL protocol. The `tnsnames.ora` file can be located on the client or in the LDAP directory. If it is located on the client, then it typically resides in the same directory as the `listener.ora` file. Depending on the operating system, these files reside in the following directory locations:

- (UNIX) *$ORACLE_HOME*/network/admin/

- (Windows) *ORACLE_BASE\ORACLE_HOME*\network\admin\

To edit the `tnsnames.ora` and `listener.ora` files, use the following steps:

1. In the client `tnsnames.ora` file, add the `SSL_SERVER_CERT_DN` parameter and specify the database server's DN as follows:

   ```
   (SECURITY=
   (SSL_SERVER_CERT_DN="cn=finance,cn=OracleContext,c=us,o=acme"))
   ```

   The client uses this information to obtain the list of DNs it expects for each of the servers, enforcing the server's DN to match its service name. Example 8–1 shows an entry for the `Finance` database in the `tnsnames.ora` file.

   Alternatively, the administrator can ensure that the common name (CN) portion of the server's DN matches the service name.

2. In the client `tnsnames.ora` file, enter `tcps` as the PROTOCOL in the `ADDRESS` parameter. This specifies that the client will use TCP/IP with SSL to connect to the database that is identified in the `SERVICE_NAME` parameter. Example 8–1 also shows an entry that specifies TCP/IP with SSL as the connecting protocol in the `tnsnames.ora` file.

3. In the `listener.ora` file, enter `tcps` as the PROTOCOL in the `ADDRESS` parameter. Example 8–2 shows an entry that specifies TCP/IP with SSL as the protocol.

Example 8–1   Sample tnsnames.ora File with Server Certificate DN and TCP/IP with SSL Specified

```
finance=
(DESCRIPTION=
(ADDRESS_LIST=
(ADDRESS= (PROTOCOL = tcps) (HOST = finance_server) (PORT = 1575)))
```

```
(CONNECT_DATA=
(SERVICE_NAME= Finance.us.acme.com))
(SECURITY=
(SSL_SERVER_CERT_DN="cn=finance,cn=OracleContext,c=us,o=acme"))
```

Example 8–2   Sample listener.ora File with TCP/IP with SSL Specified as the Protocol

```
LISTENER=
(DESCRIPTION_LIST=
(DESCRIPTION=
(ADDRESS= (PROTOCOL = tcps) (HOST = finance_server) (PORT = 1575))))
```

Step 3: Specify Required Client SSL Configuration (Wallet Location)

Use Oracle Net Manager to specify required configuration parameters for the client (Refer to "Starting Oracle Net Manager" on page 2-2):

1. Navigate to the Oracle Advanced Security profile. (Refer to "Navigating to the Oracle Advanced Security Profile" on page 2-2) The Oracle Advanced Security SSL window is displayed (Figure 8–5):

Figure 8–5   Oracle Advanced Security SSL Window (Client)
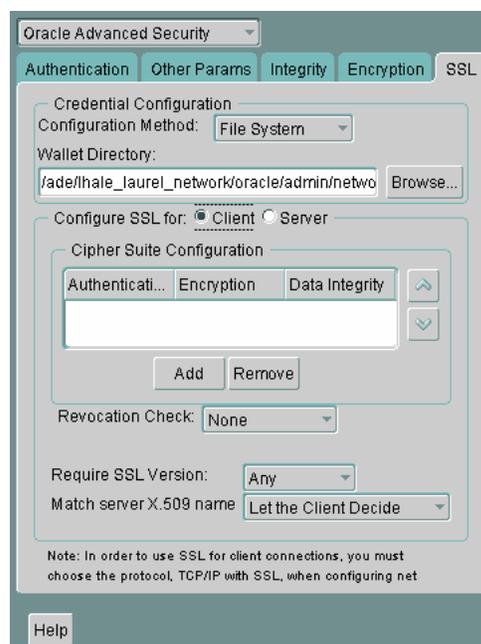
2. Click the **SSL** tab.

3. Select **Configure SSL for: Client**.

4. In the Wallet Directory box, enter the directory in which the Oracle wallet is located, or click **Browse** to find it by searching the file system.

5. From the Match server X.509 name list, select one of the following options:

    ■   **Yes:** Requires that the server's **distinguished name (DN)** match its service name. SSL ensures that the certificate is from the server and connections succeed only if there is a match.

> **Note:** This check can be made only when RSA ciphers are
> selected, which is the default setting.

- **No (default):** SSL checks for a match between the DN and the service name,
  but does not enforce it. Connections succeed regardless of the outcome but an
  error is logged if the match fails.

- **Let Client Decide:** Enables the default.

> **Note:** The following alert is displayed when you select No:
>
> Security Alert
>
> Not enforcing the server X.509 name match allows a server to
> potentially fake its identity. Oracle Corporation recommends
> selecting YES for this option so that connections are refused when
> there is a mismatch.

6. Select **File**, **Save Network Configuration**.

   The `sqlnet.ora` file on the client is updated with the following entries:

   ```
   SSL_CLIENT_AUTHENTICATION =TRUE
   wallet_location =
    (SOURCE=
     (METHOD=File)
     (METHOD_DATA=
      (DIRECTORY=wallet_location)))

   SSL_SERVER_DN_MATCH=(ON/OFF)
   ```

   > **See Also:**
   >
   > For information about the server match parameters:
   >
   > - "SSL X.509 Server Match Parameters" on page B-8
   >
   > For information about using Oracle Net Manager to configure
   > TCP/IP with SSL:
   >
   > - *Oracle Database Net Services Administrator's Guide*
   > - *Oracle Database Net Services Reference*

Step 4: Set the Client SSL Cipher Suites (Optional)

A cipher suite is a set of authentication, encryption, and data integrity algorithms used
for exchanging messages between network entities. During an SSL handshake, two
entities negotiate to see which cipher suite they will use when transmitting messages
back and forth.

When you install Oracle Advanced Security, the SSL cipher suites listed in Table 8–1
are set for you by default. This table lists them in the order they are tried when two
entities are negotiating a connection. You can override the default by setting the `SSL_
CIPHER_SUITES` parameter. For example, if you use Oracle Net Manager to add the
cipher suite `SSL_RSA_WITH_RC4_128_SHA`, all other cipher suites in the default
setting are ignored.

You can prioritize the cipher suites. When the client negotiates with servers regarding which cipher suite to use, it follows the prioritization you set. When you prioritize the cipher suites, consider the following:

- The level of security you want to use. For example, triple-DES encryption is stronger than DES.

- The impact on performance. For example, triple-DES encryption is slower than DES. Refer to "Configuring Your System to Use Hardware Security Modules" on page 8-33 for information about using SSL hardware accelerators with Oracle Advanced Security.

- Administrative requirements. The cipher suites selected for a client must be compatible with those required by the server. For example, in the case of an Oracle Call Interface (OCI) user, the server requires the client to authenticate itself. You cannot, in this case, use a cipher suite employing Diffie-Hellman anonymous authentication, which disallows the exchange of certificates.

You typically prioritize cipher suites starting with the strongest and moving to the weakest.

Table 8–1 lists the SSL cipher suites supported in the current release of Oracle Advanced Security. These cipher suites are set by default when you install Oracle Advanced Security. The table also lists the authentication, encryption, and data integrity types each cipher suite uses.

> **Note:** If the `SSL_CLIENT_AUTHENTICATION` parameter is set to `true` in the `sqlnet.ora` file, then disable all cipher suites that use Diffie-Hellman anonymous authentication. Otherwise, the connection fails.

To specify client cipher suites:

1. In the Cipher Suite Configuration region, click **Add**. A dialog box displays available cipher suites as shown in (Figure 8–2).

2. Select a suite and click **OK**. The **Cipher Suite Configuration** list is updated as shown in (Figure 8–6):

Figure 8–6   Oracle Advanced Security SSL Window (Client)
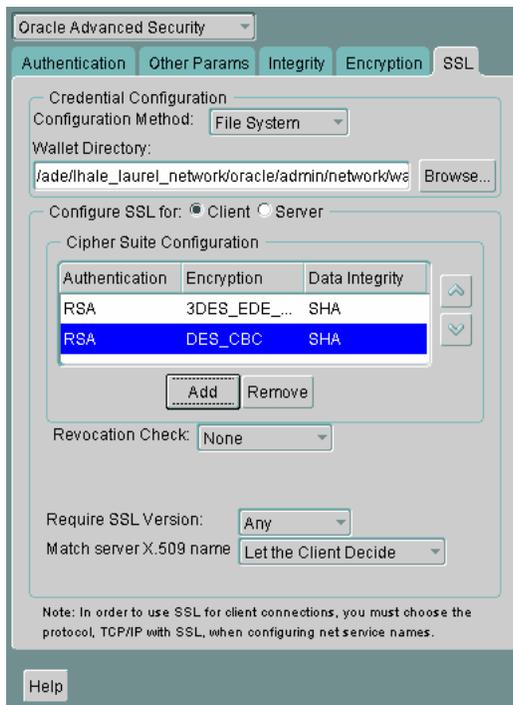
3. Use the up and down arrows to prioritize the cipher suites.

4. Select **File**, **Save Network Configuration**.

    The `sqlnet.ora` file is updated with the following entry:

    ```
    SSL_CIPHER_SUITES= (SSL_cipher_suite1 [,SSL_cipher_suite2])
    ```

Step 5: Set the Required SSL Version on the Client (Optional)

You can set the `SSL_VERSION` parameter in the `sqlnet.ora` file. This parameter defines the version of SSL that must run on the systems with which the client communicates. You can require these systems to use any valid version. The default setting for this parameter in `sqlnet.ora` is `undetermined`, which is set by selecting **Any** from the list in the **SSL** tab of the Oracle Advanced Security window. When **Any** is selected, TLS 1.0 is tried first, then SSL 3.0, and SSL 2.0 are tried in that order. Ensure that the client SSL version is compatible with the version the server uses.

To set the required SSL version for the client:

1. In the **Require SSL Version** list, the default setting is Any. Accept this default or select the SSL version you want to configure.

2. Select **File**, **Save Network Configuration**.

    The `sqlnet.ora` file is updated. If you selected **Any**, then it is updated with the following entry:

    ```
    SSL_VERSION=UNDETERMINED
    ```

Step 6: Set SSL as an Authentication Service on the Client (Optional)

The `SQLNET.AUTHENTICATION_SERVICES` parameter in the `sqlnet.ora` file sets the SSL authentication service. Typically, the `sqlnet.ora` file is located in the same

directory as the other network configuration files. Depending on the platform, the `sqlnet.ora` file is in the following directory location:

- (UNIX) *$ORACLE_HOME*/network/admin

- (Windows) *ORACLE_BASE**ORACLE_HOME*\network\admin\

Set the `SQLNET.AUTHENTICATION_SERVICES` parameter if you want to use SSL authentication in conjunction with another authentication method supported by Oracle Advanced Security. For example, use this parameter if you want the server to authenticate itself to the client by using SSL and the client to authenticate itself to the server by using RADIUS.

To set the client SQLNET.AUTHENTICATION_SERVICES parameter:
Add TCP/IP with SSL (`TCPS`) to this parameter in the `sqlnet.ora` file by using a text editor. For example, if you want to use SSL authentication in conjunction with RADIUS authentication, set this parameter as follows:

```
 SQLNET.AUTHENTICATION_SERVICES = (TCPS, radius)
```

If you do not want to use SSL authentication in conjunction with another authentication method, then do not set this parameter.

Task 4: Log on to the Database

If you are using SSL authentication for the client (`SSL_CLIENT_AUTHENTICATION=true` in the `listener.ora` file), then launch SQL*Plus and enter the following:

```
CONNECT/@net_service_name
```

If you are not using SSL authentication (`SSL_CLIENT_AUTHENTICATION=false` in the `listener.ora` file), then launch SQL*Plus and enter the following:

```
CONNECT username/password@net_service_name
```

> **See Also:** "Certificate Validation with Certificate Revocation Lists" on page 8-24 for information about configuring the client for certificate validation with certificate revocation lists

Troubleshooting SSL

The following section lists the most common errors you may receive while using the Oracle Advanced Security SSL adapter.

It may be necessary to enable Oracle Net tracing to determine the cause of an error. For information about setting tracing parameters to enable Oracle Net tracing, refer to *Oracle Database Net Services Administrator's Guide*.

ORA-28759: Failure to Open File

Cause: The system could not open the specified file. Typically, this error occurs because the wallet cannot be found.

Action: Check the following:

- Ensure that the correct wallet location is specified in the `sqlnet.ora` file. This should be the same directory location where you saved the wallet.

- Enable Oracle Net tracing to determine the name of the file that cannot be opened and the reason.

- Ensure that auto login was enabled when you saved the wallet. Refer to "Using Auto Login" on page 9-14

ORA-28786: Decryption of Encrypted Private Key Failure

Cause: An incorrect password was used to decrypt an encrypted private key. Frequently, this happens because an **auto login wallet** is not being used.

Action: Use Oracle Wallet Manager to turn the auto login feature on for the wallet. Then save the wallet again. Refer to, "Using Auto Login" on page 9-14.

ORA-28858: SSL Protocol Error

Cause: This is a generic error that can occur during SSL handshake negotiation between two processes.

Action: Enable Oracle Net tracing and attempt the connection again to produce trace output. Then contact Oracle customer support with the trace output.

ORA-28859 SSL Negotiation Failure

Cause: An error occurred during the negotiation between two processes as part of the SSL protocol. This error can occur when two sides of the connection do not support a common cipher suite.

Action: Check the following:

- Use Oracle Net Manager to ensure that the SSL versions on both the client and the server match, or are compatible. For example, if the server accepts only SSL 3.0 and the client accepts only TLS 1.0, then the SSL connection will fail.

- Use Oracle Net Manager to check what cipher suites are configured on the client and the server, and ensure that compatible cipher suites are set on both.

    See Also: "Step 4: Set the Client SSL Cipher Suites (Optional)" on page 8-18 for details about setting compatible cipher suites on the client and the server

---

Note: If you do not configure any cipher suites, then all available cipher suites are enabled.

ORA-28862: SSL Connection Failed

Cause: This error occurred because the peer closed the connection.

Action: Check the following:

- Ensure that the correct wallet location is specified in the `sqlnet.ora` file so the system can find the wallet.

- Use Oracle Net Manager to ensure that cipher suites are set correctly in the `sqlnet.ora` file. Sometimes this error occurs because the `sqlnet.ora` has been manually edited and the cipher suite names are misspelled. Ensure that case sensitive string matching is used with cipher suite names.

- Use Oracle Net Manager to ensure that the SSL versions on both the client and the server match or are compatible. Sometimes this error occurs because the SSL version specified on the server and client do not match. For example, if the server accepts only SSL 3.0 and the client accepts only TLS 1.0, then the SSL connection will fail.

- For more diagnostic information, enable Oracle Net tracing on the peer.

ORA-28865: SSL Connection Closed

Cause: The SSL connection closed because of an error in the underlying transport layer, or because the peer process quit unexpectedly.

Action: Check the following:

- Use Oracle Net Manager to ensure that the SSL versions on both the client and the server match, or are compatible. Sometimes this error occurs because the SSL version specified on the server and client do not match. For example, if the server accepts only SSL 3.0 and the client accepts only TLS 1.0, then the SSL connection will fail.

- If you are using a Diffie-Hellman anonymous cipher suite and the `SSL_CLIENT_AUTHENTICATION` parameter is set to `true` in the server's `listener.ora` file, then the client does not pass its certificate to the server. When the server does not receive the client's certificate, it (the server) cannot authenticate the client so the connection is closed. To resolve this use another cipher suite, or set this `listener.ora` parameter to false.

- Enable Oracle Net tracing and check the trace output for network errors.

- For details, refer to Actions listed for "ORA-28862: SSL Connection Failed" on page 8-22

ORA-28868: Peer Certificate Chain Check Failed

Cause: When the peer presented the **certificate chain**, it was checked and that check failed. This failure can be caused by a number of problems, including:

- One of the certificates in the chain has expired.

- A certificate authority for one of the certificates in the chain is not recognized as a **trust point**.

- The signature in one of the certificates cannot be verified.

Action: Refer to, "Opening an Existing Wallet" on page 9-9 to use Oracle Wallet Manager to open your wallet and check the following:

- Ensure that all of the certificates installed in your wallet are current (not expired).

- Ensure that a certificate authority's certificate from your peer's certificate chain is added as a **trusted certificate** in your wallet. Refer to, "Importing a Trusted Certificate" on page 9-21 to use Oracle Wallet Manager to import a trusted certificate.

ORA-28885: No certificate with the required key usage found.

Cause: Your certificate was not created with the appropriate X.509 version 3 key usage extension.

Action: Use Oracle Wallet Manager to check the certificate's key usage. Refer to, Table 9–1, " KeyUsage Values" on page 9-4.

ORA-29024: Certificate Validation Failure

Cause: The certificate sent by the other side could not be validated. This may occur if the certificate has expired, has been revoked, or is invalid for any other reason.

Action: Check the following:

- Check the certificate to determine whether it is valid. If necessary, get a new certificate, inform the sender that her certificate has failed, or resend.

- Check to ensure that the server's wallet has the appropriate **trust point**s to validate the client's certificate. If it does not, then use Oracle Wallet Manager to import the appropriate trust point into the wallet. Refer to, "Importing a Trusted Certificate" on page 9-21 for details.

- Ensure that the certificate has not been revoked and that certificate revocation list (CRL) checking is turned on. For details, refer to "Configuring Certificate Validation with Certificate Revocation Lists" on page 8-25

ORA-29223: Cannot Create Certificate Chain

Cause: A **certificate chain** cannot be created with the existing **trust point**s for the certificate being installed. Typically, this error is returned when the peer does not give the complete chain and you do not have the appropriate trust points to complete it.

Action: Use Oracle Wallet Manager to install the trust points that are required to complete the chain. Refer to,"Importing a Trusted Certificate" on page 9-21

Certificate Validation with Certificate Revocation Lists

The process of determining whether a given certificate can be used in a given context is referred to as certificate validation. Certificate validation includes determining that

- A trusted **certificate authority** (CA) has digitally signed the certificate

- The certificate's digital signature corresponds to the independently-calculated hash value of the certificate itself and the certificate signer's (CA's) public key

- The certificate has not expired

- The certificate has not been revoked

The SSL network layer automatically performs the first three validation checks, but you must configure certificate revocation list (CRL) checking to ensure that certificates have not been revoked. CRLs are signed data structures that contain a list of revoked certificates. They are usually issued and signed by the same entity who issued the original certificate. (Refer to, **certificate revocation lists**)

This section contains the following topics:

- What CRLs Should You Use?

- How CRL Checking Works

- Configuring Certificate Validation with Certificate Revocation Lists

- Certificate Revocation List Management

- Troubleshooting Certificate Validation

What CRLs Should You Use?

You should have CRLs for all of the trust points that you honor. The trust points are the trusted certificates from a third party identity that is qualified with a level of trust. Typically, the certificate authorities you trust are called trust points.

How CRL Checking Works

Certificate revocation status is checked against CRLs, which are located in file system directories, Oracle Internet Directory, or downloaded from the location specified in the **CRL Distribution Point** (CRL DP) extension on the certificate. Typically, CRL definitions are valid for a few days. If you store your CRLs on the local file system or

in the directory, then you must update them regularly. If you use CRL DPs then CRLs are downloaded each time a certificate is used, so there is no need to regularly refresh the CRLs.

The server searches for CRLs in the following locations in the order listed. When the system finds a CRL that matches the certificate CA's DN, it stops searching.

1. Local file system

   The system checks the `sqlnet.ora` file for the `SSL_CRL_FILE` parameter first, followed by the `SSL_CRL_PATH` parameter. If these two parameters are not specified, then the system checks the wallet location for any CRLs.

   > **Note:** Note: if you store CRLs on your local file system, then you must use the `orapki` utility to periodically update them. Fro more information, refer to "Renaming CRLs with a Hash Value for Certificate Validation" on page 8-28

2. Oracle Internet Directory

   If the server cannot locate the CRL on the local file system and directory connection information has been configured in an `ldap.ora` file, then the server searches in the directory. It searches the CRL subtree by using the CA's **distinguished name (DN)** and the DN of the CRL subtree.

   The server must have a properly configured `ldap.ora` file to search for CRLs in the directory. It cannot use the Domain Name System (DNS) discovery feature of Oracle Internet Directory. Also note that if you store CRLs in the directory, then you must use the `orapki` utility to periodically update them. For details, refer to "Uploading CRLs to Oracle Internet Directory" on page 8-29

3. CRL DP

   If the CA specifies a location in the CRL DP X.509, version 3, certificate extension when the certificate is issued, then the appropriate CRL that contains revocation information for that certificate is downloaded. Currently, Oracle Advanced Security supports downloading CRLs over HTTP and LDAP.

   > **Note:**
   >
   > - For performance reasons, only user certificates are checked.
   >
   > - Oracle recommends that you store CRLs in the directory rather than the local file system.

Configuring Certificate Validation with Certificate Revocation Lists

The `SSL_CERT_REVOCATION` parameter must be set to `REQUIRED` or `REQUESTED` in the `sqlnet.ora` file to enable certificate revocation status checking. By default this parameter is set to `NONE` indicating that certificate revocation status checking is turned off.

> **Note:** If you want to store CRLs on your local file system or in Oracle Internet Directory, then you must use the command line utility, `orapki`, to rename CRLs in your file system or upload them to the directory. Refer to, "Certificate Revocation List Management" on page 8-27 for information about using `orapki`.

To enable certificate revocation status checking for the client or the server:

Figure 8–7   Oracle Advanced Security SSL Window with Certificate Revocation Checking Selected
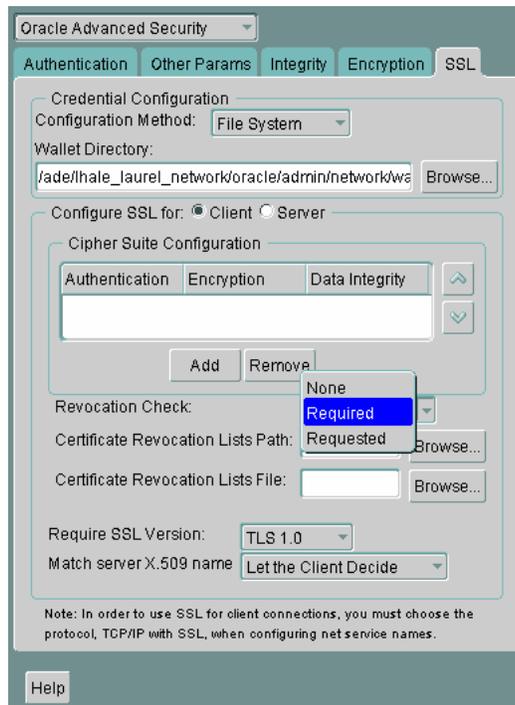
Following steps describe how to configure Certificate Validation with Certificate revocation Lists:

1. Select one of the following options from the **Revocation Check** list (refer to, Figure 8–7):

    ■   **REQUIRED**

        Requires certificate revocation status checking. The SSL connection is rejected if a certificate is revoked or no CRL is found. SSL connections are accepted only if it can be verified that the certificate has not been revoked.

    ■   **REQUESTED**

        Performs certificate revocation status checking if a CRL is available. The SSL connection is rejected if a certificate is revoked. SSL connections are accepted if no CRL is found or if the certificate has not been revoked.

    Note: For performance reasons, only user certificates are checked for revocation.

    ----

2. (Optional) If CRLs are stored on your local file system, then set one or both of the following fields that specify where they are stored. These fields are available only when **Revocation Check** is set to **REQUIRED** or **REQUESTED**.

    ■   **Certificate Revocation Lists Path:**

        Enter the path to the directory where CRLs are stored or click **Browse** to find it by searching the file system. Specifying this path sets the `SSL_CRL_PATH`

parameter in the `sqlnet.ora` file. If a path is not specified for this parameter, then the default is the wallet directory. Both DER-encoded (binary format) and **PEM**-encoded (BASE64) CRLs are supported.

- **Certificate Revocation Lists File:**

  Enter the path to a comprehensive CRL file (where PEM-encoded (BASE64) CRLs are concatenated in order of preference in one file) or click **Browse** to find it by searching the file system. Specifying this file sets the `SSL_CRL_FILE` parameter in the `sqlnet.ora` file. If this parameter is set, then the file must be present in the specified location, or else the application will error out during startup.

  > **Note:** If you want to store CRLs in a local file system directory by setting the **Certificate Revocation Lists Path**, then you must use the `orapki` utility to rename them so the system can locate them. Refer to, "Renaming CRLs with a Hash Value for Certificate Validation" on page 8-28

3. (Optional) If CRLs are fetched from Oracle Internet Directory, then directory server and port information must be specified in an `ldap.ora` file.

   > **Note:** When configuring your `ldap.ora` file, you should specify only a non-SSL port for the directory. CRL download is done as part of the SSL protocol, and making an SSL connection within an SSL connection is not supported.
   >
   > Oracle Advanced Security CRL functionality will not work if the Oracle Internet Directory non-SSL port is disabled.

4. Select **File**, **Save Network Configuration**. The `sqlnet.ora` file is updated.

To disable certificate revocation status checking:

1. Select **NONE** from the **Revocation Check** list.

2. Select **File**, **Save Network Configuration**. The `sqlnet.ora` file is updated with the following entry:

   ```
   SSL_CERT_REVOCATION=NONE
   ```

   > **See Also:** "Troubleshooting Certificate Validation" on page 8-31 for information about resolving certificate validation errors.

Certificate Revocation List Management

Before you can enable certificate revocation status checking, you must ensure that the CRLs you receive from the CAs you use are in a form (renamed with a hash value) or in a location (uploaded to the directory) where your computer can use them. Oracle Advanced Security provides a command-line utility, `orapki`, that you can use to perform the following tasks:

- Displaying `orapki` Help
- Renaming CRLs with a Hash Value for Certificate Validation
- Uploading CRLs to Oracle Internet Directory
- Listing CRLs Stored in Oracle Internet Directory

- Viewing CRLs in Oracle Internet Directory

- Deleting CRLs from Oracle Internet Directory

> **Note:** CRLs must be updated at regular intervals (before they expire) for successful validation. You can automate this task by using `orapki` commands in a script

You can also use LDAP command-line tools to manage CRLs in Oracle Internet Directory.

> **See Also:** Appendix A, "Syntax for Command-Line Tools" in *Oracle Internet Directory Application Developer's Guide* for information about LDAP command-line tools and their syntax

Displaying `orapki` **Help**

You can display all the `orapki` commands that are available for managing CRLs by entering the following at the command line:

```
orapki crl help
```

This command displays all available CRL management commands and their options.

> **Note:** Using the `-summary`, `-complete`, or `-wallet` command options is always optional. A command will still run if these command options are not specified.

Renaming CRLs with a Hash Value for Certificate Validation

When the system validates a certificate, it must locate the CRL issued by the CA who created the certificate. The system locates the appropriate CRL by matching the issuer name in the certificate with the issuer name in the CRL.

When you specify a CRL storage location for the **Certificate Revocation Lists Path** field in Oracle Net Manager,which sets the `SSL_CRL_PATH` parameter in the `sqlnet.ora` file, use the `orapki` utility to rename CRLs with a hash value that represents the issuer's name. Creating the hash value enables the server to load the CRLs.

On UNIX operating systems, `orapki` creates a symbolic link to the CRL. On Windows operating systems, it creates a copy of the CRL file. In either case, the symbolic link or the copy created by `orapki` are named with a hash value of the issuer's name. Then when the system validates a certificate, the same hash function is used to calculate the link (or copy) name so the appropriate CRL can be loaded.

Depending on the operating system, enter one of the following commands to rename CRLs stored in the file system.

To rename CRLs stored in UNIX file systems:

```
orapki crl hash -crl crl_filename [-wallet wallet_location] -symlink crl_directory
[-summary]
```
To rename CRLs stored in Windows file systems:

```
orapki crl hash -crl crl_filename [-wallet wallet_location] -copy crl_directory
[-summary]
```

where `crl_filename` is the name of the CRL file, `wallet_location` is the location of a wallet that contains the certificate of the CA that issued the CRL, and `crl_directory` is the directory where the CRL is located.

Using `-wallet` and `-summary` are optional. Specifying `-wallet` causes the tool to verify the validity of the CRL against the CA's certificate prior to renaming the CRL. Specifying the `-summary` option causes the tool to display the CRL issuer's name.

Uploading CRLs to Oracle Internet Directory

Publishing CRLs in the directory enables CRL validation throughout your enterprise, eliminating the need for individual applications to configure their own CRLs. All applications can use the CRLs stored in the directory where they can be centrally managed, greatly reducing the administrative overhead of CRL management and use.

The user who uploads CRLs to the directory by using `orapki` must be a member of the directory group CRLAdmins (`cn=CRLAdmins,cn=groups,%s_OracleContextDN%`). This is a privileged operation because these CRLs are accessible to the entire enterprise. Contact your directory administrator get added to this administrative directory group.

To upload CRLs to the directory, enter the following at the command line:

```
orapki crl upload -crl crl_location -ldap hostname:ssl_port -user username
[-wallet wallet_location] [-summary]
```

where `crl_location` is the file name or URL where the CRL is located, `hostname` and `ssl_port` (SSL port with no authentication) are for the system on which your directory is installed, `username` is the directory user who has permission to add CRLs to the CRL subtree, and `wallet_location` is the location of a wallet that contains the certificate of the CA that issued the CRL.

Using `-wallet` and `-summary` are optional. Specifying `-wallet` causes the tool to verify the validity of the CRL against the CA's certificate prior to uploading it to the directory. Specifying the `-summary` option causes the tool to print the CRL issuer's name and the LDAP entry where the CRL is stored in the directory.

The following example illustrates uploading a CRL with the `orapki` utility:

```
orapki crl upload -crl /home/user1/wallet/crldir/crl.txt -ldap
host1.oracle.com:3533 -user cn=orcladmin
```

> **Note:**
>
> ■ The `orapki` utility will prompt you for the directory password when you perform this operation.
>
> ■ Ensure that you specify the directory SSL port on which the Diffie-Hellman-based SSL server is running. This is the SSL port that does not perform authentication. Neither the server authentication nor the mutual authentication SSL ports are supported by the `orapki` utility.

Listing CRLs Stored in Oracle Internet Directory

You can display a list of all CRLs stored in the directory with `orapki`, which is useful for browsing to locate a particular CRL to view or download to your local computer. This command displays the CA who issued the CRL (Issuer) and its location (DN) in the CRL subtree of your directory.

To list CRLs in Oracle Internet Directory, enter the following at the command line:

```
orapki crl list -ldap hostname:ssl_port
```

where the *hostname* and *ssl_port* are for the system on which your directory is installed. Note that this is the directory SSL port with no authentication as described in the preceding section.

Viewing CRLs in Oracle Internet Directory

You can view specific CRLs that are stored in Oracle Internet Directory in a summarized format or you can request a complete listing of revoked certificates for the specified CRL. A summary listing provides the CRL issuer's name and its validity period. A complete listing provides a list of all revoked certificates contained in the CRL.

To view a summary listing of a CRL in Oracle Internet Directory, enter the following at the command line:

```
orapki crl display -crl crl_location [-wallet wallet_location] -summary
```

where *crl_location* is the location of the CRL in the directory. It is convenient to paste the CRL location from the list that displays when you use the `orapki crl list` command. Refer to,

To view a list of all revoked certificates contained in a specified CRL, which is stored in Oracle Internet Directory, enter the following at the command line:

```
orapki crl display -crl crl_location [-wallet wallet_location] -complete
```

For example, the following `orapki` command:

```
orapki crl display -crl $T_WORK/pki/wlt_crl/nzcrl.txt -wallet $T_WORK/pki/wlt_crl
-complete
```

produces the following output, which lists the CRL issuer's DN, its publication date, date of its next update, and the revoked certificates it contains:

```
issuer = CN=root,C=us, thisUpdate = Sun Nov 16 10:56:58 PST 2003, nextUpdate = Mon
Sep 30 11:56:58 PDT 2013, revokedCertificates = {(serialNo =
153328337133459399575438325845117876415, revocationDate - Sun Nov 16 10:56:58 PST
2003)}
CRL is valid
```

Using the `-wallet` option causes the `orapki crl display` command to validate the CRL against the CA's certificate.

Depending on the size of your CRL, choosing the `-complete` option may take a long time to display.

You can also use Oracle Directory Manager, a graphical user interface tool that is provided with Oracle Internet Directory, to view CRLs in the directory. CRLs are stored in the following directory location:

```
cn=CRLValidation,cn=Validation,cn=PKI,cn=Products,cn=OracleContext
```

Deleting CRLs from Oracle Internet Directory

The user who deletes CRLs from the directory by using `orapki` must be a member of the directory group `CRLAdmins`. Refer to for information about this directory administrative group.

To delete CRLs from the directory, enter the following at the command line:

```
orapki crl delete -issuer issuer_name -ldap host:ssl_port -user username
[-summary]
```

where *issuer_name* is the name of the CA who issued the CRL, the `hostname` and *ssl_port* are for the system on which your directory is installed, and `username` is the directory user who has permission to delete CRLs from the CRL subtree. Ensure that this must be a directory SSL port with no authentication. Refer to, "Uploading CRLs to Oracle Internet Directory" on page 8-29 for more information about this port.

Using the `-summary` option causes the tool to print the CRL LDAP entry that was deleted.

For example, the following `orapki` command:

```
orapki crl delete -issuer "CN=root,C=us" -ldap machine1:3500 -user cn=orcladmin
-summary
```

produces the following output, which lists the location of the deleted CRL in the directory:

```
Deleted CRL at cn=root
cd45860c.rN,cn=CRLValidation,cn=Validation,cn=PKI,cn=Products,cn=OracleContext
```

Troubleshooting Certificate Validation

To determine whether certificates are being validated against CRLs, you can enable Oracle Net tracing. When a revoked certificate is validated by using CRLs, then you will see the following entries in the Oracle Net tracing file without error messages logged between `entry` and `exit`:

```
nzcrlVCS_VerifyCRLSignature: entry
nzcrlVCS_VerifyCRLSignature: exit

nzcrlVCD_VerifyCRLDate: entry
nzcrlVCD_VerifyCRLDate: exit

nzcrlCCS_CheckCertStatus: entry
nzcrlCCS_CheckCertStatus: Certificate is listed in CRL
nzcrlCCS_CheckCertStatus: exit
```

> **Note:** Note that when certificate validation fails, the peer in the SSL handshake sees an `ORA-29024: Certificate Validation Failure`. If this message displays, refer to "ORA-29024: Certificate Validation Failure" on page 8-23 for information about how to resolve the error.

> **See Also:** *Oracle Database Net Services Administrator's Guide* for information about setting tracing parameters to enable Oracle Net tracing

Oracle Net Tracing File Error Messages Associated with Certificate Validation

The following trace messages, relevant to certificate validation, may be logged between the `entry` and `exit` entries in the Oracle Net tracing file. Oracle SSL looks for CRLs in multiple locations, so there may be multiple errors in the trace.

Check the following list of possible error messages for information about how to resolve them.

CRL signature verification failed with RSA status

Cause: The CRL signature cannot be verified.

Action: Ensure that the downloaded CRL is issued by the peer's CA and that the CRL was not corrupted when it was downloaded. Note that the `orapki` utility verifies the CRL before renaming it with a hash value or before uploading it to the directory.

> **See Also:** "Certificate Revocation List Management" on page 8-27 for information about using `orapki` for CRL management

CRL date verification failed with RSA status

Cause: The current time is later than the time listed in the next update field. You should not see this error if CRL DP is used. The systems searches for the CRL in the following order:

1. File system

2. Oracle Internet Directory

3. CRL DP

The first CRL found in this search may not be the latest.

Action: Update the CRL with the most recent copy.

CRL could not be found

Cause: The CRL could not be found at the configured locations. This will return error ORA-29024 if the configuration specifies that certificate validation is require.

Action: Ensure that the CRL locations specified in the configuration are correct by performing the following steps:

1. Use Oracle Net Manager to check if the correct CRL location is configured. Refer to "Configuring Certificate Validation with Certificate Revocation Lists" on page 8-25

2. If necessary, use the `orapki` utility to configure CRLs for system use as follows:

   – For CRLs stored on your local file system, refer to "Renaming CRLs with a Hash Value for Certificate Validation" on page 8-28

   – CRLs stored in the directory, refer to "Uploading CRLs to Oracle Internet Directory" on page 8-29

Oracle Internet Directory host name or port number not set

Cause: Oracle Internet Directory connection information is not set. Note that this is not a fatal error. The search continues with CRL DP.

Action: If you want to store the CRLs in Oracle Internet Directory, then use Oracle Net Configuration Assistant to create and configure an `ldap.ora` file for your Oracle home.

Fetch CRL from CRL DP: No CRLs found

Cause: The CRL could not be fetched by using the CRL DP. This happens if the certificate does not have a location specified in its CRL DP extension, or if the URL specified in the CRL DP extension is incorrect.

Action: Ensure that your certificate authority publishes the CRL to the URL that is specified in the certificate's CRL DP extension.

Manually download the CRL. Then depending on whether you want to store it on your local file system or in Oracle Internet Directory, perform the following steps:

If you want to store the CRL on your local file system:

1. Use Oracle Net Manager to specify the path to the CRL directory or file. Refer to "Configuring Certificate Validation with Certificate Revocation Lists" on page 8-25

2. Use the `orapki` utility to configure the CRL for system use. Refer to "Renaming CRLs with a Hash Value for Certificate Validation" on page 8-28

If you want to store the CRL in Oracle Internet Directory:

1. Use Oracle Net Configuration Assistant to create and configure an `ldap.ora` file with directory connection information.

2. Use the `orapki` utility to upload the CRL to the directory. Refer to "Uploading CRLs to Oracle Internet Directory" on page 8-29

Configuring Your System to Use Hardware Security Modules

Oracle Advanced Security supports hardware security modules that use APIs which conform to the RSA Security, Inc., PKCS #11 specification. Typically, these hardware devices are used to securely store and manage private keys in tokens or smart cards, or to accelerate cryptographic processing.

This section contains the following topics:

- General Guidelines for Using Hardware Security Modules with Oracle Advanced Security
- Configuring Your System to Use nCipher Hardware Security Modules
- Troubleshooting Using Hardware Security Modules

General Guidelines for Using Hardware Security Modules with Oracle Advanced Security

The following general guidelines apply if you are using a hardware security module with Oracle Advanced Security:

1. Contact your hardware device vendor to obtain the necessary hardware, software, and PKCS #11 libraries.

2. Install the hardware, software, and libraries where appropriate for the hardware security module you are using.

3. Test your hardware security module installation to ensure that it is operating correctly. Refer to your device documentation for instructions.

4. Create a wallet of the type `PKCS11` by using Oracle Wallet Manager and specify the absolute path to the PKCS #11 library (including the library name) if you wish to store the private key in the token. Oracle `PKCS11` wallets contain information that points to the token for private key access.

You can use the wallet containing PKCS #11 information just as you would use any Oracle wallet, except the private keys are stored on the hardware device and the cryptographic operations are performed on the device as well.

> **See Also:** "Creating a Wallet to Store Hardware Security Module Credentials" on page 9-8

Configuring Your System to Use nCipher Hardware Security Modules

Hardware security modules made by nCipher Corporation are certified to operate with Oracle Advanced Security. These modules provide a secure way to store keys and off load cryptographic processing. Primarily, these devices provide the following benefits:

- Off load of cryptographic processing to free your server to respond to more requests

- Secure private key storage on the device

- Administration of keys controlled through the use of smart cards

> **Note:** You must contact your nCipher representative to obtain certified hardware and software to use with Oracle Advanced Security.

Oracle Components Required To Use an nCipher Hardware Security Module

To use an nCipher hardware security module, you need the following components:

- nCipher Hardware Security Module

- Supporting nCipher PKCS #11 library for the platform as follows:
  - (UNIX 32-bit): `libcknfast.so` library
  - (UNIX 64-bit): `libcknfast-64.so` library
  - (Windows): `cknfast.dll` library

> **Note:** You must contact your nCipher representative to have the hardware security module or the secure accelerator installed and to acquire the necessary library.
>
> These tasks must be performed before you can use an nCipher hardware security module with Oracle Advanced Security.

About Installing an nCipher Hardware Security Module

To use the secure accelerator, you must provide the absolute path to the directory that contains the nCipher PKCS #11 library (including the library name) when you create the wallet by using Oracle Wallet Manager. This enables the library to be loaded at runtime. Typically, the nCipher card is installed at the following locations:

- (UNIX) `/opt/nfast`

- (Windows) `C:\nfast`

The nCipher PKCS #11 library is located at the following file system directory locations for typical installations:

- (UNIX 32-bit): `/opt/nfast/toolkits/pkcs11/libcknfast.so`

- (UNIX 64-bit): `/opt/nfast/toolkits/pkcs11/libcknfast-64.so`

- (Windows): `C:\nfast\toolkits\pkcs11\cknfast.dll`

> **Note:** Use the 32-bit library version when using the 32-bit release
> of Oracle Database and use the 64-bit library version when using
> the 64-bit release of Oracle Database. For example, use the 64-bit
> nCipher PKCS #11 library for the Oracle Database for Solaris
> Operating System (SPARC 64-bit).

Troubleshooting Using Hardware Security Modules

To detect whether the module is being used, you can turn on Oracle Net tracing. If the
wallet contains PKCS #11 information and the private key on the module is being
used, then you will see the following entries in the Oracle Net tracing file without
error messages logged between `entry` and `exit`:

```
nzpkcs11_Init: entry
nzpkcs11CP_ChangeProviders: entry
nzpkcs11CP_ChangeProviders: exit
nzpkcs11GPK_GetPrivateKey: entry
nzpkcs11GPK_GetPrivateKey: exit
nzpkcs11_Init: exit
...
nzpkcs11_Decrypt: entry
nzpkcs11_Decrypt: exit

nzpkcs11_Sign: entry
nzpkcs11_Sign: exit
```

> **See Also:** *Oracle Database Net Services Administrator's Guide* for
> information about setting tracing parameters to enable Oracle Net
> tracing

Error Messages Associated with Using Hardware Security Modules

The following errors are associated with using PKCS #11 hardware security modules:

ORA-43000: PKCS11: library not found

Cause: The system cannot locate the PKCS #11 library at the location specified
when the wallet was created. This happens only when the library is moved after
the wallet is created.

Action: Copy the PKCS #11 library back to its original location where it was when
the wallet was created.

ORA-43001: PKCS11: token not found

Cause: The smart card that was used to create the wallet is not present in the
hardware security module slot.

Action: Ensure that the smart card that was used when the wallet was created is
present in the hardware security module slot.

ORA-43002: PKCS11: passphrase is wrong

Cause: This can occur when an incorrect password is specified at wallet creation,
or the PKCS #11 device password is changed after the wallet is created and not
updated in the wallet by using Oracle Wallet Manager.

Action: Depending on the cause, take one of the following actions:

If you see this error during wallet creation, then check to ensure that you have the
correct password and reenter it.

If the password changed after wallet creation, then use Oracle Wallet Manager to open the wallet and enter a new password.

> **See Also:** "Creating a Wallet to Store Hardware Security Module Credentials" on page 9-8

> **Note:** The nCipher log file is in the directory where the module is installed at the following location:
>
> `/log/logfile`

> **See Also:** nCipher documentation for further information about troubleshooting

9

Using Oracle Wallet Manager

Security administrators use Oracle Wallet Manager to manage public key security credentials on Oracle clients and servers. The wallets it creates can be read by Oracle Database, Oracle Application Server 10g, and the Oracle Identity Management infrastructure.

This chapter describes Oracle Wallet Manager using the following topics:

- Oracle Wallet Manager Overview

- Starting Oracle Wallet Manager

- How to Create a Complete Wallet: Process Overview

- Managing Wallets

- Managing Certificates

> **See Also:**
>
> - "Public Key Infrastructure in an Oracle Environment" on page 8-3, which discusses all of the Oracle PKI components
>
> - Appendix F, "orapki Utility" for information about the `orapki` command line utility you can use to create wallets and issue certificates for testing purposes

Oracle Wallet Manager Overview

Oracle Wallet Manager is an application that wallet owners use to manage and edit the security credentials in their Oracle wallets. A wallet is a password-protected container used to store authentication and signing credentials, including private keys, certificates, and trusted certificates needed by SSL. You can use Oracle Wallet Manager to perform the following tasks:

- Creating wallets

- Generating certificate requests

- Opening wallets to access PKI-based services

- Saving credentials to hardware security modules, by using APIs that comply with the Public-Key Cryptography Standards #11 (**PKCS #11**) specification

- Uploading wallets to (and downloading them from) an LDAP directory

- Importing third-party **PKCS #12**-format wallets

- Exporting Oracle wallets to a third-party environment

Oracle Wallet Manager provides the following features:

- Wallet Password Management
- Strong Wallet Encryption
- Microsoft Windows Registry Wallet Storage
- Backward Compatibility
- Public-Key Cryptography Standards (PKCS) Support
- Multiple Certificate Support
- LDAP Directory Support

> **See Also:** "Public Key Infrastructure in an Oracle Environment" on page 8-3

Wallet Password Management

Oracle wallets are password protected. Oracle Wallet Manager includes an enhanced wallet password management module that enforces Password Management Policy guidelines, including the following:

- Minimum password length (8 characters)
- Maximum password length unlimited
- Alphanumeric character mix required

Strong Wallet Encryption

Oracle Wallet Manager stores private keys associated with X.509 certificates and uses Triple-DES encryption.

Microsoft Windows Registry Wallet Storage

Oracle Wallet Manager lets you store multiple Oracle wallets in a Windows file management system or in the user profile area of the Microsoft Windows system registry. Storing your wallets in the registry provides the following benefits:

- **Better Access Control:** Wallets stored in the user profile area of the registry are only accessible by the associated user. User access controls for the system thus become, by extension, access controls for the wallets. In addition, when a user logs out of a system, access to that user's wallets is effectively precluded.

- **Easier Administration:** Wallets are associated with specific user profiles, so no file permissions need to be managed, and the wallets stored in the profile are automatically deleted when the user profile is deleted. You can use Oracle Wallet Manager to create and manage the wallets in the registry.

Options Supported:

- Open a wallet from the registry
- Save a wallet to the registry
- Save As to a different registry location
- Delete a wallet from the registry
- Open a wallet from the file system and save it to the registry
- Open a wallet from the registry and save it to the file system

See Also: *Oracle Database Platform Guide for Windows*

Backward Compatibility

Oracle Wallet Manager is backward-compatible to Release 8.1.7.

Public-Key Cryptography Standards (PKCS) Support

RSA Laboratories, a division of RSA Security, Inc., has developed, in cooperation with representatives from industry, academia, and government, a family of basic cryptography standards called Public-Key Cryptography Standards, or PKCS for short. These standards establish interoperability between computer systems that use public-key technology to secure data across intranets and the Internet.

Oracle Wallet Manager stores X.509 certificates and **private keys** in PKCS #12 format, and generates certificate requests according to the PKCS #10 specification. These capabilities make the Oracle wallet structure interoperable with supported third-party PKI applications and provide wallet portability across operating systems.

Oracle Wallet Manager wallets can store credentials on hardware security modules that use APIs conforming to the PKCS #11 specification. When a wallet is created with PKCS11 chosen as the wallet type, then all keys stored in that wallet are saved to a hardware security module or token. Examples of such hardware devices include smart cards, **PCMCIA cards**, smart diskettes, or other portable hardware devices that store private keys or perform cryptographic operations (or both).

> **Note:** To use Oracle Wallet Manager with PKCS #11 integration on the 64-bit Solaris Operating System, enter the following at the command line: `owm -pkcs11`

See Also:

- "Importing User Certificates Created with a Third-Party Tool" on page 9-19

- "Exporting Oracle Wallets to Third-Party Environments" on page 9-10

- "Creating a Wallet to Store Hardware Security Module Credentials" on page 9-8

- To view PKCS standards documents, navigate to the following URL:

  `http://www.rsasecurity.com/rsalabs/`

Multiple Certificate Support

Oracle Wallet Manager enables you to store multiple **certificates** in each wallet, supporting any of the following **Oracle PKI certificate usages**:

- SSL authentication

- S/MIME signature

- S/MIME encryption

- Code-Signing

- CA Certificate Signing

Each certificate request you create generates a unique private/public key pair. The private key stays in the wallet and the public key is sent with the request to a certificate authority. When that certificate authority generates your certificate and signs it, you can import it only into the wallet that has the corresponding private key.

If the wallet also contains a separate certificate request, the private/public key pair corresponding to that request is of course different from the pair for the first certificate request. Sending this separate certificate request to a certificate authority can get you a separate signed certificate, which you can import into this same wallet

A single certificate request can be sent to a certificate authority multiple times to obtain multiple certificates. However, only one certificate corresponding to that certificate request can be installed in the wallet.

Oracle Wallet Manager uses the X.509 Version 3 `KeyUsage` extension to define Oracle PKI certificate usages (Table 9–1). A single certificate cannot be applied to all possible certificate usages. Table 9–2 and Table 9–3 show legal usage combinations.

Table 9–1   KeyUsage Values

Value	Usage
0	digitalSignature
1	nonRepudiation
2	keyEncipherment
3	dataEncipherment
4	keyAgreement
5	keyCertSign
6	cRLSign
7	encipherOnly
8	decipherOnly

When installing a certificate, Oracle Wallet Manager maps the `KeyUsage` extension values to Oracle PKI certificate usages as specified in Table 9–2 and Table 9–3.

Table 9–2   Oracle Wallet Manager Import of User Certificates to an Oracle Wallet

KeyUsage Value	Critical?[1]	Usage
none	NA	Certificate is importable for SSL or S/MIME encryption use.
0 alone or along with any values excluding 5 and 2	NA	Accept certificate for S/MIME signature or code-signing use.
1 alone	Yes	Not importable
1 alone	No	Accept certificate for S/MIME signature or code-signing use.
2 alone or along with any combination excluding 5	NA	Accept certificate for SSL or S/MIME encryption use.
5 alone or along with any other values	NA	Accept certificate for CA certificate signing use.
Any settings not listed previously	Yes	Not importable.

Table 9–2   (Cont.) Oracle Wallet Manager Import of User Certificates to an Oracle Wallet

KeyUsage Value	Critical?[1]	Usage
Any settings not listed previously	No	Certificate is importable for SSL or S/MIME encryption use.

[1]   If the `KeyUsage` extension is *critical*, the certificate cannot be used for other purposes.

Table 9–3   Oracle Wallet Manager Import of Trusted Certificates to an Oracle Wallet

KeyUsage Value	Critical?[1]	Usage
none	NA	Importable.
Any combination excluding 5	Yes	Not importable.
Any combination excluding 5	No	Importable
5 alone or along with any other values	NA	Importable.

[1]   If the `KeyUsage` extension is marked *critical*, the certificate cannot be used for other purposes.

You should obtain, from the certificate authority, certificates with the correct `KeyUsage` value matching your required Oracle PKI certificate usage. A single wallet can contain multiple **key pair**s for the same usage. Each certificate can support multiple Oracle PKI certificate usages, as indicated by Table 9–2 and Table 9–3. Oracle PKI applications use the first certificate containing the required PKI certificate usage.

For example, For SSL usage, the first certificate containing the SSL Oracle PKI certificate usage is used.

If you do not have a certificate with SSL usage, then an `ORA-28885` error (`No certificate with required key usage found`) is returned.

LDAP Directory Support

Oracle Wallet Manager can upload wallets to and retrieve them from an LDAP-compliant directory. Storing wallets in a centralized LDAP-compliant directory lets users access them from multiple locations or devices, ensuring consistent and reliable user authentication while providing centralized wallet management throughout the wallet life cycle. To prevent accidental overwrite of functional wallets, only wallets containing an installed certificate can be uploaded.

Directory user entries must be defined and configured in the LDAP directory before Oracle Wallet Manager can be used to upload or download wallets for a user. If a directory contains Oracle8*i* (or prior) users, then they are automatically upgraded to use the wallet upload and download feature on first use.

Oracle Wallet Manager downloads a user wallet by using a simple password-based connection to the LDAP directory. However, for uploads it uses an SSL connection if the open wallet contains a certificate with SSL Oracle PKI certificate usage. If an SSL certificate is not present in the wallet, password-based authentication is used.

> **Note:**   The directory password and the wallet password are independent and can be different. Oracle recommends that these passwords be maintained to be consistently different, where neither one can logically be derived from the other.

See Also:

- Uploading a Wallet to an LDAP Directory on page 9-11.
- Downloading a Wallet from an LDAP Directory on page 9-11
- Multiple Certificate Support on page 9-3, for more information about Oracle PKI certificate usage.

Starting Oracle Wallet Manager

To start Oracle Wallet Manager:

- (Windows) Select **Start**, **Programs**, **Oracle-HOME_NAME**, **Integrated Management Tools**, **Wallet Manager**
- (UNIX) At the command line, enter `owm`.

How to Create a Complete Wallet: Process Overview

Wallets provide a necessary repository in which you can securely store your user certificates and the **trust point** you need to validate the certificates of your peers.

The following steps provide an overview of the complete wallet creation process:

1. Use Oracle Wallet Manager to create a new wallet:

2. Generate a certificate request. Note that when you create a new wallet with Oracle Wallet Manager, the tool automatically prompts you to create a certificate request.

3. Send the certificate request to the CA you want to use. You can copy and paste the certificate request text into an e-mail message, or you can export the certificate request to a file. The certificate request becomes part of your wallet. It must remain there until you remove its associated certificate.

4. When the CA sends your signed user certificate and its associated **trusted certificate**, then you can import these certificates in the following order. The user certificates and trusted certificates in the PKCS #7 format can be imported at the same time.

    - First import the CA's trusted certificate into your wallet. This step may be optional if the new user certificate has been issued by one of the CAs whose trusted certificate is already present in Oracle Wallet Manager by default.

    - After you have successfully imported the trusted certificate, then import the user certificate that the CA sent to you into your wallet.

5. (Optional) Set the auto login feature for your wallet.

    Typically, this feature, which enables PKI-based access to services without a password, is required for most wallets. It is required for database server and client wallets. It is only optional for products that take the wallet password at the time of startup.

After completing the preceding process, you have a wallet that contains a user certificate and its associated trust points.

See Also: For more information about these steps, refer to **Managing Certificates** on page 8-20

Managing Wallets

This section describes how to create a new wallet and perform associated wallet management tasks, such as generating certificate requests, exporting certificate requests, and importing certificates into wallets, in the following subsections:

- Required Guidelines for Creating Wallet Passwords

- Creating a New Wallet

- Opening an Existing Wallet

- Closing a Wallet

- Exporting Oracle Wallets to Third-Party Environments

- Exporting Oracle Wallets to Tools that Do Not Support PKCS #12

- Uploading a Wallet to an LDAP Directory

- Downloading a Wallet from an LDAP Directory

- Saving Changes

- Saving the Open Wallet to a New Location

- Saving in System Default

- Deleting the Wallet

- Changing the Password

- Using Auto Login

Required Guidelines for Creating Wallet Passwords

Because an Oracle wallet contains user credentials that can be used to authenticate the user to multiple databases, it is especially important to choose a strong wallet password. A malicious user who guesses the wallet password can access all the databases to which the wallet owner has access.

Passwords must contain at least eight characters that consist of alphabetic characters combined with numbers or special characters.

> **Caution:** It is strongly recommended that users avoid choosing easily guessed passwords based on user names, phone numbers, or government identification numbers, such as "admin0," "oracle1," or "2135551212A." This prevents a potential attacker from using personal information to deduce the users' passwords. It is also a prudent security practice for users to change their passwords periodically, such as once in each month or once in each quarter.
>
> When you change passwords, you must regenerate auto-login wallets.

> **See Also:**
>
> - Wallet Password Management on page 9-2.
> - "Using Auto Login" on page 9-14

Creating a New Wallet

You can use Oracle Wallet Manager to create PKCS #12 wallets (the standard default wallet type) that store credentials in a directory on your file system. It can also be used to create PKCS #11 wallets that store credentials on a hardware security module for servers, or private keys on tokens for clients. The following sections explain how to create both types of wallets by using Oracle Wallet Manager.

Creating a Standard Wallet

Unless you have a hardware security module (a PKCS #11 device), then you should use a standard wallet that stores credentials in a directory on your file system.

To create a standard wallet, perform the following tasks:

1. Select **Wallet**, then **New** from the menu bar. The New Wallet dialog box is displayed.

2. Follow the "Required Guidelines for Creating Wallet Passwords" on page 9-7 and enter a password in the **Wallet Password** field. This password protects unauthorized use of your credentials.

3. Reenter that password in the **Confirm Password** field.

4. Select **Standard** from the **Wallet Type** list.

5. Click **OK** to continue. If the entered password does not conform to the required guidelines, then the following message is displayed:

   ```
   Password must have a minimum length of eight characters, and contain alphabetic
   characters combined with numbers or special characters. Do you want to try
   again?
   ```

6. An alert is displayed, and informs you that a new empty wallet has been created. It prompts you to decide whether you want to add a certificate request. Refer to "Adding a Certificate Request" on page 9-15.

   If you select **No**, then you are returned to the Oracle Wallet Manager main window. The new wallet you just created is displayed in the left window pane. The certificate has a status of **[Empty]**, and the wallet displays its default trusted certificates.

7. Select **Wallet**, then **Save In System Default** to save the new wallet.

   If you do not have permission to save the wallet in the system default, you can save it to another location. This location must be used in the SSL configuration for clients and servers.

   A message at the bottom of the window confirms that the wallet was successfully saved.

Creating a Wallet to Store Hardware Security Module Credentials

To create a wallet to store credentials on a hardware security module that complies with PKCS #11, perform the following tasks:

1. Select **Wallet**, then **New** from the menu bar. The New Wallet dialog box is displayed.

2. Follow the "Required Guidelines for Creating Wallet Passwords" on page 9-7 and enter a password in the **Wallet Password** field.

3. Reenter that password in the **Confirm Password** field.

4. Select **PKCS11** from the **Wallet Type** list, and click **OK** to continue. The New PKCS11 Wallet window is displayed.

5. Select a vendor name from the **Select Hardware Vendor** list.

> **Note:** In the current release of Oracle Wallet Manager, only nCipher hardware has been certified to interoperate with Oracle wallets.

6. In the **PKCS11 library filename** field, enter the path to the directory where the PKCS11 library is stored, or click **Browse** to find it by searching the file system.

7. Enter the **SmartCard password**, and click **OK**.

    The smart card password, which is different from the wallet password, is stored in the wallet.

8. An alert is displayed, and informs you that a new empty wallet has been created. It prompts you to decide whether you want to add a certificate request. For more information, refer to "Adding a Certificate Request" on page 9-15.

    If you select **No**, you are returned to the Oracle Wallet Manager main window. The new wallet you just created is displayed in the left window pane. The certificate has a status of **[Empty]**, and the wallet displays its default trusted certificates.

9. Select **Wallet**, then **Save In System Default** to save the new wallet.

    If you do not have permission to save the wallet in the system default, you can save it to another location.

    A message at the bottom of the window confirms that the wallet was successfully saved.

> **Note:** If you change the smart card password or move the PKCS #11 library, an error message displays when you try to open the wallet. Then you are prompted to enter the new smart card password or the new path to the library.

Opening an Existing Wallet

Open a wallet that already exists in the file system directory as follows:

1. Select **Wallet**, **Open** from the menu bar. The Select Directory dialog box is displayed.

2. Navigate to the directory location in which the wallet is located, and select the directory.

3. Click **OK**. The Open Wallet dialog box is displayed.

4. Enter the wallet password in the **Wallet Password** field.

5. Click **OK**.

    You are returned to the main window and a message is displayed at the bottom of the window indicating the wallet was opened successfully. The wallet's certificate and its trusted certificates are displayed in the left window pane.

Closing a Wallet

To close an open wallet in the currently selected directory:

Select **Wallet**, then **Close**.

A message is displayed at the bottom of the window to confirm that the wallet is closed.

Exporting Oracle Wallets to Third-Party Environments

Oracle Wallet Manager can export its own wallets to third-party environments.

To export a wallet to third-party environments:

1. Use Oracle Wallet Manager to save the wallet file.

2. Follow the procedure specific to your third-party product to import an operating system PKCS #12 wallet file created by Oracle Wallet Manager (called `ewallet.p12` on UNIX and Windows platforms).

> **Note:**
>
> - Oracle Wallet Manager supports multiple certificates for each wallet, yet current browsers typically support import of single-certificate wallets only. For these browsers, you must export an Oracle wallet containing a single key-pair.
>
> - Oracle Wallet Manager supports wallet export to only Netscape Communicator 4.7.2 and later, OpenSSL, and Microsoft Internet Explorer 5.0 and later.

Exporting Oracle Wallets to Tools that Do Not Support PKCS #12

You can export a wallet to a text-based PKI format if you want to put a wallet into a tool that does not support PKCS #12. Individual components are formatted according to the standards listed in Table 9–4. Within the wallet, only those certificates with SSL key usage are exported with the wallet.

To export a wallet to text-based PKI format:

1. Select **Operations**, **Export Wallet**. The Export Wallet dialog box is displayed.

2. Enter the destination file system directory for the wallet, or navigate to the directory structure under Folders.

3. Enter the destination file name for the wallet.

4. Click **OK** to return to the main window.

Table 9–4   PKI Wallet Encoding Standards

Component	Encoding Standard
Certificate chains	X509v3
Trusted certificates	X509v3
Private keys	PKCS #8

Uploading a Wallet to an LDAP Directory

To upload a wallet to an LDAP directory, Oracle Wallet Manager uses SSL if the specified wallet contains an SSL certificate. Otherwise, it lets you enter the directory password.

To prevent accidental destruction of your wallet, Oracle Wallet Manager will not permit you to execute the upload option unless the target wallet is currently open and contains at least one user certificate.

To upload a wallet:

1. Select **Wallet**, **Upload Into The Directory Service**. If the currently open wallet has not been saved, a dialog box is displayed with the following message:

   ```
   The wallet needs to be saved before uploading
   ```

   Click **Yes** to proceed.

2. Wallet certificates are checked for `SSL` key usage. Depending on whether a certificate with `SSL` key usage is found in the wallet, one of the following results occur:

   - **If at least one certificate has SSL key usage:** When prompted, enter the LDAP directory server host name and port information, then click **OK**. Oracle Wallet Manager attempts connection to the LDAP directory server using SSL. A message is displayed indicating whether the wallet was uploaded successfully or it failed.

   - **If no certificates have SSL key usage:** When prompted, enter the user's **distinguished name (DN)**, the LDAP server host name and port information, and click **OK**. Oracle Wallet Manager attempts connection to the LDAP directory server using simple password authentication mode, assuming that the wallet password is the same as the directory password.

     If the connection fails, a dialog box prompts for the directory password of the specified DN. Oracle Wallet Manager attempts connection to the LDAP directory server using this password and displays a warning message if the attempt fails. Otherwise, Oracle Wallet Manager displays a status message at the bottom of the window indicating that the upload was successful.

     Note:

     - You should ensure that the distinguished name used matches a corresponding user entry of object class `inetOrgPerson` in the LDAP directory.

     - When uploading a wallet with an SSL certificate, use the SSL port. When uploading a wallet that does not contain an SSL certificate, use the non-SSL port.

Downloading a Wallet from an LDAP Directory

When a wallet is downloaded from an LDAP directory, it is resident in working memory. It is not saved to the file system unless you explicitly save it using any of the save options described in the following sections.

See Also:

- "Saving Changes" on page 9-12
- "Saving the Open Wallet to a New Location" on page 9-12
- "Saving in System Default" on page 9-13

To download a wallet from an LDAP directory:

1. Select **Wallet**, **Download From The Directory Service...**.

2. A dialog box prompts for the user's distinguished name (DN), and the LDAP directory password, host name, and port information. Oracle Wallet Manager uses simple password authentication to connect to the LDAP directory.

   Depending on whether the downloading operation succeeds or not, one of the following results occurs:

   - **If the download operation fails:** Check to make sure that you have correctly entered the user's DN, and the LDAP server host name and port information. The port used must be the non-SSL port.

   - **If the download is successful:** Click **OK** to open the downloaded wallet. Oracle Wallet Manager attempts to open that wallet using the directory password. If the operation fails after using the directory password, then a dialog box prompts for the wallet password.

     If Oracle Wallet Manager cannot open the target wallet using the wallet password, then check to make sure you entered the correct password. Otherwise a message displays at the bottom of the window, indicating that the wallet was downloaded successfully.

Saving Changes

To save your changes to the current open wallet:

Select **Wallet**, then **Save**.

A message at the bottom of the window confirms that the wallet changes were successfully saved to the wallet in the selected directory location.

Saving the Open Wallet to a New Location

To save open wallets to a new location, use the **Save As** menu option:

1. Select **Wallet**, then **Save As**. The Select Directory dialog box is displayed.

2. Select a directory location in which to save the wallet.

3. Click **OK**.

   The following message is displayed if a wallet already exists in the selected location:

   ```
   A wallet already exists in the selected path. Do you want to overwrite it?
   ```

   Select **Yes** to overwrite the existing wallet or **No** to save the wallet to another location.

   A message at the bottom of the window confirms that the wallet was successfully saved to the selected directory location.

Saving in System Default

To save wallets in the default directory location, use the **Save In System Default** menu option:

Select **Wallet**, **Save In System Default**.

A message at the bottom of the window confirms that the wallet was successfully saved in the system default wallet location as follows for UNIX and Windows platforms:

- (UNIX) `etc/ORACLE/WALLETS/`*`username`*

- (Windows) *`%USERPROFILE%`*`\ORACLE\WALLETS`

> **Note:**
>
> - SSL uses the wallet that is saved in the system default directory location.
>
> - Some Oracle applications are not able to use the wallet if it is not in the system default location. Check the Oracle documentation for your specific application to determine whether wallets must be placed in the default wallet directory location.

Deleting the Wallet

To delete the current open wallet:

1. Select **Wallet**, **Delete**. The Delete Wallet dialog box is displayed.

2. Review the displayed wallet location to verify you are deleting the correct wallet.

3. Enter the wallet password.

4. Click **OK**. A dialog panel is displayed to inform you that the wallet was successfully deleted.

> **Note:** Any open wallet in application memory will remain in memory until the application exits. Therefore, deleting a wallet that is currently in use does not immediately affect system operation.

Changing the Password

A password change is effective immediately. The wallet is saved to the currently selected directory, with the new encrypted password.

> **Note:** If you are using a wallet with auto login enabled, you must regenerate the auto login wallet after changing the password. Refer to "Using Auto Login" on page 9-14

To change the password for the current open wallet:

1. Select **Wallet**, then **Change Password**. The Change Wallet Password dialog box is displayed.

2. Enter the existing wallet password.

3. Enter the new password.

4. Reenter the new password.

5. Click **OK**.

A message at the bottom of the window confirms that the password was successfully changed.

> **See Also:**
>
> - "Required Guidelines for Creating Wallet Passwords" on page 9-7
>
> - "Wallet Password Management" on page 9-2, for password policy restrictions

Using Auto Login

PKI-based access to services can be enabled without requiring human interventions to supply the necessary passwords: this feature is called auto login. Enabling auto login creates an obfuscated copy of the wallet, which is then used automatically until the auto login feature is disabled for that wallet.

Auto login wallets are protected by file system permissions. When auto login is enabled for a wallet, only the operating system user who created it can manage it, through the Oracle Wallet Manager.

You must enable auto login if you want single sign-on access to multiple Oracle databases: such access is normally disabled, by default. Sometimes the obfuscated auto login wallets are called "SSO wallets" because they support single sign-on capability.

Enabling Auto Login

To enable auto login:

1. Select **Wallet** from the menu bar.

2. Select **Auto Login**. A message at the bottom of the window indicates that auto login is enabled.

Disabling Auto Login

To disable auto login:

1. Select **Wallet** from the menu bar.

2. Deselct **Auto <u>L</u>ogin**. A message at the bottom of the window indicates that auto login is disabled.

Managing Certificates

All certificates are signed data structures that bind a network identity with a corresponding public key. Table 9–5 describes the two types of certificates distinguished in this chapter.

Table 9–5   Types of Certificates

Certificate Type	Examples
User certificates	Certificates issued to servers or users to prove an end entity's identity in a public key/private key exchange

Table 9–5   (Cont.) Types of Certificates

Certificate Type	Examples
Trusted certificates	Certificates representing entities whom you trust, such as certificate authorities who sign the user certificates they issue

The following subsections describe how to manage both types of certificates:

- Managing User Certificates

- Managing Trusted Certificates

> **Note:**   Before a user certificate can be installed, the wallet must contain the trusted certificate representing the certificate authority who issued that user certificate. However, whenever you create a new wallet, several publicly trusted certificates are automatically installed, since they are so widely used. If the necessary certificate authority is not represented, then you must install its certificate first.
>
> Also, you can import using the PKCS#7 certificate chain format, which gives you the user certificate and the CA certificate at the same time.

Managing User Certificates

User certificates, including server certificates, are used by end users, smart cards, or applications, such as Web servers. For example, if a CA issues a certificate for a Web server, placing its **distinguished name (DN)** in the Subject field, then the Web server is the certificate owner, thus the "user" for this user certificate.

Managing user certificates involves the following tasks:

- Adding a Certificate Request

- Importing the User Certificate into the Wallet

- Importing Certificates and Wallets Created by Third Parties

- Removing a User Certificate from a Wallet

- Removing a Certificate Request

- Exporting a User Certificate

- Exporting a User Certificate Request

Adding a Certificate Request

You can add multiple certificate requests with Oracle Wallet Manager. When adding multiple requests, Oracle Wallet Manager automatically populates each subsequent request dialog box with the content of the initial request that you can then edit.

The actual certificate request becomes part of the wallet. You can reuse any certificate request to obtain a new certificate. However, you cannot edit an existing certificate request. Store only a correctly filled out certificate request in a wallet.

To create a PKCS #10 certificate request:

1. Select **Operations**, then **Add Certificate Request**. The Add Certificate Request dialog box is displayed.

> **Note:** The online Help for Oracle Wallet Manager becomes unresponsive when modal dialog boxes appear, such as the one for entering certificate request information. The online Help becomes responsive once the modal dialog box is closed.

2. Enter the information specified in Table 9–6.

3. Click **OK**. A message informs you that a certificate request was successfully created. You can either copy the certificate request text from the body of this dialog panel and paste it into an e-mail message to send to a certificate authority, or you can export the certificate request to a file. At this point, Oracle Wallet Manager has created your private/public key pair and stored it in the wallet. When the certificate authority issues your certificate, it will also be stored in the wallet and associate it with its corresponding private key.

4. Click **OK** to return to the Oracle Wallet Manager main window. The status of the certificate changes to **[Requested]**.

> **See Also:**

Table 9–6   Certificate Request: Fields and Descriptions

Field Name	Description
Common Name	Mandatory. Enter the name of the user's or service's identity. Enter a user's name in first name /last name format. Example: Eileen.Sanger
Organizational Unit	Optional. Enter the name of the identity's organizational unit. Example: Finance.
Organization	Optional.Enter the name of the identity's organization. Example: XYZ Corp.
Locality/City	Optional. Enter the name of the locality or city in which the identity resides.
State/Province	Optional. Enter the full name of the state or province in which the identity resides. Enter the full state name, because some certificate authorities do not accept two–letter abbreviations.
Country	Mandatory. Select **Country** to view a list of country abbreviations. Select the country in which the organization is located.
Key Size	Mandatory. Select **Key Size** to view a list of key sizes to use when creating the public/private key pair. Refer to Table 9–7 to evaluate key size.
Advanced	Optional. Select **Advanced** to view the Advanced Certificate Request dialog panel. Use this field to edit or customize the identity's distinguished name (DN). For example, you can edit the full state name and locality.

Table 9–7 lists the available key sizes and the relative security each size provides. Typically, CAs use key sizes of 1024 or 2048. When certificate owners wish to keep their keys for a longer duration, they choose 3072 or 4096 bit keys.

Table 9–7   Available Key Sizes

Key Size	Relative Security Level
512 or 768	Not regarded as secure.
1024 or 2048	Secure.
3072 or 4096	Very secure.

Importing the User Certificate into the Wallet

When the Certificate Authority grants you a certificate, it may send you an e-mail that has your certificate in text (BASE64) form or attached as a binary file.

> **Note:** Certificate authorities may send your certificate in a PKCS #7 certificate chain or as an individual X.509 certificate. Oracle Wallet Manager can import both types.
>
> PKCS #7 certificate chains are a collection of certificates, including the user's certificate and all of the supporting trusted CA and subCA certificates.
>
> In contrast, an X.509 certificate file contains an individual certificate without the supporting certificate chain.
>
> However, before you can import any such individual certificate, the signer's certificate must be a Trusted Certificate in the wallet.

To import the user certificate from the text of the Certificate Authority's e-mail Copy the certificate, represented as text (BASE64), from the e-mail message. Include the lines `Begin Certificate` and `End Certificate`.

1. Select **Operations**, **Import User Certificate**. The Import Certificate dialog box is displayed.

2. Select **Paste the certificate**, and then click **OK**. Another Import Certificate dialog box is displayed with the following message:

    ```
    Please provide a base64 format certificate and paste it below.
    ```

3. Paste the certificate into the dialog box, and click **OK**.

    a. If the certificate received is in PKCS#7 format, it is installed, and all the other certificates included with the PKCS#7 data are placed in the Trusted Certificate list.

    b. If the certificate received is *not* in PKCS#7 format, and the certificate of its CA is not already in the Trusted Certificates list, then more must be done. Oracle Wallet Manager will ask you to import the certificate of the CA that issued your certificate. This CA certificate will be placed in the Trusted Certificates list. (If the CA certificate was already in the Trusted Certificates list, your certificate is imported without additional steps.)

    After either (a) or (b) succeeds, a message at the bottom of the window confirms that the certificate was successfully installed. You are returned to the Oracle Wallet Manager main panel, and the status of the corresponding entry in the left panel subtree changes to **[Ready]**.

> **Note:** The standard X.509 certificate includes the following start
> and end text:
>
> ```
> -----BEGIN CERTIFICATE-----
> -----END CERTIFICATE-----
> ```
>
> A typical PKCS#7 certificate includes more, as described earlier,
> and includes the following start and end text:
>
> ```
> -----BEGIN PKCS7-----
> -----END PKCS7-----
> ```
>
> You can use the standard Ctrl+c to copy, including all dashes, and
> Ctrl+v to paste.

To import the certificate from a file The user certificate in the file can be in either text
(BASE64) or binary (der) format.

1. Select **Operations**, **Import User Certificate**. The Import Certificate dialog box is
    displayed.

2. Select **Select a file that contains the certificate**, and click **OK**. Another Import
    Certificate dialog box is displayed.

3. Enter the path or folder name of the certificate file location.

4. Select the name of the certificate file (for example, cert.txt, cert.der).

5. Click **OK**.

    a. If the certificate received is in PKCS#7 format, it is installed, and all the other
        certificates included with the PKCS#7 data are placed in the Trusted
        Certificate list.

    b. If the certificate received is *not* in PKCS#7 format, and the certificate of its CA
        is not already in the Trusted Certificates list, then more must be done. Oracle
        Wallet Manager will ask you to import the certificate of the CA that issued
        your certificate. This CA certificate will be placed in the Trusted Certificates
        list. (If the CA certificate was already in the Trusted Certificates list, your
        certificate is imported without additional steps.)

    After either (a) or (b) succeeds, a message at the bottom of the window confirms
    that the certificate was successfully installed. You are returned to the Oracle Wallet
    Manager main panel, and the status of the corresponding entry in the left panel
    subtree changes to **[Ready]**.

Importing Certificates and Wallets Created by Third Parties

Third-party certificates are those created from certificate requests that were not
generated using Oracle Wallet Manager. These third-party certificates are actually
wallets, in the Oracle sense, because they contain more than just the user certificate;
they also contain the private key for that certificate. Furthermore, they include the
chain of trusted certificates validating that the certificate was created by a trustworthy
entity.

Oracle Wallet Manager makes these wallets available in a single step by importing
them in PKCS#12 format, which includes all three elements described earlier: the user
certificate, the private key, and the trusted certificates. It supports the following PKCS
#12-format certificates:

■   Netscape Communicator 4.x and later

- Microsoft Internet Explorer 5.x and later

Oracle Wallet Manager adheres to the PKCS#12 standard, so certificates exported by any PKCS#12-compliant tool should be usable with Oracle Wallet Manager.

Such third-party certificates cannot be stored into existing Oracle wallets because they would lack the private key and chain of trusted authorities. Therefore, each such certificate is exported and retrieved instead as an independent PKCS#12 file, that is, as its own wallet.

Importing User Certificates Created with a Third-Party Tool Once a third party generates the wallet, you need to import it to make use of it, as described in this section.

To import a certificate created with a third-party tool, perform the following tasks:

1. Follow the procedures for your particular product to export the certificate. Take the actions indicated in the exporting product to include the private key in the export, and specify the new password to protect the exported certificate. Also include all associated trust points. (Under PKCS #12, browsers do not necessarily export **trusted certificate**s, other than the signer's own certificate. You may need to add additional certificates to authenticate to your peers. You can use Oracle Wallet Manager to import trusted certificates.)

   The resulting file, containing the certificate, the private key, and the trust points, is the new wallet that enables the third-party certificate to be used.

2. To be used by particular applications or servers, such as a web server or an LDAP server, wallets need to be located precisely. Each application has its own expectations as to which directory it will search to find the needed wallet. You must put the wallet where it will be sought, by copying it to the correct machine and directory.

3. For use with UNIX or Windows applications or servers, the wallet must be named `ewallet.p12`.

   For other operating systems, refer to the Oracle documentation for that specific operating system.

   Once a third-party certificate is stored as `ewallet.p12`, you can open and manage it using Oracle Wallet Manager. You will have to supply the password you created when exporting this wallet.

   > **Note:** The password will be required whenever the associated application starts up or otherwise needs the certificate. To make such access automatic, refer to "Using Auto Login" on page 9-14.
   >
   > However, if the private key for the desired certificate is held in a separate hardware security module,you will not be able to import that certificate.

Removing a User Certificate from a Wallet

To remove a user certificate from a wallet:

1. In the left panel subtree, select the certificate that you want to remove.

2. Select **Operations**, then **Remove User Certificate**. A dialog panel is displayed which prompts you to verify that you want to remove the user certificate from the wallet.

3. Select **Yes** to return to the Oracle Wallet Manager main panel. The certificate displays a status of **[Requested]**.

Removing a Certificate Request

You must remove a certificate before removing its associated request.

To remove a certificate request:

1. In the left panel subtree, select the certificate request that you want to remove.

2. Select **Operations**, then **Remove Certificate Request**.

3. Click **Yes**. The certificate displays a status of **[Empty]**.

Exporting a User Certificate

To save the certificate in a file system directory, export the certificate by using the following steps:

1. In the left panel subtree, select the certificate that you want to export.

2. Select **Operations**, then **Export User Certificate** from the menu bar. The Export Certificate dialog box is displayed.

3. Enter the file system directory location where you want to save your certificate, or navigate to the directory structure under **Folders**.

4. Enter a file name for your certificate in the **Enter File Name** field.

5. Click **OK**. A message at the bottom of the window confirms that the certificate was successfully exported to the file. You are returned to the Oracle Wallet Manager main window.

> **See Also:** "Exporting Oracle Wallets to Third-Party Environments" on page 9-10 for information about exporting wallets. Note that Oracle Wallet Manager supports storing multiple certificates in a single wallet, yet current browsers typically support only single-certificate wallets. For these browsers, you must export an Oracle wallet that contains a single key-pair.

Exporting a User Certificate Request

To save the certificate request in a file system directory, export the certificate request by using the following steps:

1. In the left panel subtree, select the certificate request that you want to export.

2. Select **Operations**, then **Export Certificate Request**. The Export Certificate Request dialog box is displayed.

3. Enter the file system directory location where you want to save your certificate request, or navigate to the directory structure under Folders.

4. Enter a file name for your certificate request, in the **Enter File Name** field.

5. Select **OK**. A message at the bottom of the window confirms that the certificate request was successfully exported to the file. You are returned to the Oracle Wallet Manager main window.

Managing Trusted Certificates

Managing trusted certificates includes the following tasks:

- Importing a Trusted Certificate
- Removing a Trusted Certificate
- Exporting a Trusted Certificate
- Exporting All Trusted Certificates

Importing a Trusted Certificate

You can import a trusted certificate into a wallet in either of two ways: paste the trusted certificate from an e-mail that you receive from the certificate authority, or import the trusted certificate from a file.

Oracle Wallet Manager automatically installs trusted certificates from VeriSign, RSA, Entrust, and GTE CyberTrust when you create a new wallet.

To copy and paste the text only (BASE64) trusted certificate

Copy the trusted certificate from the body of the e-mail message you received that contained the user certificate. Include the lines `Begin Certificate` and `End Certificate`.

1. Select **Operations**, then **Import Trusted Certificate** from the menu bar. The Import Trusted Certificate dialog panel is displayed.

2. Select **Paste the Certificate** and click **OK**. Another Import Trusted Certificate dialog panel is displayed with the following message:

   `Please provide a base64 format certificate and paste it below.`

3. Paste the certificate into the window, and click **OK**. A message at the bottom of the window informs you that the trusted certificate was successfully installed.

4. Click **OK**. You are returned to the Oracle Wallet Manager main panel, and the trusted certificate is displayed at the bottom of the Trusted Certificates tree.

Keyboard shortcuts for copying and pasting certificates:
Use Ctrl+c to copy, and use Ctrl+v to paste.

To import a file that contains the trusted certificate The file containing the trusted certificate should have been saved in either text (BASE64) or binary (`der`) format.

1. Select **Operations**, then **Import Trusted Certificate**. The Import Trusted Certificate dialog panel is displayed.

2. Enter the path or folder name of the trusted certificate location.

3. Select the name of the trusted certificate file (for example, `cert.txt`).

4. Click **OK**. A message at the bottom of the window informs you that the trusted certificate was successfully imported into the wallet.

5. Click **OK** to exit the dialog panel. You are returned to the Oracle Wallet Manager main panel, and the trusted certificate is displayed at the bottom of the Trusted Certificates tree.

Removing a Trusted Certificate

You cannot remove a trusted certificate if it has been used to sign a user certificate still present in the wallet. To remove such trusted certificates, you must first remove the

certificates it has signed. Also, you cannot verify a certificate after its trusted certificate has been removed from your wallet.

To remove a trusted certificate from a wallet:

1. Select the trusted certificate listed in the Trusted Certificates tree.

2. Select **Operations**, then **Remove Trusted Certificate...** from the menu bar.

   A dialog panel warns you that your user certificate will no longer be verifiable by its recipients if you remove the trusted certificate that was used to sign it.

3. Select **Yes**. The selected trusted certificate is removed from the Trusted Certificates tree.

Exporting a Trusted Certificate

To export a trusted certificate to another file system location:

1. In the left panel subtree, select the trusted certificate that you want to export.

2. Select **Operations**, then**Export Trusted Certificate**. The Export Trusted Certificate dialog box is displayed.

3. Enter a file system directory in which you want to save your trusted certificate, or navigate to the directory structure under Folders.

4. Enter a file name to save your trusted certificate.

5. Click **OK**. You are returned to the Oracle Wallet Manager main window.

Exporting All Trusted Certificates

To export all of your trusted certificates to another file system location:

1. Select **Operations**, then **Export** <u>A</u>**ll Trusted Certificates...**. The Export Trusted Certificate dialog box is displayed.

2. Enter a file system directory location where you want to save your trusted certificates, or navigate to the directory structure under **Folders**.

3. Enter a file name to save your trusted certificates.

4. Click **OK**. You are returned to the Oracle Wallet Manager main window.

10

Configuring Multiple Authentication Methods and Disabling Oracle Advanced Security

This chapter describes how to configure multiple authentication methods under Oracle Advanced Security, and how to use conventional user name and password authentication, even if you have configured another authentication method. This also chapter describes how to configure your network so that Oracle clients can use a specific authentication method and Oracle servers can accept any method specified.

This chapter contains the following topics:

- Connecting with User Name and Password
- Disabling Oracle Advanced Security Authentication
- Configuring Multiple Authentication Methods
- Configuring Oracle Database for External Authentication

Connecting with User Name and Password

To connect to an Oracle database server using a user name and password when an Oracle Advanced Security authentication method has been configured, disable the external authentication (Refer to "Disabling Oracle Advanced Security Authentication" on page 10-1).

With the external authentication disabled, a user can connect to a database using the following format:

```
% sqlplus username/password@net_service_name
```

For example:

```
% sqlplus scott/tiger@emp
```

> **Note:** You can configure multiple authentication methods, including both externally authenticated users and password authenticated users, on a single database.


Disabling Oracle Advanced Security Authentication

Use Oracle Net Manager to disable authentication methods (Refer to "Starting Oracle Net Manager" on page 2-2):

1. Navigate to the Oracle Advanced Security profile. Refer to "Navigating to the Oracle Advanced Security Profile" on page 2-2. The Oracle Advanced Security tabbed window is displayed as shown in Figure 10–1.

Figure 10–1   Oracle Advanced Security Authentication Window
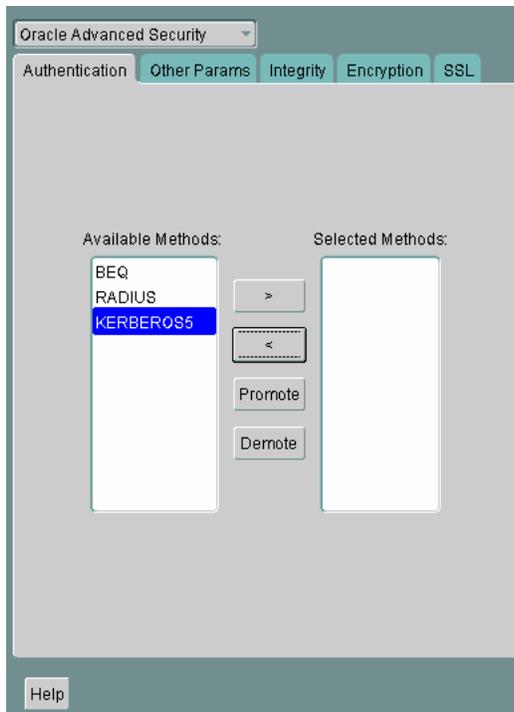
1. Click the **Authentication** tab.

2. Sequentially move all authentication methods from the Selected Method list to the Available Methods list by selecting a method and choosing the left arrow [<].

3. Select **File**, then **Save Network Configuration**.

    The `sqlnet.ora` file is updated with the following entry:

    ```
    SQLNET.AUTHENTICATION_SERVICES = (NONE)
    ```

Configuring Multiple Authentication Methods

Many networks use more than one authentication method on a single security server. Accordingly, Oracle Advanced Security lets you configure your network so that Oracle clients can use a specific authentication method, and Oracle database servers can accept any method specified.

You can set up multiple authentication methods on both client and server systems either by using Oracle Net Manager, or by using any text editor to modify the `sqlnet.ora` file.

Use Oracle Net Manager to add authentication methods to both clients and servers (Refer to "Starting Oracle Net Manager" on page 2-2)

Following steps describe how to configure Multiple authentication Methods.

1. Navigate to the Oracle Advanced Security profile. Refer to "Navigating to the Oracle Advanced Security Profile" on page 2-2. The Oracle Advanced Security tabbed window is displayed as shown in Figure 10–1.

2. Click the **Authentication** tab.

3. Select a method listed in the Available Methods list.

4. Sequentially move selected methods to the Selected Methods list by clicking the right arrow (>).

5. Arrange the selected methods in order of desired use. To do this, select a method in the Selected Methods list, and select **Promote** or **Demote** to position it in the list.

6. Select **File**, then **Save Network Configuration**.

   The `sqlnet.ora` file is updated with the following entry, listing the selected authentication methods:

   ```
   SQLNET.AUTHENTICATION_SERVICES = (RADIUS|KERBEROS5)
   ```

   > **Note:** SecurID functionality is available through RADIUS; RADIUS support is built into the RSA ACE/Server.

   > **See Also:** Chapter 6, "Configuring RADIUS Authentication" for more information

Configuring Oracle Database for External Authentication

This section describes the parameters you must set to configure Oracle Database for network authentication, using the following tasks:

- Setting the SQLNET.AUTHENTICATION_SERVICES Parameter in sqlnet.ora
- Verifying that REMOTE_OS_AUTHENT Is Not Set to TRUE
- Setting OS_AUTHENT_PREFIX to a Null Value

   > **See Also:**
   > - The corresponding chapter in this guide for information about configuring a particular authentication method
   > - Appendix B, "Authentication Parameters"

Setting the SQLNET.AUTHENTICATION_SERVICES Parameter in sqlnet.ora

The following parameter must be set in the `sqlnet.ora` file for all clients and servers to enable each to use a supported authentication method:

```
SQLNET.AUTHENTICATION_SERVICES=(oracle_authentication_method)
```

For example, for all clients and servers using Kerberos authentication, the `sqlnet.ora` parameter must be set as follows:

```
SQLNET.AUTHENTICATION_SERVICES=(KERBEROS5)
```

Verifying that REMOTE_OS_AUTHENT Is Not Set to TRUE

To verify that REMOVE_OS_AUTHENT is not set to TRUE, add the following parameter to the initialization file, in each database instance when you configure the authentication method:

```
REMOTE_OS_AUTHENT=FALSE
```

> **Caution:** Setting REMOTE_OS_AUTHENT to TRUE can cause a security exposure, because it lets someone using a non-secure protocol, such as TCP, perform an operating system-authorized login (formerly referred to as an OPS$ login).

> **Note:** In addition to setting the REMOTE_OS_AUTHENT initialization parameter file to FALSE, you should issue the startup command with a PFILE option. This ensures that the parameters from your initSID.ora are used.

If REMOTE_OS_AUTHENT is set to FALSE, and the server cannot support any of the authentication methods requested by the client, then the authentication service negotiation fails and the connection terminates.

If the parameter is set as follows in the sqlnet.ora file on either the client or server, the database attempts to use the supplied user name and password to login the user:

```
SQLNET.AUTHENTICATION_SERVICES=(NONE)
```

If REMOTE_OS_AUTHENT is set to FALSE, however, the connection fails.

Setting OS_AUTHENT_PREFIX to a Null Value

Authentication service-based user names can be long, and Oracle user names are limited to 30 characters. Oracle strongly recommends that you enter a null value for the OS_AUTHENT_PREFIX parameter in the initialization file used for the database instance as follows:

```
OS_AUTHENT_PREFIX=""
```

> **Note:** The default value for OS_AUTHENT_PREFIX is OPS$; however, you can set it to any string.

> **Attention:** If a database already has the OS_AUTHENT_PREFIX set to a value other than NULL (" "), *do not change it*, because it can inhibit previously created, externally identified users from connecting to the Oracle server.

To create a user, launch SQL*Plus and enter the following:

```
SQL> CREATE USER os_authent_prefix username IDENTIFIED EXTERNALLY;
```

When OS_AUTHENT_PREFIX is set to a null value (" "), enter the following to create the user king:

```
SQL> CREATE USER king IDENTIFIED EXTERNALLY;
```

The advantage of creating a user in this way is that the administrator no longer needs to maintain different user names for externally identified users. This is true for all supported authentication methods.

> **See Also:**
>
> - *Oracle Database Administrator's Guide*
> - *Oracle Database Heterogeneous Connectivity Administrator's Guide*

Part IV

Appendixes

Part IV contains the following reference appendixes:

- Appendix A, "Data Encryption and Integrity Parameters"
- Appendix B, "Authentication Parameters"
- Appendix C, "Integrating Authentication Devices Using RADIUS"
- Appendix D, "Oracle Advanced Security FIPS 140-1 Settings"
- Appendix E, "Oracle Advanced Security FIPS 140-2 Settings"
- Appendix F, "orapki Utility"
- Appendix G, "Entrust-Enabled SSL Authentication"

A

Data Encryption and Integrity Parameters

This appendix describes **encryption** and data **integrity** parameters supported by Oracle Advanced Security. It also includes an example of a `sqlnet.ora` file generated by performing the network configuration described in Chapter 4, "Configuring Network Data Encryption and Integrity for Oracle Servers and Clients" and Chapter 8, "Configuring Secure Sockets Layer Authentication".

This appendix contains the following topics:

- Sample sqlnet.ora File

- Data Encryption and Integrity Parameters

Sample sqlnet.ora File

This section contains a sample `sqlnet.ora` configuration file for a set of clients with similar characteristics and a set of servers with similar characteristics. The file includes examples of Oracle Advanced Security encryption and data integrity parameters.

Trace File Setup

```
#Trace file setup
trace_level_server=16
trace_level_client=16
trace_directory_server=/orant/network/trace
trace_directory_client=/orant/network/trace
trace_file_client=cli
trace_file_server=srv
trace_unique_client=true
```

Oracle Advanced Security Transparent Data Encryption

```
ENCRYPTION_WALLET_LOCATION = (SOURCE =
                              (METHOD = FILE)
                              (METHOD_DATA =
                              (DIRECTORY =
                               /oracle/dbsid/admin/pdcs11/wallet)))
```

Oracle Advanced Security Network Encryption

```
#ASO Encryption
sqlnet.encryption_server=accepted
sqlnet.encryption_client=requested
sqlnet.encryption_types_server=(RC4_40)
sqlnet.encryption_types_client=(RC4_40)
```

Oracle Advanced Security Network Data Integrity

```
#ASO Checksum
sqlnet.crypto_checksum_server=requested
sqlnet.crypto_checksum_client=requested
sqlnet.crypto_checksum_types_server = (MD5)
sqlnet.crypto_checksum_types_client = (MD5)
```

SSL

```
#SSL
WALLET_LOCATION = (SOURCE=
                            (METHOD = FILE)
                            (METHOD_DATA =
                             DIRECTORY=/wallet)

SSL_CIPHER_SUITES=(SSL_DH_anon_WITH_RC4_128_MD5)
SSL_VERSION= 3
SSL_CLIENT_AUTHENTICATION=FALSE
```

Common

```
#Common
automatic_ipc = off
sqlnet.authentication_services = (beq)
names.directory_path = (TNSNAMES)
```

Kerberos

```
#Kerberos
sqlnet.authentication_services = (beq, kerberos5)
sqlnet.authentication_kerberos5_service = oracle
sqlnet.kerberos5_conf= /krb5/krb.conf
sqlnet.kerberos5_keytab= /krb5/v5srvtab
sqlnet.kerberos5_realms= /krb5/krb.realm
sqlnet.kerberos5_cc_name = /krb5/krb5.cc
sqlnet.kerberos5_clockskew=900
sqlnet.kerberos5_conf_mit=false
```

RADIUS

```
#Radius
sqlnet.authentication_services = (beq, RADIUS )
sqlnet.radius_authentication_timeout = (10)
sqlnet.radius_authentication_retries = (2)
sqlnet.radius_authentication_port = (1645)
sqlnet.radius_send_accounting = OFF
sqlnet.radius_secret = /orant/network/admin/radius.key
sqlnet.radius_authentication = radius.us.oracle.com
sqlnet.radius_challenge_response = OFF
sqlnet.radius_challenge_keyword = challenge
sqlnet.radius_challenge_interface =
oracle/net/radius/DefaultRadiusInterface
sqlnet.radius_classpath = /jre1.1/
```

Data Encryption and Integrity Parameters

If you do not specify any values for Server Encryption, Client Encryption, Server Checksum, or Client Checksum, the corresponding configuration parameters do not appear in the sqlnet.ora file. However, Oracle Advanced Security defaults to ACCEPTED.

For both data encryption and integrity algorithms, the server selects the first algorithm listed in its `sqlnet.ora` file that matches an algorithm listed in the client `sqlnet.ora` file, or in the client installed list if the client lists no algorithms in its `sqlnet.ora` file. If there are no entries in the server `sqlnet.ora` file, the server sequentially searches its installed list to match an item on the client side—either in the client `sqlnet.ora` file or in the client installed list. *If no match can be made and one side of the connection REQUIRED the algorithm type (data encryption or integrity), the connection fails.* Otherwise, the connection succeeds with the algorithm type `inactive`.

Data encryption and integrity algorithms are selected independently of each other. Encryption can be activated without integrity, and integrity can be activated without encryption, as shown by Table A–1:

Table A–1   Algorithm Type Selection

Encryption Selected?	Integrity Selected?
Yes	No
Yes	Yes
No	Yes
No	No

There are three classes of parameters used to enable data encryption and integrity. The first two classes listed here are required and the third (seeding the random key generator) is optional:

- Encryption and Integrity Parameters
- Seeding the Random Key Generator (Optional)

> **See Also:**
>
> - Chapter 4, "Configuring Network Data Encryption and Integrity for Oracle Servers and Clients"
> - "About Activating Encryption and Integrity" on page 4-4

Encryption and Integrity Parameters

The following sections describe data encryption and integrity parameters:

- "SQLNET.ENCRYPTION_SERVER Parameter"
- "SQLNET.ENCRYPTION_CLIENT Parameter"
- "SQLNET.CRYPTO_CHECKSUM_SERVER Parameter"
- "SQLNET.CRYPTO_CHECKSUM_CLIENT Parameter"
- "SQLNET.ENCRYPTION_TYPES_SERVER Parameter"
- "SQLNET.ENCRYPTION_TYPES_CLIENT Parameter"
- "SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER Parameter"
- "SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT Parameter"

SQLNET.ENCRYPTION_SERVER Parameter

This parameter specifies the desired encryption behavior when a client or a server acting as a client connects to this server. The behavior of the server partially depends on the `SQLNET.ENCRYPTION_CLIENT` setting at the other end of the connection.

Table A–2   SQLNET.ENCRYPTION_SERVER Parameter Attributes

Attribute	Description
Syntax	SQLNET.ENCRYPTION_SERVER = *valid_value*
Valid Values	ACCEPTED, REJECTED, REQUESTED, REQUIRED
Default Setting	ACCEPTED

SQLNET.ENCRYPTION_CLIENT Parameter

This parameter specifies the desired encryption behavior when this client or server acting as a client connects to a server. The behavior of the client partially depends on the value set for SQLNET.ENCRYPTION_SERVER at the other end of the connection.

Table A–3   SQLNET.ENCRYPTION_CLIENT Parameter Attributes

Attribute	Description
Syntax	SQLNET.ENCRYPTION_CLIENT = *valid_value*
Valid Values	ACCEPTED, REJECTED, REQUESTED, REQUIRED
Default Setting	ACCEPTED

SQLNET.CRYPTO_CHECKSUM_SERVER Parameter

This parameter specifies the desired data integrity behavior when a client or another server acting as a client connects to this server. The behavior partially depends on the SQLNET.CRYPTO_CHECKSUM_CLIENT setting at the other end of the connection.

Table A–4   SQLNET.CRYPTO_CHECKSUM_SERVER Parameter Attributes

Attribute	Description
Syntax	SQLNET.CRYPTO_CHECKSUM_SERVER = *valid_value*
Valid Values	ACCEPTED, REJECTED, REQUESTED, REQUIRED
Default Setting	ACCEPTED

SQLNET.CRYPTO_CHECKSUM_CLIENT Parameter

This parameter specifies the desired data integrity behavior when this client or server acting as a client connects to a server. The behavior partially depends on the SQLNET.CRYPTO_CHECKSUM_SERVER setting at the other end of the connection.

Table A–5   SQLNET.CRYPTO_CHECKSUM_CLIENT Parameter Attributes

Attribute	Description
Syntax	SQLNET.CRYPTO_CHECKSUM_CLIENT = *valid_value*
Valid Values	ACCEPTED, REJECTED, REQUESTED, REQUIRED
Default Setting	ACCEPTED

SQLNET.ENCRYPTION_TYPES_SERVER Parameter

This parameter specifies a list of encryption algorithms used by this server in the order of intended use. This list is used to negotiate a mutually acceptable algorithm with the client end of the connection. Each algorithm is checked against the list of available client algorithm types until a match is found. If an algorithm that is not installed is specified on this side, the connection terminates with the error message ORA-12650.

Table A–6   SQLNET.ENCRYPTION_TYPES_SERVER Parameter Attributes

Attribute	Description
Syntax	`SQLNET.ENCRYPTION_TYPES_SERVER = (`*valid_encryption_ algorithm* `[,`*valid_encryption_algorithm*`])`
Valid Values	■ RC4_256: RSA RC4 (256-bit key size) ■ AES256: AES (256-bit key size) ■ AES192: AES (192-bit key size) ■ 3DES168: 3-key Triple-DES (168-bit effective key size) ■ RC4_128: RSA RC4 (128-bit key size) ■ AES128: AES (128-bit key size) ■ 3DES112: 2-key Triple-DES (112-bit effective key size) ■ RC4_56: RSA RC4 (56-bit key size) ■ DES: Standard DES (56-bit key size) ■ RC4_40: RSA RC4 (40-bit key size) ■ DES40: DES40 (40-bit key size)
Default Setting	If no algorithms are defined in the local `sqlnet.ora` file, all installed algorithms are used in a negotiation in the preceding sequence.

Usage Notes	You can specify multiple encryption algorithms. It can be either a single value or a list of algorithm names. For example, either of the following encryption parameters is acceptable:
	
	`SQLNET.ENCRYPTION_TYPES_SERVER=(RC4_40)`
	
	`SQLNET.ENCRYPTION_TYPES_SERVER=(3DES112,RC4_ 56,RC4_128,3DES168)`

SQLNET.ENCRYPTION_TYPES_CLIENT Parameter

This parameter specifies a list of encryption algorithms used by this client or server acting as a client. This list is used to negotiate a mutually acceptable algorithm with the other end of the connection. If an algorithm that is not installed is specified on this side, the connection terminates with the `ORA-12650` error message.

Table A–7   SQLNET.ENCRYPTION_TYPES_CLIENT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.ENCRYPTION_TYPES_CLIENT = (`*valid_encryption_ algorithm* `[,`*valid_encryption_algorithm*`])`
Valid Values	■ RC4_256: RSA RC4 (256-bit key size). ■ AES256: AES (256-bit key size). ■ AES192: AES (192-bit key size). ■ 3DES168: 3-key Triple-DES (168-bit effective key size). ■ RC4_128: RSA RC4 (128-bit key size). ■ AES128: AES (128-bit key size). ■ 3DES112: 2-key Triple-DES (112-bit effective key size). ■ RC4_56: RSA RC4 (56-bit key size). ■ DES: Standard DES (56-bit key size). ■ RC4_40: RSA RC4 (40-bit key size). ■ DES40: DES40 (40-bit key size).

Table A–7 (Cont.) SQLNET.ENCRYPTION_TYPES_CLIENT Parameter Attributes

Attribute	Description
Default Setting	If no algorithms are defined in the local `sqlnet.ora` file, all installed algorithms are used in a negotiation.
Usage Notes	You can specify multiple encryption algorithms.

SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER Parameter

This parameter specifies a list of data integrity algorithms that this server or client to another server uses, in order of intended use. This list is used to negotiate a mutually acceptable algorithm with the other end of the connection. Each algorithm is checked against the list of available client algorithm types until a match is found. If an algorithm is specified that is not installed on this side, the connection terminates with the `ORA-12650` error message

Table A–8 SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER Parameter Attributes

Attribute	Description
Syntax	`SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER = (`*valid_crypto_checksum_algorithm* `[,`*valid_crypto_checksum_algorithm*`])`
Valid Values	■ SHA-1: Secure Hash Algorithm ■ MD5: Message Digest 5
Default Setting	If no algorithms are defined in the local `sqlnet.ora` file, all installed algorithms are used in a negotiation in the preceding sequence.

SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT Parameter

This parameter specifies a list of data integrity algorithms that this client or server acting as a client uses. This list is used to negotiate a mutually acceptable algorithm with the other end of the connection. If an algorithm that is not installed on this side is specified, the connection terminates with the `ORA-12650` error message.

Table A–9 SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT = (`*valid_crypto_checksum_algorithm* `[,`*valid_crypto_checksum_algorithm*`])`
Valid Values	■ SHA-1: Secure Hash Algorithm ■ MD5: Message Digest 5
Default Setting	If no algorithms are defined in the local `sqlnet.ora` file, all installed algorithms are used in a negotiation.

Seeding the Random Key Generator (Optional)

Setting this parameter in the `sqlnet.ora` file as follows is optional:

```
SQLNET.CRYPTO_SEED = "10-70 random characters"
```

The characters that form the value for this parameter can be used to seed the random number generator that is used to generate cryptographic key material. The more random the characters entered in this field, the stronger the keys are. You set this parameter by entering from 10 to 70 random characters into the preceding statement.

> **Note:** If you use this parameter to seed the random number generator, then Oracle recommends that you enter as many characters as possible, up to 70, to make the resulting key more random and therefore stronger.

If you do not use this parameter, the system uses various sources of random numbers, depending on your operating system, to seed the random number generator.

B

Authentication Parameters

This appendix illustrates some sample configuration files with the profile file (`sqlnet.ora`) and the database initialization file authentication parameters, when using Kerberos, RADIUS, or SSL authentication.

This appendix contains the following topics:

- Parameters for Clients and Servers using Kerberos Authentication
- Parameters for Clients and Servers using RADIUS Authentication
- Parameters for Clients and Servers using SSL

Parameters for Clients and Servers using Kerberos Authentication

Following is a list of parameters to insert into the configuration files for clients and servers using Kerberos.

Table B–1   Kerberos Authentication Parameters

File Name	Configuration Parameters

`sqlnet.ora`	`SQLNET.AUTHENTICATION_SERVICES=(KERBEROS5)`
	`SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=oracle`
	`SQLNET.KERBEROS5_CC_NAME=/usr/tmp/DCE-CC`
	`SQLNET.KERBEROS5_CLOCKSKEW=1200`
	`SQLNET.KERBEROS5_CONF=/krb5/krb.conf`
	`SQLNET.KERBEROS5_CONF_MIT=(FALSE)`
	`SQLNET.KERBEROS5_REALMS=/krb5/krb.realms`
	`SQLNET.KERBEROS5_KEYTAB=/krb5/v5srvtab`

`initialization parameter file`	`REMOTE_OS_AUTHENT=FALSE` `OS_AUTHENT_PREFIX=""`

Parameters for Clients and Servers using RADIUS Authentication

The following sections describe the parameters for RADIUS authentication

- sqlnet.ora File Parameters
- Minimum RADIUS Parameters
- Initialization File Parameters

sqlnet.ora File Parameters

The following sections describe the `sqlnet.ora` parameters that are used to specify RADIUS authentication.

SQLNET.AUTHENTICATION_SERVICES

This parameter configures the client or the server to use the RADIUS adapter. Table B–2 describes this parameter's attributes.

Table B–2   SQLNET.AUTHENTICATION_SERVICES Parameter Attributes

Attribute	Description
Syntax	`SQLNET.AUTHENTICATION_SERVICES=radius`
Default setting	None

SQLNET.RADIUS_AUTHENTICATION

This parameter sets the location of the primary RADIUS server, either host name or dotted decimal format. If the RADIUS server is on a different computer from the Oracle server, you must specify either the host name or the IP address of that computer. Table B–3 describes this parameter's attributes.

Table B–3   SQLNET.RADIUS_AUTHENTICATION Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_AUTHENTICATION=RADIUS_server_IP_address`
Default setting	`localhost`

SQLNET.RADIUS_AUTHENTICATION_PORT

This parameter sets the listening port of the primary RADIUS server. Table B–4 describes this parameter's attributes.

Table B–4   SQLNET.RADIUS_AUTHENTICATION_PORT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_AUTHENTICATION_PORT=port_number`
Default setting	`1645`

SQLNET.RADIUS_AUTHENTICATION_TIMEOUT

This parameter sets the time to wait for response. Table B–5 describes this parameter's attributes.

Table B–5   SQLNET.RADIUS_AUTHENTICATION_TIMEOUT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_AUTHENTICATION_TIMEOUT=time_in_seconds`
Default setting	`5`

SQLNET.RADIUS_AUTHENTICATION_RETRIES

This parameter sets the number of times to resend authentication information. Table B–6 describes this parameter's attributes.

Table B–6   SQLNET.RADIUS_AUTHENTICATION_RETRIES Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_AUTHENTICATION_RETRIES=n_times_to_resend`

Table B–6   (Cont.) SQLNET.RADIUS_AUTHENTICATION_RETRIES Parameter Attributes

Attribute	Description
Default setting	3

SQLNET.RADIUS_SEND_ACCOUNTING

This parameter turns accounting on and off. If you enable accounting, packets will be sent to the active RADIUS server at the listening port plus one. By default, packets are sent to port 1646. You need to turn this feature on only when your RADIUS server supports accounting and you want to keep track of the number of times the user is logging on to the system. Table B–7 describes this parameter's attributes.

Table B–7   SQLNET.RADIUS_SEND_ACCOUNTING Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_SEND_ACCOUNTING=on`
Default setting	`off`

SQLNET.RADIUS_SECRET

This parameter specifies the file name and location of the RADIUS secret key. Table B–8 describes this parameter's attributes.

Table B–8   SQLNET.RADIUS_SECRET Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_SECRET=path_to_RADIUS_secret_key`
Default setting	`$ORACLE_HOME/network/security/radius.key`

SQLNET.RADIUS_ALTERNATE

This parameter sets the location of an alternate RADIUS server to be used in case the primary server becomes unavailable for fault tolerance. Table B–9 describes this parameter's attributes.

Table B–9   SQLNET.RADIUS_ALTERNATE Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_ALTERNATE=alternate_RADIUS_server_` `hostname_or_IP_address`
Default setting	`off`

SQLNET.RADIUS_ALTERNATE_PORT

This parameter sets the listening port for the alternate RADIUS server. Table B–10 describes this parameter's attributes.

Table B–10   SQLNET.RADIUS_ALTERNATE_PORT Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_ALTERNATE_PORT=alternate_RADIUS_` `server_listening_port_number`
Default setting	`1645`

SQLNET.RADIUS_ALTERNATE_TIMEOUT

This parameter sets the time to wait for response for the alternate RADIUS server. Table B–11 describes this parameter's attributes.

Table B–11   SQLNET.RADIUS_ALTERNATE_TIMEOUT Parameter Attributes

Attribute	Description
Syntax	SQLNET.RADIUS_ALTERNATE_TIMEOUT=*time_in_seconds*
Default setting	5

SQLNET.RADIUS_ALTERNATE_RETRIES

This parameter sets the number of times that the alternate RADIUS server resends messages. Table B–12 describes this parameter's attributes.

Table B–12   SQLNET.RADIUS_ALTERNATE_RETRIES Parameter Attributes

Attribute	Description
Syntax	SQLNET.RADIUS_ALTERNATE_RETRIES=*n_times_to_resend*
Default setting	3

SQLNET.RADIUS_CHALLENGE_RESPONSE

This parameter turns on or turns off the challenge-response or asynchronous mode support. Table B–13 describes this parameter's attributes.

Table B–13   SQLNET.RADIUS_CHALLENGE_RESPONSE Parameter Attributes

Attribute	Description
Syntax	SQLNET.RADIUS_CHALLENGE_RESPONSE=*on*
Default setting	off

SQLNET.RADIUS_CHALLENGE_KEYWORD

This parameter sets the keyword to request a challenge from the RADIUS server. User types no password on the client. Table B–14 describes this parameter's attributes.

Table B–14   SQLNET.RADIUS_CHALLENGE_KEYWORD Parameter Attributes

Attribute	Description
Syntax	SQLNET.RADIUS_CHALLENGE_KEYWORD=*keyword*
Default setting	challenge

SQLNET.RADIUS_AUTHENTICATION_INTERFACE

This parameter sets the name of the Java class that contains the graphical user interface when RADIUS is in the challenge-response (asynchronous) mode. Table B–15 describes this parameter's attributes.

Table B–15   SQLNET.RADIUS_AUTHENTICATION_INTERFACE Parameter Attributes

Attribute	Description
Syntax	SQLNET.RADIUS_AUTHENTICATION_INTERFACE=*Java_class_name*

Table B–15 (Cont.) SQLNET.RADIUS_AUTHENTICATION_INTERFACE Parameter

Attribute	Description
Default setting	`DefaultRadiusInterface` `(oracle/net/radius/DefaultRadiusInterface)`

SQLNET.RADIUS_CLASSPATH

If you decide to use the challenge-response authentication mode, RADIUS presents the user with a Java-based graphical interface requesting first a password, then additional information, for example, a dynamic password that the user obtains from a token card. Add the `SQLNET.RADIUS_CLASSPATH` parameter in the `sqlnet.ora` file to set the path for the Java classes for that graphical interface, and to set the path to the JDK Java libraries. Table B–16 describes this parameter's attributes.

Table B–16 SQLNET.RADIUS_CLASSPATH Parameter Attributes

Attribute	Description
Syntax	`SQLNET.RADIUS_CLASSPATH=`*path_to_GUI_Java_classes*
Default setting	`$ORACLE_HOME/jlib/netradius.jar:$ORACLE_` `HOME/JRE/lib/sparc/native_threads`

Minimum RADIUS Parameters

```
sqlnet.authentication_services = (radius)
sqlnet.radius.authentication = IP-address-of-RADIUS-server
sqlnet.radius_challenge_response = ON
```

Initialization File Parameters

```
REMOTE_OS_AUTHENT=FALSE
OS_AUTHENT_PREFIX=""
```

Parameters for Clients and Servers using SSL

There are two ways to configure a parameter:

- Static: The name of the parameter that exists in the `sqlnet.ora` file.

- Dynamic: The name of the parameter used in the security subsection of the Oracle Net address.

SSL Authentication Parameters

This section describes the static and dynamic parameters for configuring SSL on the server.

Attribute	Description
Parameter Name (static)	SQLNET.AUTHENTICATION_SERVICES
Parameter Name (dynamic)	AUTHENTICATION
Parameter Type	String LIST
Parameter Class	Static
Permitted Values	Add TCPS to the list of available authentication services.

Attribute	Description
Default Value	No default value.
Description	To control which authentication services a user wants to use.
	Note: The dynamic version supports only the setting of one type.
Existing/New Parameter	Existing
Syntax (static)	SQLNET.AUTHENTICATION_SERVICES = (TCPS, *selected_method_1*, *selected_method_2*)
Example (static)	SQLNET.AUTHENTICATION_SERVICES = (TCPS, radius)
Syntax (dynamic)	AUTHENTICATION = *string*
Example (dynamic)	
	AUTHENTICATION = (TCPS)

Cipher Suite Parameters

This section describes the static and dynamic parameters for configuring cipher suites.

Attribute	Description
Parameter Name (static)	SSL_CIPHER_SUITES
Parameter Name (dynamic)	SSL_CIPHER_SUITES
Parameter Type	String LIST
Parameter Class	Static
Permitted Values	Any known SSL cipher suite
Default Value	No default
Description	Controls the combination of encryption and data integrity used by SSL.
Existing/New Parameter	Existing
Syntax (static)	SSL_CIPHER_SUITES=(*SSL_cipher_suite1*[, *SSL_cipher_suite2*, ... *SSL_cipher_suiteN*])
Example (static)	SSL_CIPHER_SUITES=(SSL_DH_DSS_WITH_DES_CBC_SHA)
Syntax (dynamic)	SSL_CIPHER_SUITES=(*SSL_cipher_suite1*
	[, *SSL_cipher_suite2*, ...*SSL_cipher_suiteN*])
Example (dynamic)	SSL_CIPHER_SUITES=(SSL_DH_DSS_WITH_DES_CBC_SHA)

Supported SSL Cipher Suites

Oracle Advanced Security supports the following cipher suites:

- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_DES_CBC_SHA

- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA

- SSL_DH_anon_WITH_RC4_128_MD5

- SSL_DH_anon_WITH_DES_CBC_SHA

- SSL_RSA_EXPORT_WITH_RC4_40_MD5

- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA

- SSL_RSA_WITH_AES_128_CBC_SHA

- SSL_RSA_WITH_AES_256_CBC_SHA

Note that the cipher suites that use Advanced Encryption Standard (AES) work with Transport Layer Security (TLS 1.0) only.

SSL Version Parameters

This section describes the static and dynamic parameters for configuring the version of SSL to be used.

Attribute	Description
Parameter Name (static)	SSL_VERSION
Parameter Name (dynamic)	SSL_VERSION
Parameter Type	string
Parameter Class	Static
Permitted Values	Any version which is valid to SSL. (0, 3.0)
Default Value	"0"
Description	To force the version of the SSL connection.
Existing/New Parameter	New
Syntax (static)	SSL_VERSION=*version*
Example (static)	SSL_VERSION=3.0
Syntax (dynamic)	SSL_VERSION=*version*
Example (dynamic)	SSL_VERSION=3.0

SSL Client Authentication Parameters

This section describes the static and dynamic parameters for configuring SSL on the client.

Attribute	Description
Parameter Name (static)	SSL_CLIENT_AUTHENTICATION
Parameter Name (dynamic)	SSL_CLIENT_AUTHENTICATION
Parameter Type	Boolean
Parameter Class	Static
Permitted Values	TRUE/FALSE

Attribute	Description
Default Value	TRUE
Description	To control whether a client, in addition to the server, is authenticated using SSL.
Existing/New Parameter	New
Syntax (static)	SSL_CLIENT_AUTHENTICATION={*TRUE*	*FALSE*}
Example (static)	SSL_CLIENT_AUTHENTICATION=FALSE
Syntax (dynamic)	SSL_CLIENT_AUTHENTICATION={*TRUE*	*FALSE*}
Example (dynamic)	SSL_CLIENT_AUTHENTICATION=FALSE

SSL X.509 Server Match Parameters

This section describes the parameters that are used to validate the identity of a server that the client connects to.

SSL_SERVER_DN_MATCH

Attribute	Description
Parameter Name	SSL_SERVER_DN_MATCH
Where stored	`sqlnet.ora`
Purpose	Use this parameter to force the server's **distinguished name (DN)** to match its service name. If you force the match verifications, SSL ensures that the certificate is from the server. If you choose not to enforce the match verification, SSL performs the check but permits the connection, regardless of whether there is a match. *Not forcing the match lets the server potentially fake its identity.*
Values	`yes|on|true`. Specify to enforce a match. If the DN matches the service name, the connection succeeds; otherwise, the connection fails.
	`no|off|false`. Specify to not enforce a match. If the DN does not match the service name, the connection is successful, but an error is logged to the `sqlnet.log` file.
Default	Oracle8*i*, or later:.FALSE. SSL client (always) checks server DN. If it does not match the service name, the connection succeeds but an error is logged to `sqlnet.log` file.
Usage Notes	Additionally configure the `tnsnames.ora` parameter `SSL_SERVER_CERT_DN` to enable server DN matching.

SSL_SERVER_CERT_DN

Attribute	Description
Parameter Name	SSL_SERVER_CERT_DN
Where stored	`tnsnames.ora`. It can be stored on the client, for every server it connects to, or it can be stored in the LDAP directory, for every server it connects to, updated centrally.
Purpose	This parameter specifies the **distinguished name (DN)** of the server. The client uses this information to obtain the list of DNs it expects for each of the servers to force the server's DN to match its service name.

Attribute	Description
Values	Set equal to **distinguished name (DN)** of the server.
Default	n/a
Usage Notes	Additionally configure the `sqlnet.ora` parameter `SSL_SERVER_DN_MATCH` to enable server DN matching.
Example	`dbalias=(description=address_list=(address=(protocol=tcps)(host=hostname)(port=portnum)))(connect_data=(sid=Finance))(security=(SSL_SERVER_CERT_DN="CN=Finance,CN=OracleContext,C=US,O=Acme"))`

Wallet Location

For any application that must access a wallet for loading the security credentials into the process space, you must specify the wallet location parameters defined by Table B–17 in each of the following configuration files:

- `sqlnet.ora`
- `listener.ora`

Table B–17   Wallet Location Parameters

Static Configuration	Dynamic Configuration

`WALLET_LOCATION =`
`(SOURCE=`
`  (METHOD=File)`
`  (METHOD_DATA=`
`     (DIRECTORY=your wallet location)`
`      )`
`  )`
	`MY_WALLET_DIRECTORY`
	`= your_wallet_dir`

The default wallet location is the `ORACLE_HOME` directory.

C

Integrating Authentication Devices Using RADIUS

This appendix describes how third-party authentication vendors customize the RADIUS challenge-response user interface to fit their particular device.

This appendix contains the following topics:

- About the RADIUS Challenge-Response User Interface
- Customizing the RADIUS Challenge-Response User Interface

> **See Also:** Chapter 6, "Configuring RADIUS Authentication"

About the RADIUS Challenge-Response User Interface

You can set up any authentication device that supports the RADIUS standard to authenticate Oracle users. When your authentication device uses the challenge-response mode, a graphical interface prompts the user first for a password and then for additional information. For example, a dynamic password that the user obtains from a token card. This interface is Java-based to provide optimal platform independence.

Third party vendors of authentication devices must customize this graphical user interface to fit their particular device. For example, a smart card vendor customizes the Oracle client to issue the challenge to the smart card reader. Then, when the smart card receives a challenge, it responds by prompting the user for more information, such as a PIN.

Customizing the RADIUS Challenge-Response User Interface

You can customize this interface by creating your own class to support the functionality described in Table C–1. You can then open the `sqlnet.ora` file, look up the `SQLNET.RADIUS_AUTHENTICATION_INTERFACE` parameter, and replace the name of the class listed there (`DefaultRadiusInterface`), with the name of the new class you have just created. When you make this change in the `sqlnet.ora` file, the class is loaded on the Oracle client in order to handle the authentication process.

The third party must implement the Oracle RADIUS Interface, which is located in the `ORACLE.NET.RADIUS` package.

```
public interface OracleRadiusInterface {
  public void radiusRequest();
  public void radiusChallenge(String challenge);
  public String getUserName();
  public String getPassword();
```

```
}
```

Table C–1   Server Encryption Level Setting

Parameter	Description
radiusRequest	Generally, this prompts the user for a user name and password, which will later be retrieved through `getUserName` and `getPassword`.
getUserName	Extracts the user name the user enters. If this method returns an empty string, it is assumed that the user wants to cancel the operation. The user then receives a message indicating that the authentication attempt failed.
getPassword	Extracts the password the user enters. If `getUserName` returns a valid string, but `getPassword` returns an empty string, the challenge keyword is replaced as the password by the database. If the user enters a valid password, a challenge may or may not be returned by the RADIUS server.
radiusChallenge	Presents a request sent from the RADIUS server for the user to respond to the server's challenge.
getResponse	Extracts the response the user enters. If this method returns a valid response, that information then populates the `User-Password` attribute in the new `Access-Request` packet. If an empty string is returned, the operation is aborted from both sides by returning the corresponding value.

D

Oracle Advanced Security FIPS 140-1 Settings

Oracle Advanced Security Release 8.1.6 has been validated under **Federal Information Processing Standard (FIPS)** 140-1 at the Level 2 security level. This appendix describes the formal configuration required for Oracle Advanced Security to comply with the FIPS 140-1 standard. Refer to the NIST Cryptographic Modules Validation list at the following Web site address:

http://csrc.nist.gov/cryptval/140-1/1401val.htm

This appendix contains the following topics:

- Configuration Parameters

- Post Installation Checks

- Status Information

- Physical Security

> **Note:** The information contained in this appendix should be used with the information provided in Appendix A, "Data Encryption and Integrity Parameters".

Configuration Parameters

This appendix contains information on the Oracle Advanced Security parameters required in the `sqlnet.ora` files to ensure that any connections created between a client and server are encrypted under the control of the server.

Configuration parameters are contained in the `sqlnet.ora` file that is held locally for each of the client and server processes. The protection placed on these files should be equivalent to the level of a DBA.

The following configuration parameters are described in this appendix:

- `ENCRYPTION_SERVER`

- `ENCRYPTION_CLIENT`

- `ENCRYPTION_TYPES_SERVER`

- `CRYPTO_SEED`

- `CRYPTO_SEED_CLIENT`

- `FIPS_140`

Server Encryption Level Setting

The server side of the negotiation notionally controls the connection settings. The following parameter in the server file is mandatory:

```
SQLNET.ENCRYPTION_SERVER=REQUIRED
```

Setting the encryption as REQUIRED on the server side of the connection ensures that a connection is only permitted if encryption is used, irrespective of the parameter value on the client.

Client Encryption Level Setting

The ENCRYPTION_CLIENT parameter specifies the connection behavior for the client. One of the following parameter settings in the client file is mandatory:

```
SQLNET.ENCRYPTION_CLIENT=(ACCEPTED|REQUESTED|REQUIRED)
```

A connection to the server is only possible if there is agreement between client and server for the connection encryption. The server has this set to REQUIRED, therefore the client must not reject encryption for a valid connection to be the result. Failure to specify one of these values results in error when attempting to connect to a FIPS 140-1 compliant server.

Server Encryption Selection List

The ENCRYPTION_TYPES_SERVER parameter specifies a list of encryption algorithms that the server is permitted to use when acting as a server in the order of required usage. The specified algorithm must be installed or the connection terminates. For FIPS 140-1 compliance, only DES encryption is permitted and therefore the following parameter setting is mandatory:

```
SQLNET.ENCRYPTION_TYPES_SERVER=(DES|DES40)
```

Client Encryption Selection List

The ENCRYPTION_TYPES_CLIENT parameter specifies the list of encryption algorithms which the client is prepared to use for the connection with the server. In order for a connection to be successful, the algorithm must first be installed and the encryption type must be mutually acceptable to the server.

To create a connection with a server that is configured for FIPS 140-1, the following parameter setting is mandatory:

```
SQLNET.ENCRYPTION_TYPES_CLIENT=(DES|DES40)
```

Cryptographic Seed Value

The CRYPTO_SEED parameter contains characters which are part of the seed for the random number generator. There are no explicit requirements to set this parameter to satisfy the FIPS 140-1 standard. If you would like to use this parameter, then you can set it using10 to 70 random characters to improve the quality of one of the seeds. Note that the random number generator produces higher quality output when you use a large number of random characters with this parameter.

To set this parameter, use the following syntax:

```
SQLNET.CRYPTO_SEED=10_to_70_random_characters
```

FIPS Parameter

The default setting of the `FIPS_140` parameter is `FALSE`. Setting the parameter to `TRUE` is mandatory for both client and server to ensure Oracle Advanced Security complies with the standards defined in FIPS 140-1 as follows:

```
SQLNET.FIPS_140=TRUE
```

> **Note:** Use a text editor to set the `FIPS_140` parameter in the `sqlnet.ora` file. You cannot use Oracle Net Manager to set this parameter.

Post Installation Checks

After the installation, the following permissions must be verified in the operating system:

- Execute permissions must be set on all Oracle Advanced Security executable files so as to prevent execution of Oracle Advanced Security by users who are unauthorized to do so in accordance with the system security policy.

- Read and write permissions must be set on all executable files so as to prevent accidental or deliberate reading or modification of Oracle Advanced Security files by any user.

To comply with FIPS 140-1 Level 2 requirements, the security policy must include procedures to prevent unauthorized users from reading or modifying Oracle Advanced Security processes and the memory they are using in the operating system.

Status Information

Status information for Oracle Advanced Security is available after the connection has been established. The information is contained in the RDBMS virtual table `v$session_connect_info`.

Running the query `SELECT * from V$SESSION_CONNECT_INFO` displays all of the product banner information for the active connection. Table D–1 shows an example of a connection configuration where both DES encryption and MD5 data integrity is defined:

Table D–1   Sample Output from v$session_connect_info

SID	AUTHENTICATION	OSUSER	NETWORK_SERVICE_BANNER
7	DATABASE	oracle	Oracle Bequeath operating system adapter for Solaris, v8.1.6.0.0
7	DATABASE	oracle	Oracle Advanced Security: encryption service for Solaris
7	DATABASE	oracle	Oracle Advanced Security: DES encryption service adapter
7	DATABASE	oracle	Oracle Advanced Security: crypto-checksumming service
7	DATABASE	oracle	Oracle Advanced Security: MD5 crypto-checksumming service adapter.

Physical Security

To comply with FIPS 140-1 Level 2 requirements, tamper-evident seals must be applied to the cover of each computer to ensure that removal of the cover is detectable.

E

Oracle Advanced Security FIPS 140-2 Settings

The cryptographic libraries for SSL included in Oracle Database 10g are designed to meet FIPS 140-2 Level 2 certification. Oracle Advanced Security makes use of these cryptographic libraries for SSL authentication. Please verify the current status of the certification at the Cryptographic Modules Validation Program Web site address:

http://csrc.nist.gov/cryptval/

The security policy, which would be available at the NIST site upon successful certification, includes requirements for secure configuration of the host operating system.

The following topics are covered in this appendix:

- Configuring FIPS Parameter
- Selecting Cipher Suites
- Post-Installation Checks
- Verifying FIPS Connections

Configuring FIPS Parameter

Oracle Advanced Security SSL adapter can be configured to run in FIPS mode by setting the `SQLNET.SSLFIPS_140` parameter to `TRUE` in the `sqlnet.ora` file.

```
SQLNET.SSLFIPS_140=TRUE
```

This parameter is set to `FALSE` by default. It must be set to `TRUE` on both the client and the server for FIPS mode operation.

Make sure that the `sqlnet.ora` file is either located in the `ORACLE_HOME/network/admin` directory, or is pointed to by the `TNS_ADMIN` environment variable. This procedure can be repeated in any Oracle home for any database server or client.

> **Note:** You must add or edit the `SQLNET.SSLFIPS_140` parameter in the `sqlnet.ora` file with a text editor. You cannot use Oracle Net Manager to set this parameter

Selecting Cipher Suites

A cipher suite is a set of authentication, encryption and data integrity algorithms used for exchanging messages between network nodes. During an SSL handshake, for example, the two nodes negotiate to see as to which cipher suite they will use when transmitting messages back and forth.

Only the cipher suites listed below are approved for FIPS validation:

- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA

- SSL_DH_anon_WITH_DES_CBC_SHA

- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

- SSL_RSA_WITH_3DES_EDE_CBC_SHA

- SSL_RSA_WITH_DES_CBC_SHA

- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA

Oracle Advanced Security SSL cipher suites are automatically set to FIPS approved cipher suites. If you wish to configure specific cipher suites, you can do so by editing the SSL_CIPHER_SUITES parameter in the sqlnet.ora file.

```
SSL_CIPHER_SUITES=(SSL_cipher_suite1[,SSL_cipher_suite2[,..]])
```

You can also use Oracle Net Manager to set this parameter on the server and the client.

> **See Also:** "Step 3: Set the SSL Cipher Suites on the Server (Optional)" on page 8-11 and "Step 4: Set the Client SSL Cipher Suites (Optional)" on page 8-18 for more information on setting cipher suites.

Post-Installation Checks

After installation, the following permissions must be verified in the operating system:

- Execute permissions must be set on all Oracle executable files so as to prevent execution of Oracle Cryptographic Libraries by users who are unauthorized to do so in accordance with the system security policy.

- Read and write permissions must be set on all Oracle executable files so as to prevent accidental or deliberate reading or modification of Oracle Cryptographic Libraries by any user.

To comply with FIPS 140-2 Level 2 requirements, the security policy must include procedures to prevent unauthorized users from reading, modifying or executing Oracle Cryptographic Libraries processes and the memory they are using in the operating system.

Verifying FIPS Connections

To check if FIPS mode is enabled, tracing can be added to the sqlnet.ora file. FIPS self-test messages can be found in the trace file. Add the following lines to sqlnet.ora to enable tracing:

```
trace_directory_server=trace_dir
trace_file_server=trace_file
trace_level_server=trace_level
```

For example:

```
trace_directory=/private/oracle/owm
```

```
trace_file_server=fips_trace.trc
trace_level_server=6
```

Trace level 6 is the minimum trace level required to check the results of the FIPS self-tests.

F

orapki Utility

The `orapki` utility is provided to manage public key infrastructure (PKI) elements, such as wallets and certificate revocation lists, on the command line so the tasks it performs can be incorporated into scripts. Providing a way to incorporate the management of PKI elements into scripts makes it possible to automate many of the routine tasks of maintaining a PKI.

The following topics are included in this appendix:

- orapki Utility Overview
- Creating Signed Certificates for Testing Purposes
- Managing Oracle Wallets with orapki Utility
- Managing Certificate Revocation Lists (CRLs) with orapki Utility
- orapki Usage Examples
- orapki Utility Commands Summary

orapki Utility Overview

This command line utility can be used to perform the following tasks:

- Creating and viewing signed certificates for testing purposes
- Manage Oracle wallets:
  - Create and display Oracle wallets
  - Add and remove certificate requests
  - Add and remove certificates
  - Add and remove trusted certificates
- Manage certificate revocation lists (CRLs):
  - Renaming CRLs with a hash value for certificate validation
  - Uploading, listing, viewing, and deleting CRLs in Oracle Internet Directory

orapki Utility Syntax

The basic syntax of the `orapki` command line utility is as follows:

```
orapki module command -parameter value
```

where *module* can be `wallet` (Oracle wallet), `crl` (certificate revocation list), or `cert` (PKI digital certificate). The available commands depend on the `module` you are

using. For example, if you are working with a `wallet`, then you can add a certificate or a key to the wallet with the `add` command. The following example adds the user certificate located at `/private/lhale/cert.txt` to the wallet located at *$ORACLE_HOME*/wallet/ewallet.p12:

```
orapki wallet add -wallet $ORACLE_HOME/wallet/ewallet.p12 -user_cert -cert
/private/lhale/cert.txt
```

Creating Signed Certificates for Testing Purposes

This command line utility provides a convenient, lightweight way to create signed certificates for testing purposes. The following syntax can be used to create signed certificates and to view certificates:

To create a signed certificate for testing purposes:

```
orapki cert create [-wallet wallet_location] -request certificate_request_location
-cert certificate_location -validity number_of_days [-summary]
```

This command creates a signed certificate from the certificate request. The `-wallet` parameter specifies the wallet containing the user certificate and private key that will be used to sign the certificate request. The `-validity` parameter specifies the number of days, starting from the current date, that this certificate will be valid. Specifying a certificate and certificate request is mandatory for this command.

To view a certificate:

```
orapki cert display -cert certificate_location [-summary | -complete]
```

This command enables you to view a test certificate that you have created with `orapki`. You can choose either `-summary` or `-complete`, which determines how much detail the command will display. If you choose `-summary`, the command will display the certificate and its expiration date. If you choose `-complete`, it will display additional certificate information, including the serial number and public key.

Managing Oracle Wallets with orapki Utility

The following sections describe the syntax used to create and manage Oracle wallets with the `orapki` command line utility. You can use these `orapki` utility `wallet` module commands in scripts to automate the wallet creation process.

- Creating and Viewing Oracle Wallets with orapki
- Adding Certificates and Certificate Requests to Oracle Wallets with orapki
- Exporting Certificates and Certificate Requests from Oracle Wallets with orapki

> **Note:** The `-wallet` parameter is mandatory for all `wallet` module commands.

Creating and Viewing Oracle Wallets with orapki

To create an Oracle wallet:

```
orapki wallet create -wallet wallet_location [-pwd password]
```

This command will prompt you to enter and reenter a wallet password, if no password has been specified on the command line. It creates a wallet in the location specified for -wallet.

To create an Oracle wallet with auto login enabled:

```
orapki wallet create -wallet wallet_location -auto_login [-pwd password]
```

This command creates a wallet with auto login enabled, or it can also be used to enable auto login on an existing wallet. If the wallet_location already contains a wallet, then auto login will be enabled for it. To turn the auto login feature off, use Oracle Wallet Manager.

> **See Also:** "Using Auto Login" on page 9-14 for more information

---

> **Note:** For wallets with the auto login feature enabled, you are prompted for a password only for operations that modify the wallet, such as add.

To view an Oracle wallet:

```
orapki wallet display -wallet wallet_location
```

Displays the certificate requests, user certificates, and trusted certificates contained in the wallet, which must be a binary PKCS12 file, with extension .p12. Other files will fail.

Adding Certificates and Certificate Requests to Oracle Wallets with orapki

Following sections describe how to add certificates and certificate requests to Oracle Wallets with orapki utility.

To add a certificate request to an Oracle wallet:

```
orapki wallet add -wallet wallet_location -dn user_dn -keySize 512|1024|2048
```

This command adds a certificate request to a wallet for the user with the specified distinguished name (user_dn). The request also specifies the requested certificate's key size (512, 1024, or 2048 bits). To sign the request, export it with the export option.

> **See Also:** "Exporting Certificates and Certificate Requests from Oracle Wallets with orapki" on page F-4 for more information

To add a trusted certificate to an Oracle wallet:

```
orapki wallet add -wallet wallet_location -trusted_cert -cert
certificate_location
```

This command adds a trusted certificate, at the specified location (-cert certificate_location), to a wallet. You must add all trusted certificates in the certificate chain of a user certificate before adding a user certificate, or the command to add the user certificate will fail.

To add a root certificate to an Oracle wallet

```
orapki wallet add -wallet wallet_location -dn certificate_dn -keySize
512|1024|2048 -self_signed -validity number_of_days
```

This command creates a new self-signed (root) certificate and adds it to the wallet. The `-validity` parameter (mandatory) specifies the number of days, starting from the current date, that this certificate will be valid. You can specify a key size for this root certificate (`-keySize`) of 512, 1024, or 2048 bits.

To add a user certificate to an Oracle wallet:

```
orapki wallet add -wallet wallet_location -user_cert -cert certificate_location
```

This command adds the user certificate at the location specified with the `-cert` parameter to the Oracle wallet at the `wallet_location`. Before you add a user certificate to a wallet, you must add all the trusted certificates that make up the certificate chain. If all trusted certificates are not installed in the wallet before you add the user certificate, then adding the user certificate will fail.

To add PKCS#11 information to a wallet

You can use a wallet containing PKCS#11 information like any Oracle wallet. The private keys are stored on a hardware device. The cryptographic operations are also performed on the device.

Use the following command to add PKCS#11 information to a wallet:

```
orapki wallet p11_add -wallet wallet_location -p11_lib pkcs11Lib
[-p11_tokenlabel tokenLabel] [-p11_tokenpw tokenPassphrase]
[-p11_certlabel certLabel] [-pwd password]
```

The parameters have the following meaning:

- `-wallet` specifies the wallet location.

- `-p11_lib` specifies the path to the PKCS#11 library. This includes the library filename.

- `-p11_tokenlabel` specifies the token or smart card used on the device. Use this when there are multiple tokens on the device. Token labels are set using vendor tools.

- `-p11_tokenpw` specifies the password that is used to access the token. Token passwords are set using vendor tools.

- `-p11_certlabel` is used to specify a certificate label on the token. Use this when a token contains multiple certificates. Certificate labels are set using vendor tools.

- `-pwd` is used to specify the wallet password.

You can verify credentials on the hardware device using the PKCS#11 wallet. Use the following command for this purpose:

```
orapki wallet p11_verify -wallet wallet_location [-pwd password]
```

Exporting Certificates and Certificate Requests from Oracle Wallets with orapki

Following steps describe how to export certificates and certificate requests from Oracle Wallets with orapki utility.

To export a certificate from an Oracle wallet:

```
orapki wallet export -wallet wallet_location -dn certificate_dn -cert certificate_
filename
```

This command exports a certificate with the subject's distinguished name (`-dn`) from a wallet to a file that is specified by `-cert`.

To export a certificate request from an Oracle wallet:

```
orapki wallet export -wallet wallet_location -dn certificate_request_dn -request
certificate_request_filename
```

This command exports a certificate request with the subject's distinguished name
(-dn) from a wallet to a file that is specified by -request.

Managing Certificate Revocation Lists (CRLs) with orapki Utility

CRLs must be managed with orapki. This utility creates a hashed value of the CRL
issuer's name to identify the CRLs location in your system. If you do not use orapki,
your Oracle server cannot locate CRLs to validate PKI digital certificates. For detailed
information about using orapki to manage CRLs refer to "Certificate Revocation List
Management" on page 8-27.

orapki Usage Examples

This section includes examples of some of the orapki commands discussed in the
preceding sections.

Example F–1 illustrates the steps to create a wallet with a self-signed certificate and
export the certificate to a file.

Example F–1   Create a Wallet with a Self-Signed Certificate and Export the Certificate

The following steps illustrate creating a wallet, adding a self-signed certificate to it,
viewing the wallet and exporting the certificate:

1. Create a wallet

   ```
   orapki wallet create -wallet /private/user/orapki_use/root
   ```

   The wallet is created at the location, /private/user/orapki_use/root.

2. Add a self-signed certificate to the wallet

   ```
   orapki wallet add -wallet /private/user/orapki_use/root -dn
   'CN=root_test,C=US' -keysize 2048 -self_signed -validity 3650
   ```

   This creates a self-signed certificate with a validity of 3650 days. The distinguished
   name of the subject is CN=root_test,C=US. The key size for the certificate is
   2048 bits.

3. View the wallet

   ```
   orapki wallet display -wallet /private/user/orapki_use/root
   ```

   This is used to view the certificate contained in the wallet.

4. Export the certificate

   ```
   orapki wallet export -wallet /private/user/orapki_use/root -dn
   'CN=root_test,C=US' -cert /private/user/orapki_use/root/b64certificate.txt
   ```

   This exports the self-signed certificate to the file, b64certificate.txt. Note
   that the distinguished name used is the same as in step 2.

Example F–2 illustrates miscellaneous tasks related to creating user certificates.

Example F–2   Create a Wallet and a User Certificate

The following steps illustrate creating a wallet, creating a certificate request, exporting the certificate request, creating a signed certificate from the request for testing, viewing the certificate, adding a trusted certificate to the wallet and adding a user certificate to the wallet.

1. Create a wallet with auto login enabled

   ```
   orapki wallet create -wallet /private/user/orapki_use/server -auto_login
   ```

   This creates a wallet at `/private/user/orapki_use/server` with auto login enabled.

2. Add a certificate request to the wallet

   ```
   orapki wallet add -wallet /private/user/orapki_use/server -dn 'CN=server_
   test,C=US' -keysize 2048
   ```

   This adds a certificate request to the wallet that was created. The distinguished name of the subject is `CN=server_test,C=US`. The key size specified is 2048 bits.

3. Export the certificate request to a file

   ```
   orapki wallet export -wallet /private/user/orapki_use/server -dn 'CN=server_
   test,C=US' -request /private/user/orapki_use/server/creq.txt
   ```

   This exports the certificate request to the specified file, which is `creq.txt` in this case.

4. Create a signed certificate from the request for test purposes

   ```
   orapki cert create -wallet /private/user/orapki_use/server -request
   /private/user/orapki_use/server/creq.txt -cert /private/user/orapki_
   use/server/cert.txt -validity 3650
   ```

   This creates a certificate, `cert.txt` with a validity of 3650 days. The certificate is created from the certificate request generated in the preceding step.

5. View the certificate

   ```
   orapki cert display -cert /private/user/orapki_use/server/cert.txt -complete
   ```

   This displays the the certificate generated in the preceding step. The `-complete` option allows you to display additional certificate information, including the serial number and public key.

6. Add a trusted certificate to the wallet

   ```
   orapki wallet add -wallet /private/user/orapki_use/server -trusted_cert -cert
   /private/user/orapki_use/root/b64certificate.txt
   ```

   This adds a trusted certificate, `b64certificate.txt` to the wallet. You must add all trusted certificates in the certificate chain of a user certificate before adding a user certificate.

7. Add a user certificate to the wallet

   ```
   orapki wallet add -wallet /private/user/orapki_use/server -user_cert -cert
   /private/user/orapki_use/server/cert.txt
   ```

   This adds the user certificate, `cert.txt` to the wallet.

orapki Utility Commands Summary

This section lists and describes the following `orapki` commands:

- orapki cert create
- orapki cert display
- orapki crl delete
- orapki crl display
- orapki crl hash
- orapki crl list
- orapki crl upload
- orapki wallet add
- orapki wallet create
- orapki wallet display
- orapki wallet export

orapki cert create

The following sections describe this command.

Purpose

Use the `orapki cert create` command to create a signed certificate for testing purposes.

Syntax

```
orapki cert create [-wallet wallet_location] -request certificate_request_location
-cert certificate_location -validity number_of_days [-summary]
```

- The `-wallet` parameter specifies the wallet containing the user certificate and private key that will be used to sign the certificate request.
- The `-request` parameter (mandatory) specifies the location of the certificate request for the certificate you are creating.
- The `-cert` parameter (mandatory) specifies the directory location where the tool places the new signed certificate.
- The `-validity` parameter (mandatory) specifies the number of days, starting from the current date, that this certificate will be valid.

orapki cert display

The following sections describe this command.

Purpose

Use the `orapki cert display` command to display details of a specific certificate.

Syntax

```
orapki cert display -cert certificate_location [-summary|-complete]
```

- The `-cert` parameter specifies the location of the certificate you want to display.

- You can use either the -summary or the -complete parameter to display the following information:

    - -summary displays the certificate and its expiration date

    - -complete displays additional certificate information, including the serial number and public key

orapki crl delete

The following sections describe this command.

Purpose

Use the orapki crl delete command to delete CRLs from Oracle Internet Directory. Note that the user who deletes CRLs from the directory by using orapki must be a member of the CRLAdmins (cn=CRLAdmins,cn=groups,%s_OracleContextDN%) directory group.

Prerequisites

None

Syntax

```
orapki crl delete -issuer issuer_name -ldap hostname:ssl_port -user username
[-wallet wallet_location] [-summary]
```

- The -issuer parameter specifies the name of the certificate authority (CA) who issued the CRL.

- The -ldap parameter specifies the host name and SSL port for the directory where the CRLs are to be deleted. Note that this must be a directory SSL port with no authentication.

    See Also: Refer to "Uploading CRLs to Oracle Internet Directory" on page 8-29 for more information about this port.

- The -user parameter specifies the user name of the directory user who has permission to delete CRLs from the CRL subtree in the directory.

- The -wallet parameter (optional) specifies the location of the wallet that contains the certificate of the certificate authority (CA) who issued the CRL. Using it causes the tool to verify the validity of the CRL against the CA's certificate prior to deleting it from the directory.

- The -summary parameter is optional. Using it causes the tool to print the CRL LDAP entry that was deleted.

orapki crl display

The following sections describe this command.

Purpose

Use the orapki crl display command to display specific CRLs that are stored in Oracle Internet Directory.

Syntax

```
orapki crl display -crl crl_location [-wallet wallet_location]
[-summary|-complete]
```

- The `-crl` parameter specifies the location of the CRL in the directory. It is convenient to paste the CRL location from the list that displays when you use the `orapki crl list` command. Refer to "orapki crl list" on page F-9

- The `-wallet` parameter (optional) specifies the location of the wallet that contains the certificate of the certificate authority (CA) who issued the CRL. Using it causes the tool to verify the validity of the CRL against the CA's certificate prior to displaying it.

- Choosing either the `-summary` or the `-complete` parameters displays the following information:

  - `-summary` provides a listing that contains the CRL issuer's name and the CRL's validity period

  - `-complete` provides a list of all revoked certificates that the CRL contains. Note that this option may take a long time to display, depending on the size of the CRL.

orapki crl hash

The following sections describe this command.

Purpose

Use the `orapki crl hash` command to generate a hash value of the certificate revocation list (CRL) issuer to identify the location of the CRL in the file system for certificate validation.

Syntax

```
orapki crl hash -crl crl_filename|URL [-wallet wallet_location] [-symlink|-copy]
crl_directory [-summary]
```

- The `-crl` parameter specifies the filename that contains the CRL or the URL where it can be found.

- The `-wallet` parameter (optional) specifies the location of the wallet that contains the certificate of the certificate authority (CA) who issued the CRL. Using it causes the tool to verify the validity of the CRL against the CA's certificate prior to uploading it to the directory.

- Depending on the operating system, use either the `-symlink` or the `-copy` parameter:

  - (UNIX) use `-symlink` to create a symbolic link to the CRL at the `crl_directory` location

  - (Windows) use `-copy` to create a copy of the CRL at the `crl_directory` location

- The `-summary` parameter (optional) causes the tool to display the CRL issuer's name.

orapki crl list

The following sections describe this command.

Purpose

Use the `orapki crl list` command to display a list of CRLs stored in Oracle Internet Directory. This is useful for browsing to locate a particular CRL to view or download to your local file system.

Syntax

```
orapki crl list -ldap hostname:ssl_port
```

The `-ldap` parameter specifies the host name and SSL port for the directory server from where you want to list CRLs. Note that this must be a directory SSL port with no authentication.

> **Tip:** "Uploading CRLs to Oracle Internet Directory" on page 8-29 for more information about this port

orapki crl upload

The following sections describe this command.

Purpose

Use the `orapki crl upload` command to upload certificate revocation lists (CRLs) to the CRL subtree in Oracle Internet Directory. Note that you must be a member of the directory administrative group CRLAdmins (`cn=CRLAdmins,cn=groups,%s_OracleContextDN%`) to upload CRLs to the directory.

Syntax

```
orapki crl upload -crl crl_location -ldap hostname:ssl_port -user username
[-wallet wallet_location] [-summary]
```

- The `-crl` parameter specifies the directory location or the URL where the CRL is located that you are uploading to the directory.

- The `-ldap` parameter specifies the host name and SSL port for the directory where you are uploading the CRLs. Note that this must be a directory SSL port with no authentication.

  > **See Also:** "Uploading CRLs to Oracle Internet Directory" on page 8-29 for more information about this port

- The `-user` parameter specifies the user name of the directory user who has permission to add CRLs to the CRL subtree in the directory.

- The `-wallet` parameter specifies the location of the wallet that contains the certificate of the certificate authority (CA) who issued the CRL. This is an optional parameter. Using it causes the tool to verify the validity of the CRL against the CA's certificate prior to uploading it to the directory.

- The `-summary` parameter is also optional. Using it causes the tool to display the CRL issuer's name and the LDAP entry where the CRL is stored in the directory.

orapki wallet add

The following sections describe this command.

Purpose

Use the `orapki wallet add` command to add certificate requests and certificates to an Oracle wallet.

Syntax

To add certificate requests:

```
orapki wallet add -wallet wallet_location -dn user_dn -keySize 512|1024|2048
```

- The `-wallet` parameter specifies the location of the wallet to which you want to add a certificate request.

- The `-dn` parameter specifies the distinguished name of the certificate owner.

- The `-keySize` parameter specifies the key size for the certificate.

- To sign the request, export it with the export option. Refer to "orapki wallet export" on page F-12

To add trusted certificates:

```
orapki wallet add -wallet wallet_location -trusted_cert -cert certificate_location
```

- The `-trusted_cert` parameter causes the tool to add the trusted certificate, at the location specified with `-cert`, to the wallet.

To add root certificates:

```
orapki wallet add -wallet wallet_location -dn certificate_dn -keySize
512|1024|2048 -self_signed -validity number_of_days
```

- The `-self_signed` parameter causes the tool to create a root certificate.

- The `-validity` parameter is mandatory. Use it to specify the number of days, starting from the current date, that this root certificate will be valid.

To add user certificates:

```
orapki wallet add -wallet wallet_location -user_cert -cert certificate_location
```

- The `-user_cert` parameter causes the tool to add the user certificate at the location specified with the `-cert` parameter to the wallet. Before you add a user certificate to a wallet, you must add all the trusted certificates that make up the certificate chain. If all trusted certificates are not installed in the wallet before you add the user certificate, then adding the user certificate will fail.

orapki wallet create

The following sections describe this command.

Purpose

Use the `orapki wallet create` command to create an Oracle wallet or to set auto login on for an Oracle wallet.

Syntax

```
orapki wallet create -wallet wallet_location [-auto_login]
```

- The `-wallet` parameter specifies a location for the new wallet or the location of the wallet for which you want to turn on auto login.

- The -auto_login parameter creates an **auto login wallet**, or it turns on automatic login for the wallet specified with the -wallet option.

    See Also: "Using Auto Login" on page 9-14 for details about auto login wallets

orapki wallet display

The following sections describe this command.

Purpose

Use the orapki wallet display command to view the certificate requests, user certificates, and trusted certificates in an Oracle wallet.

Syntax

```
orapki wallet display -wallet wallet_location
```

- The -wallet parameter specifies a location for the wallet you want to open if it is not located in the current working directory.

orapki wallet export

The following sections describe this command.

Purpose

Use the orapki wallet export command to export certificate requests and certificates from an Oracle wallet.

Syntax

To export a certificate from an Oracle wallet:

```
orapki wallet export -wallet wallet_location -dn certificate_dn -cert certificate_
filename
```

- The -wallet parameter specifies the location of the wallet from which you want to export the certificate.
- The -dn parameter specifies the distinguished name of the certificate.
- The -cert parameter specifies the name of the file that contains the exported certificate.

To export a certificate request from an Oracle wallet:

```
orapki wallet export -wallet wallet_location -dn certificate_request_dn -request
certificate_request_filename
```

- The -request parameter specifies the name of the file that contains the exported certificate request.

G

Entrust-Enabled SSL Authentication

Entrust Authority (formerly known as Entrust/PKI) is a suite of PKI products provided by Entrust, Inc., that provides certificate generation, certificate revocation, and key and certificate management. Oracle Advanced Security is integrated with Entrust Authority so both Entrust and Oracle users can enhance their Oracle environment security.

This appendix contains the following topics:

- Benefits of Entrust-Enabled Oracle Advanced Security
- Required System Components for Entrust-Enabled Oracle Advanced Security
- Entrust Authentication Process
- Enabling Entrust Authentication
- Issues and Restrictions that Apply to Entrust-Enabled SSL
- Troubleshooting Entrust In Oracle Advanced Security

Benefits of Entrust-Enabled Oracle Advanced Security

Entrust-enabled Oracle Advanced Security provides:

- Enhanced X.509-Based Authentication and Single Sign-On
- Integration with Entrust Authority Key Management
- Integration with Entrust Authority Certificate Revocation

> **Note:** Oracle Advanced Security has been certified as *Entrust-Ready* by Entrust, Inc., as of Release 8.1.7.

> **See Also:** http://www.entrust.com for more information:

Enhanced X.509-Based Authentication and Single Sign-On

Entrust-enabled Oracle Advanced Security supports the use of Entrust credentials for X.509-based authentication and single sign-on. Instead of using an Oracle wallet to hold user PKI credentials, Oracle Advanced Security can access PKI credentials that are created by Entrust Authority and held in an Entrust profile (a .epf file). Users who have deployed Entrust software within their enterprise are able to use it for authentication and single sign-on to Oracle Database.

Integration with Entrust Authority Key Management

Entrust-enabled Oracle Advanced Security uses the extensive key management and rollover functionality provided by Entrust Authority, which shields users from the complexity of a PKI deployment. For example, users are automatically notified when their certificates are expiring, and certificates are reissued according to preferences that administrators can configure.

Integration with Entrust Authority Certificate Revocation

Entrust provides a certificate authority component, which natively checks certificate revocation status and enables the revocation of certificates.

Users using Entrust credentials for authentication to Oracle are assured that the revocation status of the certificate is checked, and connections are prevented if the certificate is revoked.

Required System Components for Entrust-Enabled Oracle Advanced Security

To implement Entrust-enabled Oracle Advanced Security, the following system components are required:

- Entrust Authority for Oracle
- Entrust Authority Server Login Feature
- Entrust Authority IPSec Negotiator Toolkit

> **Note:** In the following sections, the term **client** refers to a client connecting to an Oracle database, and the term **server** refers to the host on which the Oracle database resides.

Contact your Entrust representative to get these components.

> **Note:** Oracle Advanced Security supports Entrust Authority Security Manager, Entrust Authority Server Login Feature, and Entrust Authority IPSec Negotiator Toolkit versions 6.0 and later.
>
> Contact your Entrust representative for the latest product classification and naming details.

Entrust Authority for Oracle

Entrust Authority for Oracle requires a database for storing information about Entrust users and the infrastructure, and a Lightweight Directory Access Protocol (LDAP)-compliant directory for information such as user names, public certificates, and certificate revocation lists.

Entrust Authority for Oracle comprises the following software components:

- Entrust Authority Security Manager
- Entrust Authority Self-Administration Server
- Entrust Entelligence Desktop Manager

Entrust Authority Security Manager

Entrust Authority Security Manager is the centerpiece of Entrust's PKI technology. It performs core certificate authority, certificate, and user management functions, such as creating users and user profiles containing the user's credentials.

> **Note:** Oracle only supports the use of Entrust-enabled Oracle Advanced Security with versions of Entrust Authority Security Manager that run on Oracle Database.

> **See Also:** Chapter 8, "Configuring Secure Sockets Layer Authentication", for information about certificate authorities.

Entrust Authority Security Manager supports unattended login, also called Server Login, which eliminates the need for a **Database Administrator** (DBA) to repeatedly enter a password for the Entrust profile on the server. With unattended login, the DBA need only enter a password once to open the Entrust profile for the server to authenticate itself to multiple incoming connections.

Entrust Authority Self-Administration Server

Entrust Authority Self-Administration Server is the administrator's secure interface to Entrust Authority Security Manager.

Entrust Entelligence Desktop Manager

Entrust Entelligence Desktop Manager provides support for user key management and single sign-on functionality on both clients and server by enabling Oracle Database server process access to incoming SSL connections.

> **Note:** Do not install Entrust Entelligence Desktop Manager on the server computer because it uses unattended login credentials files with .ual extensions.
>
> Refer to "Configuring Entrust on the Server" on page G-6 for information about creating .ual files.

Entrust Authority Server Login Feature

Entrust Authority Server Login Feature is required for single sign-on functionality on servers operating on UNIX platforms.

Entrust Authority Server Login Feature provides single sign-on by enabling Oracle Database server process access to incoming SSL connections. Without this capability, a database administrator or other privileged user would have to enter the password for the Entrust profile on the server for every incoming connection.

Contact your Entrust representative to get Entrust Authority Server Login Feature.

Entrust Authority IPSec Negotiator Toolkit

The Entrust Authority IPSec Negotiator Toolkit is required on both clients and servers for integrating the Oracle Advanced Security SSL stack with Entrust Authority, enabling SSL authentication to use Entrust profiles.

Contact your Entrust representative to get Entrust Authority IPSec Negotiator Toolkit.

Entrust Authentication Process

Figure G–1 illustrates the following Entrust authentication process:

1. The Entrust user on the Oracle client establishes a secure connection with the server using SSL and Entrust credentials.

2. The Oracle SSL adapter on the server communicates with the Entrust Authority to check the certificate revocation status of the Entrust user.

> **Note:** Figure G–1 does not include client and server profiles creation, which is presumed.

Figure G–1   Entrust Authentication Process
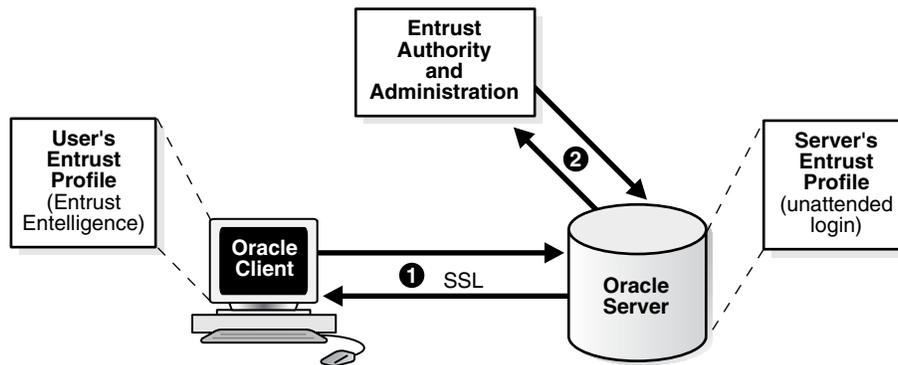
> **See Also:** "How SSL Works in an Oracle Environment: The SSL Handshake" on page 8-2

Enabling Entrust Authentication

This section describes the following tasks, which are required to configure Entrust-enabled Oracle Advanced Security SSL authentication:

- Creating Entrust Profiles
- Installing Oracle Advanced Security and Related Products for Entrust-Enabled SSL
- Configuring SSL on the Client and Server for Entrust-Enabled SSL
- Configuring Entrust on the Client
- Configuring Entrust on the Server
- Creating Entrust-Enabled Database Users
- Logging Into the Database Using Entrust-Enabled SSL

Creating Entrust Profiles

This section describes how to create Entrust profiles, which can be created by either administrators or users. On UNIX platforms, administrators create the Entrust profiles for all clients. On Windows platforms, users can create their own Entrust profiles.

Administrator-Created Entrust Profiles

Administrators create Entrust profiles as follows:

1. The Entrust administrator adds the Entrust user using the Entrust Authority Self-Administration Server.

> **See Also:** The Entrust administration documentation for information about creating Entrust Users

2. The administrator enters the user's name and password.

3. The Entrust Authority creates the profile, or `.epf` file.

4. The administrator securely sends all profile-related files to the user. The preset password can be changed by the user.

User-Created Entrust Profiles

Entrust users create their own Entrust profiles as follows:

1. The Entrust administrator adds the Entrust user using the Entrust Authority Self-Administration Server. In the New User dialog box, the Create Profile option should be deselected.

> **See Also:** The Entrust administration documentation for information about creating Entrust profiles

2. The user receives a secure e-mail notification from the administrator that contains a reference number, authorization code, and expiration date.

3. The user navigates to the Create Entrust Profiles screen in Entrust Entelligence Desktop Manager as follows:

   Start, **Programs**, **Entrust**, **Entrust Profiles**, **Create Entrust Profiles**

4. The user enters the reference number, authorization code, and expiration date provided in the e-mail notification, creating a profile, or `.epf` file, and the Entrust initialization file.

Installing Oracle Advanced Security and Related Products for Entrust-Enabled SSL

For Oracle Advanced Security 10*g* Release 2 (10.2), Entrust support installs in Typical mode. A single Oracle installation supports the use of both Oracle Wallets and Entrust profiles.

> **See Also:** Oracle Database operating system-specific installation documentation

Configuring SSL on the Client and Server for Entrust-Enabled SSL

Configure SSL on the client and server.

> **See Also:** Chapter 8, "Configuring Secure Sockets Layer Authentication", for information about configuring SSL on the client and server and skip the section that describes the Oracle wallet location.

Configuring Entrust on the Client

The steps for configuring Entrust on the client vary according to the type of platform:

- Configuring Entrust on a UNIX Client
- Configuring Entrust on a Windows Client

Configuring Entrust on a UNIX Client

If the client resides on a non-Windows platform, perform the following steps:

1. Set the `JAVA_HOME` variable to the JDK or JRE location.

For example:

```
>setenv JAVA_HOME $ORACLE_HOME/JRE
```

2. Set `WALLET_LOCATION` in the `sqlnet.ora` file.

For example:

```
WALLET_LOCATION=
(SOURCE=
(METHOD=entr)
(METHOD_DATA =
    (PROFILE=profile_location)
    (INIFILE=initialization_file_location)
)
)
```

Configuring Entrust on a Windows Client

If the client resides on a Windows platform, ensure that the Entrust Entelligence Desktop Manager component is installed on the client and perform the following steps to set up the Entrust credentials.

1. Set the `WALLET_LOCATION` parameter in the `sqlnet.ora` file.

For example:

```
WALLET_LOCATION=
(SOURCE=
(METHOD=entr)
(METHOD_DATA=
    (INIFILE=initialization_file_location)
)
)
```

where `initialization_file_location` is the path to the `.ini` file.

2. Select the Entrust icon on the system tray to open the Entrust_Login dialog box.

3. Log on to Entrust by entering the profile name and password.

Configuring Entrust on the Server

The steps for configuring Entrust on the server vary according to the type of platform:

■ Configuring Entrust on a UNIX Server

■ Configuring Entrust on a Windows Server

Configuring Entrust on a UNIX Server

If the server is a UNIX platform, ensure that the Entrust/Server Login Toolkit component is installed on the server and perform the following steps:

> **See Also:** "Required System Components for Entrust-Enabled Oracle Advanced Security" on page G-2 for information about downloading the Entrust Server Login toolkit.

1. Stop the Oracle database instance.

2. Set the `WALLET_LOCATION` parameter in the `sqlnet.ora` and `listener.ora` files to specify the paths to the server's profile and the Entrust initialization file:

```
WALLET_LOCATION =
  (SOURCE =
      (METHOD = ENTR)
      (METHOD_DATA =
          (PROFILE = profile_location)
          (INIFILE = initialization_file_location)
      )
  )
```

3. Set the `CLASSPATH` environment variable to include the following paths:

```
$ORACLE_HOME/JRE/lib/rt.jar
$ORACLE_HOME/JRE/lib/i18n.jar
$ORACLE_HOME/jlib/ewt*.jar
$ORACLE_HOME/jlib/help*.jar
$ORACLE_HOME/jlib/share*.jar
$ORACLE_HOME/jlib/swingall*.jar
$ORACLE_HOME/network/jlib/netentrust.jar
```

4. Enter the `etbinder` command to create unattended login credentials, or `.ual` files by using the following steps:

    a. Set the `PATH` environment variable to include the path to the `etbinder` command, which is located in the `/bin` directory where the Server Login Toolkit is installed.

    b. Set the `LD_LIBRARY_PATH` to include the path to the Entrust libraries.

    c. Set the `SSL_ENTRUST_INI` environment variable to include the full path to the Entrust initialization file.

    d. Enter the command as follows:

```
etbinder
```

    e. When prompted to enter the location of the profile file, enter the full path name, including the name of the file. Then, when prompted, type in the password.

    A message displays indicating that the credentials file (`filename.ual`) has been created.

> **Note:** Ensure that the listener has a TCPS listening endpoint, then start the listener.

5. Start the Oracle database instance.

Configuring Entrust on a Windows Server

If the server is on a Windows platform, perform the following steps:

> **See Also:** "Required System Components for Entrust-Enabled Oracle Advanced Security" for information about downloading Entrust Entelligence Desktop Manager.

1. Stop the Oracle database instance.

2. Set the `WALLET_LOCATION` parameter in the `sqlnet.ora` and `listener.ora` files to specify the paths to the server's profile and the Entrust initialization file:

```
     WALLET_LOCATION =
(SOURCE =
(METHOD = ENTR)
(METHOD_DATA =
     (PROFILE = profile_location)
     (INIFILE = initialization_file_location)
)
)
```

3. Run the Entrust binder command to create unattended login credentials, which are files with a `.ual` extension. Ensure that the owner of the `.ual` file is the same as the owner of the Oracle service.

To run the binder command Select

Start, **Programs**, **Entrust Toolkit**, **Server Login**, **Entrust Binder**

Enter the path to the profile, the password, and the path to the Entrust initialization file. A message informs you that you have successfully created a credential file.

4. Start the Oracle database instance.

> **Note:** For all Windows environments, Oracle Corporation recommends that you do not install Entrust Entelligence Desktop Manager on the server computer.

Creating Entrust-Enabled Database Users

Create global users in the database based on the **distinguished name (DN)** of each Entrust user.

For example:

```
SQL> create user jdoe identified globally as 'cn=jdoe,o=oracle,c=us';
```

where `"cn=jdoe, o=oracle, c=us"` is the Entrust distinguished name of the user.

Logging Into the Database Using Entrust-Enabled SSL

1. Use SQL*Plus to connect to the Oracle instance as follows:

```
sqlplus /@net_service_name
```

where `net_service_name` is the service name of the Oracle instance.

The Entrust_Login dialog box is displayed.

2. Enter the path to the profile and the password.

3. If you did not specify a value for the `WALLET_LOCATION` parameter, you are prompted to enter the path to the Entrust initialization file.

> **Note:** Oracle Corporation recommends that the initialization file be specified in the `WALLET_LOCATION` parameter file.

Issues and Restrictions that Apply to Entrust-Enabled SSL

An application must be specifically modified to work with Entrust. If a product is designated as Entrust-ready, then it has been integrated with Entrust by using an Entrust toolkit.

For example, Oracle has modified its SSL libraries to access an Entrust profile instead of an Oracle wallet.

In addition, the following restrictions apply:

- The use of Entrust components for digital signatures in applications based on Oracle is not supported.

- The Entrust-enabled Oracle Advanced Security integration is only supported with versions of Entrust Authority Release 6.0 and later running on Oracle Database.

- The use of earlier releases of Entrust Authority with Entrust-enabled Oracle Advanced Security is not supported.

- Interoperability between Entrust and non-Entrust PKIs is not supported.

- Entrust has certified Oracle Internet Directory version 2.1.1 for Release 8.1.7 and subsequent releases.

Troubleshooting Entrust In Oracle Advanced Security

This section describes how to diagnose errors returned from Entrust to Oracle Advanced Security users.

> **Note:** Entrust returns the following generic error message to Oracle Advanced Security users:
>
> ```
> ORA-28890 "Entrust Login Failed"
> ```
>
> This troubleshooting section describes how to get more details about the underlying error, and how to diagnose the problem.

Error Messages Returned When Running Entrust on Any Platform

You may encounter the following error messages regardless of what platform you are running Entrust on.

ORA-28890 Entrust Login Failed

Cause: SQL*Plus login on an Entrust-enabled Oracle client errors out with this generic error message. This error can be caused by a number of problems, including the following causes:

- Entrust /Authority is not online

- Invalid Entrust profile password specified

- Invalid path to the Entrust profile specified

- Invalid Entrust initialization file specified

- Entrust Server Login program has not executed on the server

Action: To get more detail on the Entrust error, turn on tracing for SQL*Plus and the trace output should indicate the Entrust failure code. Enable tracing by specifying the following parameters in the `sqlnet.ora` file:

On the client:

- `TRACE_LEVEL_CLIENT=16`

- `TRACE_DIRECTORY_CLIENT=`*`valid_client_directory_name`*

- `TRACE_FILE_CLIENT=client`

- `TRACE_UNIQUE_CLIENT=ON`

On the server:

- `TRACE_LEVEL_SERVER=16`

- `TRACE_DIRECTORY_SERVER=`*`valid_server_directory name`*

- `TRACE_FILE_SERVER=server`

- `TRACE_UNIQUE_SERVER=ON`

Search for and locate the string `IKMP` in the generated trace file. Adjacent to this string, error messages are listed that provide details about the problem you are encountering. This detailed error code information is returned by the Entrust API.

> **Note:** The following are examples of valid client directory names for setting the `TRACE_DIRECTORY_CLIENT` or `TRACE_DIRECTORY_SERVER` parameters in the `sqlnet.ora` file:
>
> - (UNIX) `/tmp`
> - (Windows) `C:\TEMP`

ORA-28890 Entrust Login Failed (GUI does not display on the client)

    Cause: The `WALLET_LOCATION` parameter does not specify the Entrust initialization file location in the client side `sqlnet.ora` file.

    Action: Ensure that the location of the Entrust initialization file is specified in the `WALLET_LOCATION` parameter in the `sqlnet.ora` file on the client.

> **See Also:**
>
> - "Configuring Entrust on a UNIX Client" on page G-6
> - "Configuring Entrust on a Windows Client" on page G-6

Error Messages Returned When Running Entrust on Windows Platforms

You may encounter the following error messages if you are running Entrust on a Windows platform.

The software authentication failed. (error code - 162).

    Cause: Due to a known FIPS mode incompatibility, Entrust logins may fail and return this error message.

    Action: Contact Entrust support to resolve this issue.

Algorithm self-test failed. (error code - 176).

    Cause: Due to a known symbol conflict between Entrust and Oracle libraries, Entrust login may fail and return this error message.

    Action: Contact Entrust support to resolve this issue.

TNS-12560: TNS protocol adapter error TNS-00558> Entrust Login Failed ORACLE SERVER (*host_name*)

This error may occur in the `listener.log` *file on the server when you attempt to log in to Entrust.*

Cause: If you configure the client by making the following recommended changes:

- Remove the `.ual` file

- De-install the Server Login

- Specify the Entrust initialization file location in the `SSL_ENTRUST_INI_FILE` parameter in the client `sqlnet.ora` file

then the server may not be able to authenticate the client when you enter the following command:

```
sqlplus/@net_service_name
```

Action: Perform the following tasks to enable tracing on the server:

1. Select **Control Panel**, then **Services**.

2. In the Services dialog box, double click **OracleTNSListener** and change the Log On As from the System Account to the account that is currently logged in. This enables the server process to read the `.ual` file. Click **OK** to make the change and you are returned to the Services dialog box.

   In the Services dialog box, make the same changes for OracleService.

3. Make the following changes to the `listener.ora` file:

   - Specify only `TCPS` as the `PROTOCOL` in the listener `ADDRESS`. For example, change all of the `PROTOCOL` definitions to `TCPS` as follows:

   ```
   listener_name=
      (DESCRIPTION=
          (ADDRESS=(PROTOCOL=TCPS) (KEY=extproc0))
          (ADDRESS=(PROTOCOL=TCPS) (HOST=sales-pc) (PORT=1521)))
   ```

   Bringing up the listener only using `TCPS` will show whether there is a problem accessing the Entrust profile when you turn on tracing.

   - Set the `SSL_CLIENT_AUTHENTICATION` parameter to `FALSE` as follows:

   ```
   SSL_CLIENT_AUTHENTICATION=FALSE
   ```

   - Turn on tracing by setting the following parameters:

   ```
   TRACE_LEVEL_LISTENER=16
   TRACE_DIRECTORY_LISTENER=C:\temp
   ```

   The trace file is created in the `C:\temp` directory.

4. Make the following changes to the `sqlnet.ora` file to turn on tracing:

   ```
   TRACE_LEVEL_SERVER=16
   TRACE_DIRECTORY_SERVER=C:\temp
   ```

   The trace file is created in the `C:\temp` directory.

5. Ensure that Entrust Entelligence Desktop Manager is not installed on the server.

Search for and locate the string `fail` or `ntz*` function calls. Adjacent to these, error messages are listed that provide details about the problem you are encountering.

General Checklist for Running Entrust on Any Platform

The following items apply to all platforms:

1. Confirm that the Entrust Authority is online.

2. Confirm that the `.ual` file is generated. These files are created for unattended login credentials.

   > **Note:** Oracle recommends that you generate an unattended login credential file (`.ual` file) for the server only. If you generate a `.ual` file for the server only, then when users attempt to log in, they are presented a GUI that prompts them for their password and their Entrust profile name. After users supply this information, the connection request is forwarded to the Entrust server, which looks up the revocation file and the `.ual` file to determine the permissions for granting the request.

3. Confirm that the Entrust initialization file contains the following entry in the first section that specifies the Entrust Settings:

   ```
   IdentityLibrary=location
   ```

   The full path to the location of the `libidapi.so` file should be specified in the `IdentityLibrary` parameter. This parameter setting enables generating a `.ual` file on the server.

4. Ensure that all Entrust toolkits, including the Entrust IPSEC Negotiator toolkit and the Server Login toolkit, are the same version so they are compatible.

5. Ensure that you have specified TCP/IP with SSL in the `SQLNET.AUTHENTICATION_SERVICES` parameter in the `sqlnet.ora` file as shown in the following example:

   ```
   SQLNET.AUTHENTICATION_SERVICES=(tcps, authentication_type1, authentication_
      type2)
   ```

Checklist for Entrust Installations on Windows

The following checklist items apply only to Entrust installations on the Windows platform.

1. Ensure that you are logged into Entrust Entelligence Desktop Manager and retry.

2. Select **Windows**, then **Control Panel**, and click **Services** to confirm that the Entrust Login Interface service has started and is running.

3. Confirm that the Entrust initialization file location is specified in the `SSL_ENTRUST_INI_FILE` parameter of the `sqlnet.ora` file. However, if you select not to specify the location there, then the Entrust initialization file must reside in `c:\WINNT`.

4. Ensure that you are not running Entrust Entelligence Desktop Manager if your database is running on a Microsoft platform. If this is the case, then only the `.ual` file, which enables unattended login, is required.

   > **See Also:** Step 4 of "Configuring Entrust on a Windows Server" on page G-7 for information about creating a `.ual` file with the Entrust binder command.

5. Confirm that Entrust Authority, as specified in the Entrust Initialization file, is accessible and running.

6. Confirm that the profile password is correctly entered.

7. If an Oracle database server fails to log in to Entrust, confirm that the unattended login credential file (`.ual`) is generated using a valid password. Also, confirm that the versions for Entrust Server Login toolkit and Entrust IPSEC Negotiator toolkit match (that is, that the IPSec Toolkit 6.0 works with Server Login Toolkit 6.0).

8. Ensure that the Entrust initialization file has the following entry in the first section, Entrust Settings:

```
IdentityLibrary = location
```

where `location` is the location of `libidapi.so`, including the file name.

Glossary

access control

The ability of a system to grant or limit access to specific data for specific clients or groups of clients.

Access Control Lists (ACLs)

The group of access directives that you define. The directives grant levels of access to specific data for specific clients, or groups of clients, or both.

Advanced Encryption Standard

Advanced Encryption Standard (AES) is a new cryptographic algorithm that has been approved by the National Institute of Standards and Technology as a replacement for DES. The AES standard is available in Federal Information Processing Standards Publication 197. The AES algorithm is a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits.

AES

See **Advanced Encryption Standard**

attribute

An item of information that describes some aspect of an entry in an LDAP directory. An entry comprises a set of attributes, each of which belongs to an **object class**. Moreover, each attribute has both a *type*, which describes the kind of information in the attribute, and a *value,* which contains the actual data.

authentication

The process of verifying the identity of a user, device, or other entity in a computer system, often as a prerequisite to granting access to resources in a system. A recipient of an authenticated message can be certain of the message's origin (its sender). Authentication is presumed to preclude the possibility that another party has impersonated the sender.

authentication method

A security method that verifies a user's, client's, or server's identity in distributed environments. Network authentication methods can also provide the benefit of **single sign-on (SSO)** for users. The following authentication methods are supported in Oracle Database when Oracle Advanced Security is installed:

- **Kerberos**

- **RADIUS**

- **Secure Sockets Layer (SSL)**

- **Windows native authentication**

authorization

Permission given to a user, program, or process to access an object or set of objects. In Oracle, authorization is done through the role mechanism. A single person or a group of people can be granted a role or a group of roles. A role, in turn, can be granted other roles. The set of privileges available to an authenticated entity.

auto login wallet

An Oracle Wallet Manager feature that enables PKI- or password-based access to services without providing credentials at the time of access. This auto login access stays in effect until the auto login feature is disabled for that wallet. File system permissions provide the necessary security for auto login wallets. When auto login is enabled for a wallet, it is only available to the operating system user who created that wallet. Sometimes these are called "SSO wallets" because they provide single sign-on capability.

base

The root of a subtree search in an **LDAP**-compliant directory.

CA

See **certificate authority**

CDS

See **Cell Directory Services (CDS)**

Cell Directory Services (CDS)

An external naming method that enables users to use Oracle tools transparently and applications to access Oracle Database databases in a Distributed Computing Environment (DCE).

certificate

An ITU x.509 v3 standard data structure that securely binds an identify to a public key.

A certificate is created when an entity's public key is signed by a trusted identity, a certificate authority. The certificate ensures that the entity's information is correct and that the public key actually belongs to that entity.

A certificate contains the entity's name, identifying information, and public key. It is also likely to contain a serial number, expiration date, and information about the rights, uses, and privileges associated with the certificate. Finally, it contains information about the certificate authority that issued it.

certificate authority

A trusted third party that certifies that other entities—users, databases, administrators, clients, servers—are who they say they are. When it certifies a user, the certificate authority first seeks verification that the user is not on the certificate revocation list (CRL), then verifies the user's identity and grants a certificate, signing it with the certificate authority's private key. The certificate authority has its own certificate and public key which it publishes. Servers and clients use these to verify signatures the certificate authority has made. A certificate authority might be an external company that offers certificate services, or an internal organization such as a corporate MIS department.

certificate chain

An ordered list of certificates containing an end-user or subscriber certificate and its certificate authority certificates.

certificate request

A certificate request, which consists of three parts: certification request information, a signature algorithm identifier, and a digital signature on the certification request information. The certification request information consists of the subject's distinguished name, public key, and an optional set of attributes. The attributes may provide additional information about the subject identity, such as postal address, or a challenge password by which the subject entity may later request certificate revocation. See **PKCS #10**

certificate revocation lists

(CRLs) Signed data structures that contain a list of revoked **certificate**s. The authenticity and integrity of the CRL is provided by a digital signature appended to it. Usually, the CRL signer is the same entity that signed the issued certificate.

checksumming

A mechanism that computes a value for a message packet, based on the data it contains, and passes it along with the data to authenticate that the data has not been tampered with. The recipient of the data recomputes the cryptographic checksum and compares it with the cryptographic checksum passed with the data; if they match, it is "probabilistic" proof the data was not tampered with during transmission.

Cipher Block Chaining (CBC)

An encryption method that protects against block replay attacks by making the encryption of a cipher block dependent on all blocks that precede it; it is designed to make unauthorized decryption incrementally more difficult. Oracle Advanced Security employs *outer* cipher block chaining because it is more secure than *inner* cipher block chaining, with no material performance penalty.

cipher suite

A set of authentication, encryption, and data integrity algorithms used for exchanging messages between network nodes. During an SSL handshake, for example, the two nodes negotiate to see which cipher suite they will use when transmitting messages back and forth.

cipher suite name

Cipher suites describe the kind of cryptographics protection that is used by connections in a particular session.

ciphertext

Message text that has been encrypted.

cleartext

Unencrypted plain text.

client

A client relies on a service. A client can sometimes be a user, sometimes a process acting on behalf of the user during a database link (sometimes called a proxy).

confidentiality

A function of cryptography. Confidentiality guarantees that only the intended recipient(s) of a message can view the message (decrypt the ciphertext).

connect descriptor

A specially formatted description of the destination for a network connection. A connect descriptor contains destination **service** and network route information. The destination service is indicated by using its service name for Oracle9*i* or Oracle8*i* databases or its Oracle **system identifier (SID)** for Oracle databases version 8.0. The network route provides, at a minimum, the location of the **listener** through use of a network address. See **connect identifier**

connect identifier

A **connect descriptor** or a name that maps to a connect descriptor. A connect identifier can be a **net service name**, database **service name**, or **net service alias**. Users initiate a connect request by passing a user name and password along with a connect identifier in a connect string for the service to which they wish to connect:

```
CONNECT username/password@connect_identifier
```

connect string

Information the user passes to a **service** to connect, such as **user name**, password and **net service name**. For example:

```
CONNECT username/password@net_service_name
```

credentials

A **user name**, password, or certificate used to gain access to the database.

CRL

See **certificate revocation lists**

CRL Distribution Point

(CRL DP) An optional extension specified by the X.509 version 3 certificate standard, which indicates the location of the Partitioned CRL where revocation information for a certificate is stored. Typically, the value in this extension is in the form of a URL. CRL DPs allow revocation information within a single **certificate authority** domain to be posted in multiple CRLs. CRL DPs subdivide revocation information into more manageable pieces to avoid proliferating voluminous CRLs, thereby providing performance benefits. For example, a CRL DP is specified in the certificate and can point to a file on a Web server from which that certificate's revocation information can be downloaded.

CRL DP

See **CRL Distribution Point**

cryptography

The practice of encoding and decoding data, resulting in secure messages.

data dictionary

A set of read-only tables that provide information about a database.

Data Encryption Standard (DES)

The U.S. data encryption standard.

Database Administrator

(1) A person responsible for operating and maintaining an Oracle Server or a database application. (2) An Oracle user name that has been given DBA privileges and can perform database administration functions. Usually the two meanings coincide. Many sites have multiple DBAs.

database alias

See **net service name**

Database Installation Administrator

Also called a database creator. This administrator is in charge of creating new databases. This includes registering each database in the directory using the Database Configuration Assistant. This administrator has create and modify access to database service objects and attributes. This administrator can also modify the Default **domain**.

database link

A network object stored in the local database or in the network definition that identifies a remote database, a communication path to that database, and optionally, a user name and password. Once defined, the database link is used to access the remote database.

A public or private database link from one database to another is created on the local database by a DBA or user.

A global database link is created automatically from each database to every other database in a network with Oracle Names. Global database links are stored in the network definition.

database method

See **Oracle database method**

database password verifier

A database password verifier is an irreversible value that is derived from the user's database password. This value is used during password authentication to the database to prove the identity of the connecting user.

Database Security Administrator

The highest level administrator for database enterprise user security. This administrator has permissions on all of the enterprise domains and is responsible for:

■ Administering the Oracle DBSecurityAdmins and OracleDBCreators groups.

Creating new **enterprise domains**.

■ Moving databases from one **domain** to another within the enterprise.

DCE

See **Distributed Computing Environment (DCE)**

decryption

The process of converting the contents of an encrypted message (ciphertext) back into its original readable format (plaintext).

DES

See **Data Encryption Standard (DES)**

dictionary attack

A common attack on passwords. The attacker creates a list of many common passwords and encrypts them. Then the attacker steals a file containing encrypted passwords and compares it to his list of encrypted common passwords. If any of the encrypted password values (called verifiers) match, then the attacker can steal the corresponding password. Dictionary attacks can be avoided by using "salt" on the password before encryption. See **salt**

Diffie-Hellman key negotiation algorithm

This is a method that lets two parties communicating over an insecure channel to agree upon a random number known only to them. Though the parties exchange information over the insecure channel during execution of the Diffie-Hellman key negotiation algorithm, it is computationally infeasible for an attacker to deduce the random number they agree upon by analyzing their network communications. Oracle Advanced Security uses the Diffie-Hellman key negotiation algorithm to generate session keys.

digital signature

A digital signature is created when a public key algorithm is used to sign the sender's message with the sender's private key. The digital signature assures that the document is authentic, has not been forged by another entity, has not been altered, and cannot be repudiated by the sender.

directory information tree (DIT)

A hierarchical tree-like structure consisting of the DNs of the entries in an LDAP directory. See **distinguished name (DN)**

directory naming

A **naming method** that resolves a database service, **net service name**, or **net service alias** to a **connect descriptor** stored in a central directory server. A

directory naming context

A subtree which is of significance within a directory server. It is usually the top of some organizational subtree. Some directories only permit one such context which is fixed; others permit none to many to be configured by the directory administrator.

Distributed Computing Environment (DCE)

A set of integrated network services that works across multiple systems to provide a distributed environment. The middleware between distributed applications and the operating system or networkservices; based on a client/server computing model.

distinguished name (DN)

The unique name of a directory entry. It is comprised of all of the individual names of the parent entries back to the root entry of the directory information tree. See **directory information tree (DIT)**

domain

Any tree or subtree within the **Domain Name System (DNS)** namespace. Domain most commonly refers to a group of computers whose host names share a common suffix, the domain name.

Domain Name System (DNS)

A system for naming computers and network services that is organized into a hierarchy of **domain**s. DNS is used in TCP/IP networks to locate computers through user-friendly names. DNS resolves a friendly name into an IP address, which is understood by computers.

In **Oracle Net Services**, DNS translates the host name in a TCP/IP address into an IP address.

encrypted text

Text that has been encrypted, using an encryption algorithm; the output stream of an encryption process. On its face, it is not readable or decipherable, without first being subject to **decryption**. Also called **ciphertext**. Encrypted text ultimately originates as **plaintext**.

encryption

The process of disguising a message rendering it unreadable to any but the intended recipient.

enterprise domain

A directory construct that consists of a group of databases and **enterprise role**s. A database should only exist in one enterprise domain at any time. Enterprise domains are different from Windows 2000 domains, which are collections of computers that share a common directory database.

Enterprise Domain Administrator

User authorized to manage a specific **enterprise domain**, including the authority to add new enterprise domain administrators.

enterprise role

Access privileges assigned to **enterprise user**s. A set of Oracle role-based **authorization**s across one or more databases in an **enterprise domain**. Enterprise roles are stored in the directory and contain one or more **global role**s.

enterprise user

A user defined and managed in a directory. Each enterprise user has a unique identify across an enterprise.

entry

The building block of a directory, it contains information about an object of interest to directory users.

external authentication

Verification of a user identity by a third party authentication service, such as Kerberos or RADIUS.

file system method

Storing fingerprint templates in files when configuring Identix Biometric authentication. The alternative is to use the **Oracle database method**.

Federal Information Processing Standard (FIPS)

A U.S. government standard that defines security requirements for cryptographic modules—employed within a security system protecting unclassified information

within computer and telecommunication systems. Published by the National Institute of Standards and Technology (NIST).

FIPS

See **Federal Information Processing Standard (FIPS)**

forest

A group of one or more Active Directory trees that trust each other. All trees in a forest share a common **schema**, configuration, and global catalog. When a forest contains multiple trees, the trees do not form a contiguous namespace. All trees in a given forest trust each other through transitive bidirectional trust relationships.

forwardable ticket-granting ticket

In Kerberos. A service ticket with the FORWARDABLE flag set. This flag enables authentication forwarding without requiring the user to enter a password again.

GDS

See **Global Directory Service (GDS)**

Global Directory Service (GDS)

GDS is the **DCE** directory service that acts as an agent between **DCE CDS** and any X.500 directory service. Both GDS and **CDS** are obsolete; they are only used by **DCE**.

global role

A role managed in a directory, but its privileges are contained within a single database. A global role is created in a database by using the following syntax:

```
CREATE ROLE role_name IDENTIFIED GLOBALLY;
```

grid computing

A computing architecture that coordinates large numbers of servers and storage to act as a single large computer. Oracle Grid Computing creates a flexible, on-demand computing resource for all enterprise computing needs. Applications running on the Oracle 10*g* grid computing infrastructure can take advantage of common infrastructure services for failover, software provisioning, and management. Oracle Grid Computing analyzes demand for resources and adjusts supply accordingly.

HTTP

Hypertext Transfer Protocol: The set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. Relative to the TCP/IP suite of protocols (which are the basis for information exchange on the Internet), HTTP is an application protocol.

HTTPS

The use of Secure Sockets Layer (SSL) as a sublayer under the regular HTTP application layer.

identity

The combination of the public key and any other public information for an entity. The public information may include user identification data such as, for example, an e-mail address. A user certified as being the entity it claims to be.

identity management

The creation, management, and use of online, or digital, entities. Identity management involves securely managing the full life cycle of a digital identity from creation (provisioning of digital identities) to maintenance (enforcing organizational policies regarding access to electronic resources), and, finally, to termination.

identity management realm

A subtree in Oracle Internet Directory, including not only an **Oracle Context**, but also additional subtrees for users and groups, each of which are protected with access control lists.

initial ticket

In Kerberos authentication, an initial ticket or ticket granting ticket (TGT) identifies the user as having the right to ask for additional service tickets. No tickets can be obtained without an initial ticket. An initial ticket is retrieved by running the `okinit` program and providing a password.

instance

Every running Oracle database is associated with an Oracle instance. When a database is started on a database server (regardless of the type of computer), Oracle allocates a memory area called the **System Global Area (SGA)** and starts an Oracle process. This combination of the SGA and an Oracle process is called an instance. The memory and the process of an instance manage the associated database's data efficiently and serve the one or more users of the database.

integrity

The guarantee that the contents of the message received were not altered from the contents of the original message sent.

java code obfuscation

Java code **obfuscation** is used to protect Java programs from reverse engineering. A special program (an obfuscator) is used to scramble Java symbols found in the code. The process leaves the original program structure intact, letting the program run correctly while changing the names of the classes, methods, and variables in order to hide the intended behavior. Although it is possible to decompile and read non-obfuscated Java code, the obfuscated Java code is sufficiently difficult to decompile to satisfy U.S. government export controls.

Java Database Connectivity (JDBC)

An industry-standard Java interface for connecting to a relational database from a Java program, defined by Sun Microsystems.

JDBC

See **Java Database Connectivity (JDBC)**

KDC

Key Distribution Center. In Kerberos authentication, the KDC maintains a list of user principals and is contacted through the `kinit` (`okinit` is the Oracle version) program for the user's **initial ticket**. Frequently, the KDC and the Ticket Granting Service are combined into the same entity and are simply referred to as the KDC. The Ticket Granting Service maintains a list of service principals and is contacted when a user wants to authenticate to a server providing such a service. The KDC is a trusted third party that must run on a secure host. It creates ticket-granting tickets and service tickets.

Kerberos

A network authentication service developed under Massachusetts Institute of Technology's Project Athena that strengthens security in distributed environments. Kerberos is a trusted third-party authentication system that relies on shared secrets and assumes that the third party is secure. It provides single sign-on capabilities and database link authentication (MIT Kerberos only) for users, provides centralized password storage, and enhances PC security.

key

When encrypting data, a key is a value which determines the ciphertext that a given algorithm will produce from given plaintext. When decrypting data, a key is a value required to correctly decrypt a ciphertext. A ciphertext is decrypted correctly only if the correct key is supplied.

With a symmetric encryption algorithm, the same key is used for both encryption and decryption of the same data. With an asymmetric encryption algorithm (also called a public-key encryption algorithm or public-key cryptosystem), different keys are used for encryption and decryption of the same data.

key pair

A **public key** and its associated **private key**. See **public and private key pair**

keytab file

A Kerberos key table file containing one or more service keys. Hosts or services use *keytab* files in the same way as users use their passwords.

kinstance

An instantiation or location of a Kerberos authenticated service. This is an arbitrary string, but the host Computer name for a service is typically specified.

kservice

An arbitrary name of a Kerberos service object.

LDAP

See **Lightweight Directory Access Protocol (LDAP)**

ldap.ora file

A file created by Oracle Net Configuration Assistant that contains the following directory server access information:

- Type of directory server

- Location of the directory server

- Default identity management realm or Oracle Context (including ports) that the client or server will use

Lightweight Directory Access Protocol (LDAP)

A standard, extensible directory access protocol. It is a common language that LDAP clients and servers use to communicate. The framework of design conventions supporting industry-standard directory products, such as the Oracle Internet Directory.

listener

A process that resides on the server whose responsibility is to listen for incoming client connection requests and manage the traffic to the server.

Every time a client requests a network session with a server, a listener receives the actual request. If the client information matches the listener information, then the listener grants a connection to the server.

listener.ora file

A configuration file for the listener that identifies the:

- Listener name

- Protocol addresses that it is accepting connection requests on

- Services it is listening for

The `listener.ora` file typically resides in $*ORACLE_HOME*/network/admin on UNIX platforms and *ORACLE_BASE**ORACLE_HOME*\network\admin on Windows.

man-in-the-middle

A security attack characterized by the third-party, surreptitious interception of a message, wherein the third-party, the *man-in-the-middle*, decrypts the message, re-encrypts it (with or without alteration of the original message), and re-transmits it to the originally-intended recipient—all without the knowledge of the legitimate sender and receiver. This type of security attack works only in the absence of **authentication**.

MD5

An algorithm that assures data integrity by generating a 128-bit cryptographic message digest value from given data. If as little as a single bit value in the data is modified, the MD5 checksum for the data changes. Forgery of data in a way that will cause MD5 to generate the same result as that for the original data is considered computationally infeasible.

message authentication code

Also known as data authentication code (DAC). A **checksumming** with the addition of a secret key. Only someone with the key can verify the cryptographic checksum.

message digest

See **checksumming**

naming method

The resolution method used by a client application to resolve a **connect identifier** to a **connect descriptor** when attempting to connect to a database service.

National Institute of Standards and Technology (NIST)

An agency within the U.S. Department of Commerce responsible for the development of security standards related to the design, acquisition, and implementation of cryptographic-based security systems within computer and telecommunication systems, operated by a Federal agency or by a contractor of a Federal agency or other organization that processes information on behalf of the Federal Government to accomplish a Federal function.

net service alias

An alternative name for a **directory naming** object in a directory server. A directory server stores net service aliases for any defined **net service name** or database service. A net service alias entry does not have connect descriptor information. Instead, it only references the location of the object for which it is an alias. When a client requests a directory lookup of a net service alias, the directory determines that the entry is a net service alias and completes the lookup as if it was actually the entry it is referencing.

net service name

The name used by clients to identify a database server. A net service name is mapped to a port number and protocol. Also known as a **connect string**, or **database alias**.

network authentication service

A means for authenticating clients to servers, servers to servers, and users to both clients and servers in distributed environments. A network authentication service is a repository for storing information about users and the services on different servers to which they have access, as well as information about clients and servers on the network. An authentication server can be a physically separate computer, or it can be a facility co-located on another server within the system. To ensure availability, some authentication services may be replicated to avoid a single point of failure.

network listener

A listener on a server that listens for connection requests for one or more databases on one or more protocols. See **listener**

NIST

See **Federal Information Processing Standard (FIPS)**

non-repudiation

Incontestable proof of the origin, delivery, submission, or transmission of a message.

obfuscation

A process by which information is scrambled into a non-readable form, such that it is extremely difficult to de-scramble if the algorithm used for scrambling is not known.

obfuscator

A special program used to obfuscate Java source code. See **obfuscation**

object class

A named group of **attributes**. When you want to assign attributes to an entry, you do so by assigning to that entry the object classes that hold those attributes. All objects associated with the same object class share the same attributes.

Oracle Context

1. An entry in an LDAP-compliant internet directory called `cn=OracleContext`, under which all Oracle software relevant information is kept, including entries for **Oracle Net Services** directory naming and **checksumming** security.

There can be one or more Oracle Contexts in a directory. An Oracle Context is usually located in an **identity management realm**.

Oracle database method

Using an Oracle database to store fingerprint templates when configuring Indentix Biometric authentication. The alternative is to use the **file system method**.

Oracle Net Services

An Oracle product that enables two or more computers that run the Oracle server or Oracle tools such as Designer/2000 to exchange data through a third-party network. Oracle Net Services support distributed processing and distributed database capability. Oracle Net Services is an open system because it is independent of the communication protocol, and users can interface Oracle Net to many network environments.

Oracle PKI certificate usages

Defines Oracle application types that a **certificate** supports.

Password-Accessible Domains List

A group of **enterprise domain**s configured to accept connections from password-authenticated users.

PCMCIA cards

Small credit card-sized computing devices that comply with the Personal Computer Memory Card International Association (PCMCIA) standard. These devices, also called PC cards, are used for adding memory, modems, or as hardware security modules. PCMCIA cards that are used as hardware security modules securely store the private key component of a **public and private key pair** and some also perform the cryptographic operations as well.

peer identity

SSL connect sessions are between a particular client and a particular server. The identity of the peer may have been established as part of session setup. Peers are identified by **X.509 certificate chain**s.

PEM

The Internet Privacy-Enhanced Mail protocols standard, adopted by the Internet Architecture Board to provide secure electronic mail over the Internet. The PEM protocols provide for encryption, authentication, message integrity, and key management. PEM is an inclusive standard, intended to be compatible with a wide range of key-management approaches, including both symmetric and public-key schemes to encrypt data-encrypting keys. The specifications for PEM come from four Internet Engineering Task Force (IETF) documents: RFCs 1421, 1422, 1423, and 1424.

PKCS #10

An RSA Security, Inc., Public-Key Cryptography Standards (PKCS) specification that describes a syntax for certification requests. A certification request consists of a distinguished name, a public key, and optionally a set of attributes, collectively signed by the entity requesting certification. Certification requests are referred to as certificate requests in this manual. See **certificate request**

PKCS #11

An RSA Security, Inc., Public-Key Cryptography Standards (PKCS) specification that defines an application programming interface (API), called Cryptoki, to devices which hold cryptographic information and perform cryptographic operations. See **PCMCIA cards**

PKCS #12

An RSA Security, Inc., Public-Key Cryptography Standards (PKCS) specification that describes a transfer syntax for storing and transferring personal authentication credentials—typically in a format called a **wallet**.

PKI

See **public key infrastructure (PKI)**

plaintext

Message text that has not been encrypted.

principal

A string that uniquely identifies a client or server to which a set of Kerberos credentials is assigned. It generally has three parts: kservice/kinstance@REALM. In the case of a user, kservice is the user name. See also **kservice**, **kinstance**, and **realm**

private key

In public-key cryptography, this key is the secret key. It is primarily used for decryption but is also used for encryption with digital signatures. See **public and private key pair**

proxy authentication

A process typically employed in an environment with a middle tier such as a firewall, wherein the end user authenticates to the middle tier, which thence authenticates to the directory on the user's behalf—as its *proxy*. The middle tier logs into the directory as a *proxy user*. A proxy user can switch identities and, once logged into the directory, switch to the end user's identity. It can perform operations on the end user's behalf, using the authorization appropriate to that particular end user.

public key

In public-key cryptography, this key is made public to all. It is primarily used for encryption but can be used for verifying signatures. See **public and private key pair**

public key encryption

The process where the sender of a message encrypts the message with the public key of the recipient. Upon delivery, the message is decrypted by the recipient using its private key.

public key infrastructure (PKI)

Information security technology utilizing the principles of public key cryptography. Public key cryptography involves encrypting and decrypting information using a shared public and private key pair. Provides for secure, private communications within a public network.

public and private key pair

A set of two numbers used for **encryption** and **decryption**, where one is called the **private key** and the other is called the **public key**. Public keys are typically made widely available, while private keys are held by their respective owners. Though mathematically related, it is generally viewed as computationally infeasible to derive the private key from the public key. Public and private keys are used only with asymmetric encryption algorithms, also called public-key encryption algorithms, or public-key cryptosystems. Data encrypted with either a public key or a private key from a **key pair** can be decrypted with its associated key from the key-pair. However, data encrypted with a public key cannot be decrypted with the same public key, and data enwrapped with a private key cannot be decrypted with the same private key.

RADIUS

Remote Authentication Dial-In User Service (RADIUS) is a client/server protocol and software that enables remote access servers to communication with a central server to authenticate dial-in users and authorize their access to the requested system or service.

realm

1. Short for **identity management realm**. 2. A Kerberos object. A set of clients and servers operating under a single key distribution center/ticket-granting service (KDC/TGS). Services (see **kservice**) in different realms that share the same name are unique.

realm Oracle Context

An **Oracle Context** that is part of an **identity management realm** in Oracle Internet Directory.

registry

A Windows repository that stores configuration information for a computer.

remote computer

A computer on a network other than the local computer.

root key certificate

See **trusted certificate**

salt

1. In cryptography, generally speaking, "salt" is a way to strengthen the security of encrypted data. Salt is a random string that is added to the data before it is encrypted. Then, it is more difficult for attackers to steal the data by matching patterns of ciphertext to known ciphertext samples. 2. Salt is also used to avoid dictionary attacks, a method that unethical hackers (attackers) use to steal passwords. It is added to passwords before the passwords are encrypted. Then it is difficult for attackers to match the hash value of encrypted passwords (sometimes called verifiers) with their dictionary lists of common password hash values. See **dictionary attack**

schema

1. Database schema: A named collection of objects, such as tables, **views**, clusters, procedures, packages, **attribute**s, **object class**es, and their corresponding matching rules, which are associated with a particular user. 2. LDAP directory schema: The collection of attributes, object classes, and their corresponding matching rules.

schema mapping

See **user-schema mapping**

Secure Hash Algorithm (SHA)

An algorithm that assures data integrity by generating a 160-bit cryptographic message digest value from given data. If as little as a single bit in the data is modified, the Secure Hash Algorithm checksum for the data changes. Forgery of a given data set in a way that will cause the Secure Hash Algorithm to generate the same result as that for the original data is considered computationally infeasible.

An algorithm that takes a message of less than 264 bits in length and produces a 160-bit message digest. The algorithm is slightly slower than MD5, but the larger message digest makes it more secure against brute-force collision and inversion attacks.

Secure Sockets Layer (SSL)

An industry standard protocol designed by Netscape Communications Corporation for securing network connections. SSL provides authentication, encryption, and data integrity using public key infrastructure (PKI).

server

A provider of a service.

service

1. A network resource used by clients; for example, an Oracle database server.

2. An executable process installed in the Windows **registry** and administered by Windows. Once a service is created and started, it can run even when no user is logged on to the computer.

service name

For Kerberos-based authentication, the **kservice** portion of a service principal.

service principal

See **principal**

service table

In Kerberos authentication, a service table is a list of service principals that exist on a **kinstance**. This information must be extracted from Kerberos and copied to the Oracle server computer before Kerberos can be used by Oracle.

service ticket

Trusted information used to authenticate the client. A ticket-granting ticket, which is also known as the initial ticket, is obtained by directly or indirectly running okinit and providing a password, and is used by the client to ask for service tickets. A *service ticket* is used by a client to authenticate to a service.

session key

A key shared by at least two parties (usually a client and a server) that is used for data encryption for the duration of a single communication session. Session keys are typically used to encrypt network traffic; a client and a server can negotiate a session key at the beginning of a session, and that key is used to encrypt all network traffic between the parties for that session. If the client and server communicate again in a new session, they negotiate a new session key.

session layer

A network layer that provides the services needed by the presentation layer entities that enable them to organize and synchronize their dialogue and manage their data exchange. This layer establishes, manages, and terminates network sessions between the client and server. An example of a session layer is Network Session.

SHA

See **Secure Hash Algorithm (SHA)**

shared schema

A database or application schema that can be used by multiple enterprise users. Oracle Advanced Security supports the mapping of multiple enterprise users to the same shared schema on a database, which lets an administrator avoid creating an account for each user in every database. Instead, the administrator can create a user in one

location, the enterprise directory, and map the user to a shared schema that other enterprise users can also map to. Sometimes called **user/schema separation**.

single key-pair wallet

A **PKCS #12**-format **wallet** that contains a single user **certificate** and its associated **private key**. The **public key** is imbedded in the certificate.

single password authentication

The ability of a user to authenticate with multiple databases by using a single password. In the Oracle Advanced Security implementation, the password is stored in an LDAP-compliant directory and protected with encryption and Access Control Lists.

single sign-on (SSO)

The ability of a user to *authenticate once,* combined with strong authentication occurring transparently in subsequent connections to other databases or applications. Single sign-on lets a user access multiple accounts and applications with a single password, entered during a single connection. *Single password, single authentication.* Oracle Advanced Security supports Kerberos and SSL-based single sign-on.

smart card

A plastic card (like a credit card) with an embedded integrated circuit for storing information, including such information as user names and passwords, and also for performing computations associated with authentication exchanges. A smart card is read by a hardware device at any client or server.

A smartcard can generate random numbers which can be used as one-time use passwords. In this case, smartcards are synchronized with a service on the server so that the server expects the same password generated by the smart card.

sniffer

Device used to surreptitiously listen to or capture private data traffic from a network.

sqlnet.ora file

A configuration file for the client or server that specifies:

- Client domain to append to unqualified service names or net service names

- Order of naming methods the client should use when resolving a name

- Logging and tracing features to use

- Route of connections

- Preferred Oracle Names servers

- External naming parameters

- Oracle Advanced Security parameters

The `sqlnet.ora` file typically resides in *$ORACLE_HOME*/network/admin on UNIX platforms and *ORACLE_BASE**ORACLE_HOME*\network\admin on Windows platforms.

SSO

See **single sign-on (SSO)**

System Global Area (SGA)

A group of shared memory structures that contain data and control information for an Oracle **instance**.

system identifier (SID)

A unique name for an Oracle **instance**. To switch between Oracle databases, users must specify the desired SID. The SID is included in the CONNECT DATA parts of the **connect descriptor** in a **tnsnames.ora** file, and in the definition of the **network listener** in a **listener.ora file**.

ticket

A piece of information that helps identify who the owner is. See **service ticket**.

tnsnames.ora

A file that contains connect descriptors; each **connect descriptor** is mapped to a **net service name**. The file may be maintained centrally or locally, for use by all or individual clients. This file typically resides in the following locations depending on your platform:

- (UNIX) *ORACLE_HOME*/network/admin

- (Windows) *ORACLE_BASE\ORACLE_HOME*\network\admin

token card

A device for providing improved ease-of-use for users through several different mechanisms. Some token cards offer one-time passwords that are synchronized with an authentication service. The server can verify the password provided by the token card at any given time by contacting the authentication service. Other token cards operate on a challenge-response basis. In this case, the server offers a challenge (a number) which the user types into the token card. The token card then provides another number (cryptographically-derived from the challenge), which the user then offers to the server.

transport layer

A networking layer that maintains end-to-end reliability through data flow control and error recovery methods. **Oracle Net Services** uses *Oracle protocol supports* for the transport layer.

trusted certificate

A trusted certificate, sometimes called a root key certificate, is a third party identity that is qualified with a level of trust. The trusted certificate is used when an identity is being validated as the entity it claims to be. Typically, the certificate authorities you trust are called trusted certificates. If there are several levels of trusted certificates, a trusted certificate at a lower level in the certificate chain does not need to have all its higher level certificates reverified.

trusted certificate authority

See **certificate authority**

trust point

See **trusted certificate**

user name

A name that can connect to and access objects in a database.

user-schema mapping

An **LDAP** directory entry that contains a pair of values: the **base** in the directory at which users exist, and the name of the database schema to which they are mapped. The users referenced in the mapping are connected to the specified schema when they connect to the database. User-schema mapping entries can apply only to one database or they can apply to all databases in a domain. See **shared schema**

user/schema separation

See **shared schema**

user search base

The node in the LDAP directory under which the user resides.

views

Selective presentations of one or more tables (or other views), showing both their structure and their data.

wallet

A wallet is a data structure used to store and manage security credentials for an individual entity. A **Wallet Resource Locator** (WRL) provides all the necessary information to locate the wallet.

wallet obfuscation

Wallet **obfuscation** is used to store and access an Oracle **wallet** without querying the user for a password prior to access (supports **single sign-on (SSO)**).

Wallet Resource Locator

A wallet resource locator (WRL) provides all necessary information to locate a **wallet**. It is a path to an operating system directory that contains a wallet.

Windows native authentication

An **authentication method** that enables a client single login access to a Windows server and a database running on that server.

WRL

See **Wallet Resource Locator**

X.509

An industry-standard specification for digital **certificate**s.

Index