Oracle® Database

Advanced Security Administrator's Guide

11*g* Release 2 (11.2)

E10746-07

April 2013

ORACLE

Oracle Database Advanced Security Administrator's Guide 11*g* Release 2 (11.2)

E10746-07

Primary Author: Sumit Jeloka

Contributor: Peter Wahl, Rahil Mir, Indra Fitzgerald, Paul Youn, Adam Lee, Preetam Ramakrishna, Gopal Mulagund, Daniel Wong, 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, Paul Needham

Contents

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

5 Configuring Network Authentication, 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

G Entrust-Enabled SSL Authentication

List of Figures

List of Tables

Preface

Welcome to the *Oracle Database Advanced Security Administrator's Guide* for the 11*g* Release 2 (11.2) 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

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

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

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, "Securing Stored Data Using Transparent Data Encryption"

This chapter provides an overview of the transparent data encryption feature introduced in Oracle Advanced Security 11*g* Release 2 (11.2). 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 11*g* Release 2 (11.2) network.

Chapter 5, "Configuring Network Authentication, 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 User'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.

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/technetwork/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/technetwork/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

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in Oracle Advanced Security?

This section describes new features of Oracle Advanced Security 11*g* Release 2 (11.2) and provides pointers to additional information.

Oracle Database 11*g* Release 2 (11.2.0.3) New Features in Oracle Advanced Security

This release includes the following new features:

- Support for SHA-2 Certificate Signatures

  This feature introduces support for SHA-2 (256-bit) signed certificates that are used by the database for network encryption and authentication.

  These certificates are issued by a separate certificate authority (CA), and are exchanged between the database and a client when a secure database connection is being established.

- Support for PIN and Multiple Certificates on Smart Card

  This feature introduces support for authenticating to the database using Common Access Cards (CAC, HSPD-12) that contain multiple certificates.

  When a database user inserts a card containing one or more digital certificates into a card reader, the database attempts to intelligently select which certificate to read. If the database cannot determine which certificate to read, a selection box is presented on Windows clients. The user also must manually enter the correct PIN.

- TDE Hardware Acceleration for Solaris

  Transparent Data Encryption (TDE) can automatically detect whether the database host machine includes specialized cryptographic silicon that accelerates the encryption and decryption processing. When detected, TDE uses the specialized silicon for cryptographic processing, accelerating the overall cryptographic performance significantly.

  In prior releases, cryptographic hardware acceleration for TDE was only available on Intel Xeon, and only for Linux. Starting with release 11.2.0.3, it works with the current versions of Solaris 11 running on both SPARC T-Series and Intel Xeon.

Oracle Database 11*g* Release 2 (11.2) New Features in Oracle Advanced Security

This release includes the following new features:

- Enhanced TDE Tablespace Encryption

Oracle Database 11*g* Release 2 (11.2) implements the following enhancements to TDE Tablespace Encryption:

- A unified master encryption key is used for both Transparent Data Encryption (TDE) Column Encryption and TDE Tablespace Encryption.
- The unified master encryption key can optionally be stored in a hardware security module. This enables you to use the TDE Tablespace Encryption feature along with hardware security modules.
- You can reset (`rekey`) the unified master encryption key. This provides enhanced security and helps meet security and compliance requirements.

  See Also: "Encrypting Entire Tablespaces" on page 3-16

- TDE Supports Intel Advanced Encryption Standard New Instructions (Intel AES-NI)

  Transparent Data Encryption (TDE) now supports Intel AES-NI. Oracle Database 11*g* Release 2 (11.2) running on Intel Xeon 5600 series processor-based servers with Intel AES-NI shows a multifold increase in TDE encryption and decryption speed.

  According to benchmark results, TDE shows a 10x speedup of AES encryption processing rate and an 8x speedup of decryption processing rate, using 256 bit keys, on Intel Xeon X5680 processor utilizing AES-NI as compared to Intel Xeon X5560 processor without AES-NI.

- Internet Protocol Version 6 (IPv6) Support

  Oracle Advanced Security fully supports Internet Protocol Version 6 (IPv6) networks.

- Kerberos Enhancements

  The Oracle Kerberos authentication mechanism now supports the Microsoft Windows Server 2003 constrained delegation feature. The middle tier can use the Kerberos adapter to authenticate to the Oracle Database without providing the user's forwarded Kerberos credentials.

  A user can authenticate to the middle tier using a non-Kerberos authentication mechanism. The middle tier authenticates to the backend Oracle Database using the Kerberos authentication mechanism on behalf of the user.

  See Also: Microsoft documentation for more information on the Microsoft Windows Server 2003 constrained delegation feature

Oracle Database 11*g* Release 1 (11.1) New Features in Oracle Advanced Security

This release includes the following new features:

- Enhanced Transparent Data Encryption

  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.

You can now encrypt entire tablespaces using Tablespace Encryption. All objects created in the encrypted tablespace are automatically encrypted. See "TDE Tablespace Encryption" in on page 3-3 for more information.

Transparent Data Encryption now enables you to use a hardware security module (HSM) to store the master encryption key. This allows for enhanced security. See "Using Hardware Security Modules with TDE" on page 3-20 for more information.

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

- Kerberos authentication is more secure and manageable

  The Kerberos implementation now makes use of secure encryption algorithms like `3DES` and `AES` in place of `DES`. This makes using Kerberos more secure. The Kerberos authentication mechanism in Oracle Database now supports the following encryption types:

  – `DES3-CBC-SHA` (`DES3` algorithm in `CBC` mode with `HMAC-SHA1` as checksum)

  – `RC4-HMAC` (`RC4` algorithm with `HMAC-MD5` as checksum)

  – `AES128-CTS` (`AES` algorithm with 128-bit key in `CTS` mode with `HMAC-SHA1` as checksum)

  – `AES256-CTS` (`AES` algorithm with 256-bit key in `CTS` mode with `HMAC-SHA1` as checksum)

  The Kerberos implementation has been enhanced to interoperate smoothly with Microsoft and MIT Key Distribution Centers.

  The Kerberos prinicipal name can now contain more than 30 characters. It is no longer restricted by the number of characters allowed in a database user name.

  > **See Also:** Chapter 7, "Configuring Kerberos Authentication"

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.

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:

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

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 rest and 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

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:

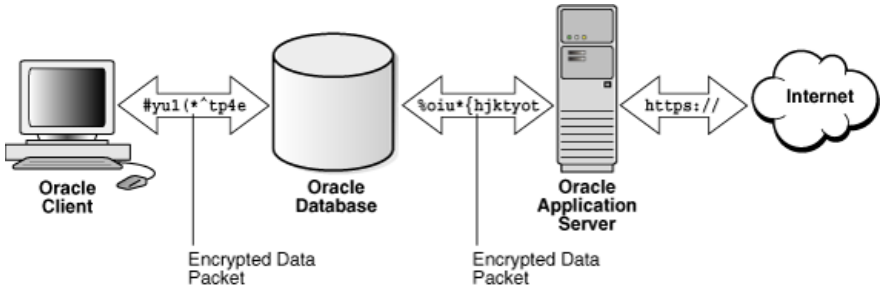
- Data Encryption
- Strong Authentication

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 11*g* Release 2 (11.2) 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 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".

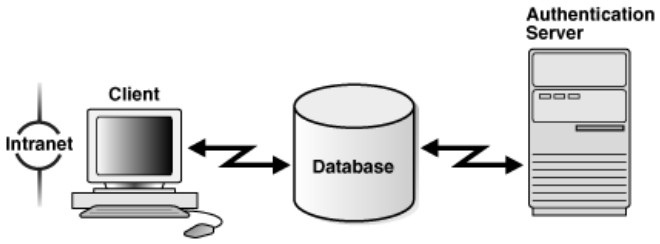
The cryptographic libraries for SSL included in Oracle Database 10g have been validated under FIPS 140-2 at the Level 2 security level. FIPS 140-2 related configuration settings are described in Appendix E, "Oracle Advanced Security FIPS 140-2 Settings".

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:

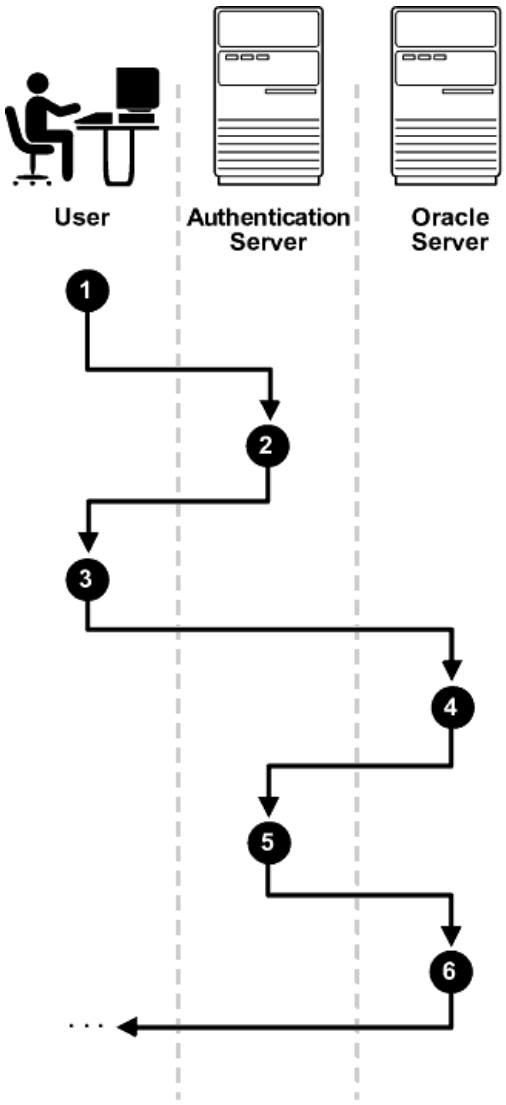
- Centralized Authentication and Single Sign-On
- Supported Authentication Methods

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. The authentication server checks the credentials and notifies the Oracle server.
6. If the credentials were accepted by the authentication server, then the Oracle server authenticates the user. If the authentication server rejected 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

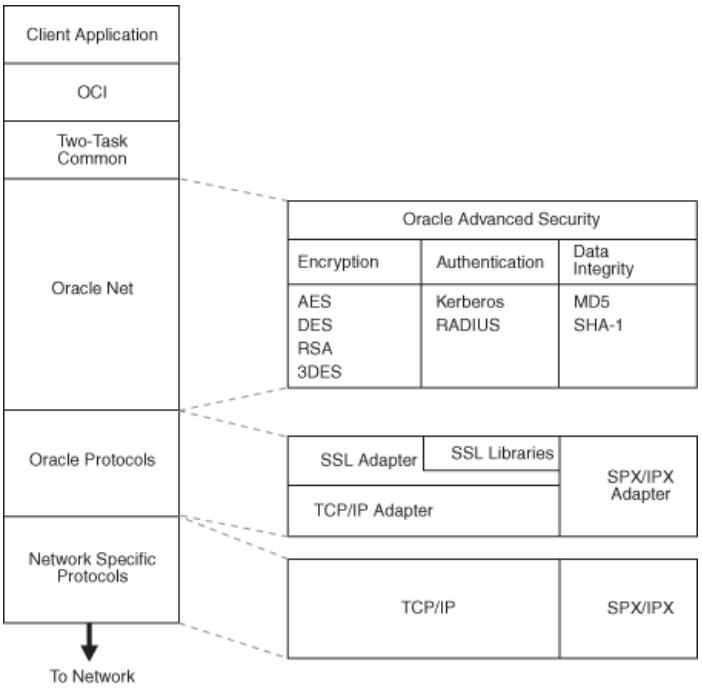
Entrust/PKI : Oracle Advanced Security supports the public key infrastructure provided by the Entrust / PKI software from Entrust Technologies, Inc. Entrust-enabled Oracle Advanced Security lets Entrust users incorporate Entrust single sign-on into their Oracle applications, and it lets Oracle users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

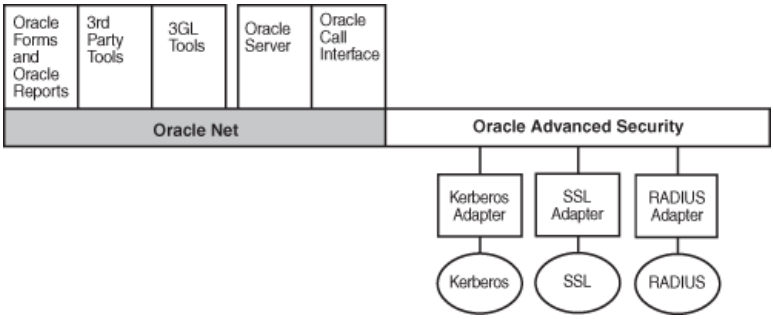
Oracle Advanced Security complements an Oracle server or client installation with advanced security features. Figure 1–4 shows the Oracle Advanced Security architecture within an Oracle networking environment.

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

System Requirements

Oracle Advanced Security 11*g* Release 2 (11.2) requires Oracle Net 11*g* Release 2 (11.2) 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 or above. ■ 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:

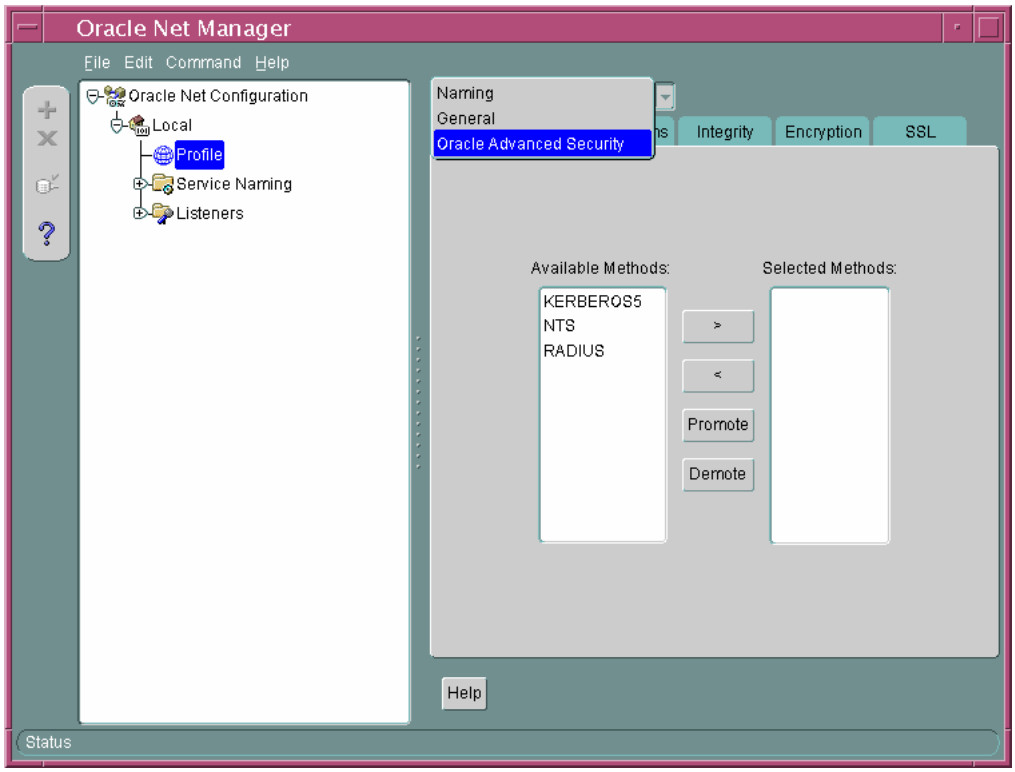
  ```
  netmgr
  ```

- (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 suites** 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 pairs**
- 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:

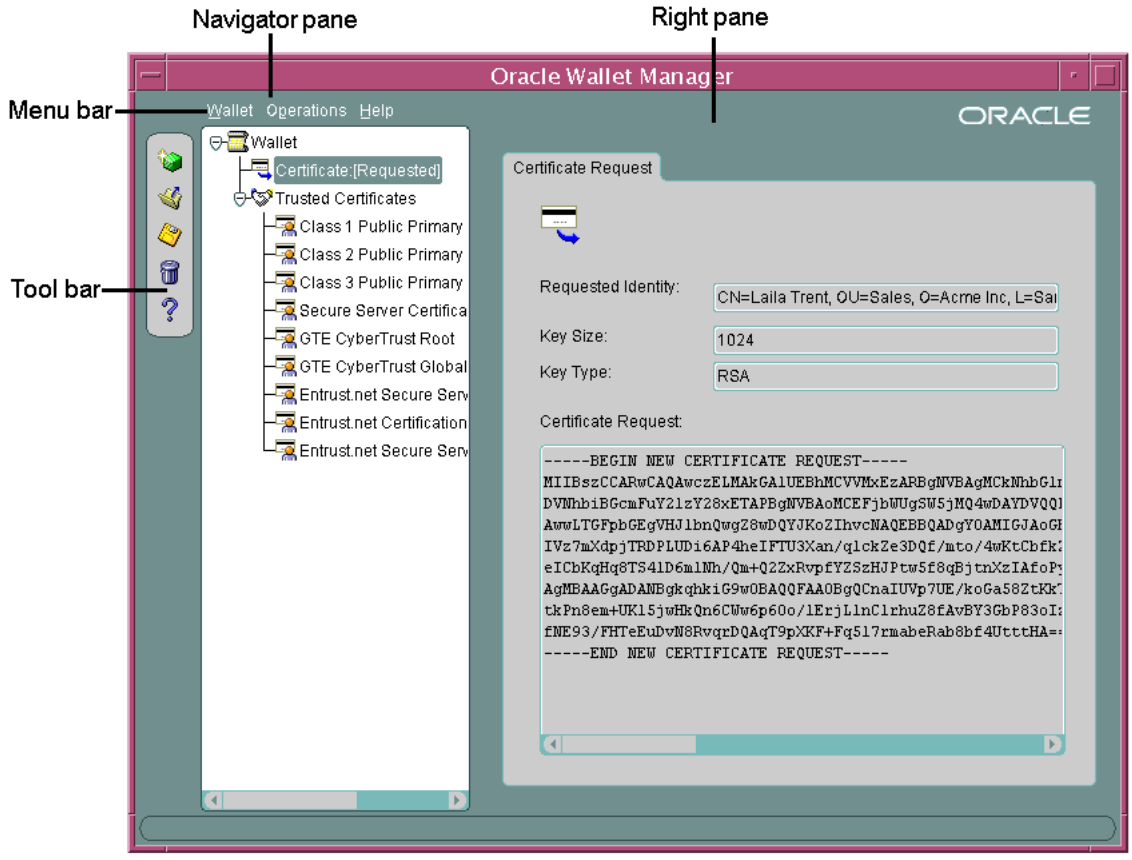
  ```
  owm
  ```

- (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.

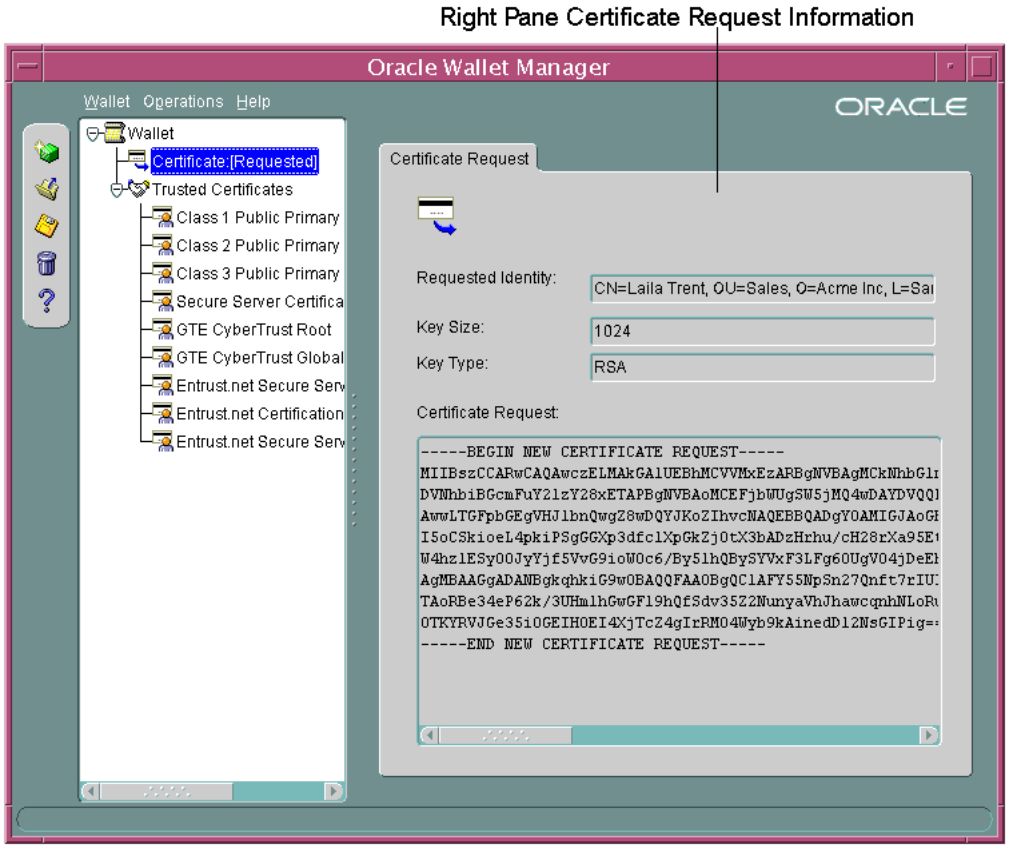
[1] These objects display only after you create a wallet, generate a certificate request, and import a certificate into the wallet.

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 (**CRL**s), 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.example.com` and that uses port 389:

```
orapki crl list -ldap machine1.us.example.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 6-12
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 7-8
Manage Kerberos credentials in the credential cache	■ `okinit` ■ `oklist` ■ `okdstry`	■ "Obtaining the Initial Ticket with the okinit Utility" on page 7-9 ■ "Displaying Credentials with the oklist Utility" on page 7-9 ■ "Removing Credentials from the Cache File with the okdstry Utility" on page 7-10
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:

3

Securing Stored Data Using Transparent Data Encryption

Transparent Data Encryption(TDE) enables you to encrypt sensitive data, such as credit card numbers, stored in tables and tablespaces. Encrypted data is transparently decrypted for a database user or application that has access to data. TDE helps protect data stored on media in the event that the storage media or data file gets stolen.

This chapter is divided into the following topics:

About Transparent Data Encryption

Oracle Database uses authentication, authorization, and auditing mechanisms to secure data in the database, but not in the operating system data files where data is stored. To protect these data files, Oracle Database provides Transparent Data Encryption (TDE). TDE encrypts sensitive data stored in data files. To prevent unauthorized decryption, TDE stores the encryption keys in a security module external to the database.

Database users and applications do not need to manage key storage or create auxiliary tables, views, and triggers. An application that processes sensitive data can use TDE to provide strong data encryption with little or no change to the application.

Use TDE to protect confidential data, such as credit card and social security numbers, stored in table columns. You can also use TDE to encrypt entire tablespaces.

This section contains the following topics:

Benefits of Using Transparent Data Encryption

Transparent Data Encryption (TDE) has the following advantages:

- As a security administrator, you can be sure that sensitive data is safe in case the storage media or data file gets stolen.
- Implementing TDE helps you address security-related regulatory compliance issues.
- You do not need to create triggers or views to decrypt data for the authorized user or application. Data from tables is transparently decrypted for the database user and application.
- Database users and applications need not be aware of the fact that the data they are accessing is stored in encrypted form. Data is transparently decrypted for the database users and applications.
- Applications need not be modified to handle encrypted data. Data encryption and decryption is managed by the database.
- Key management operations are automated. The user or application does not need to manage encryption keys.

Types of Transparent Data Encryption

Transparent Data Encryption (TDE) column encryption enables you to encrypt sensitive data stored in select table columns. TDE tablespace encryption enables you to encrypt all data stored in a tablespace.

Both TDE column encryption and TDE tablespace encryption use a two-tiered, key-based architecture. 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.

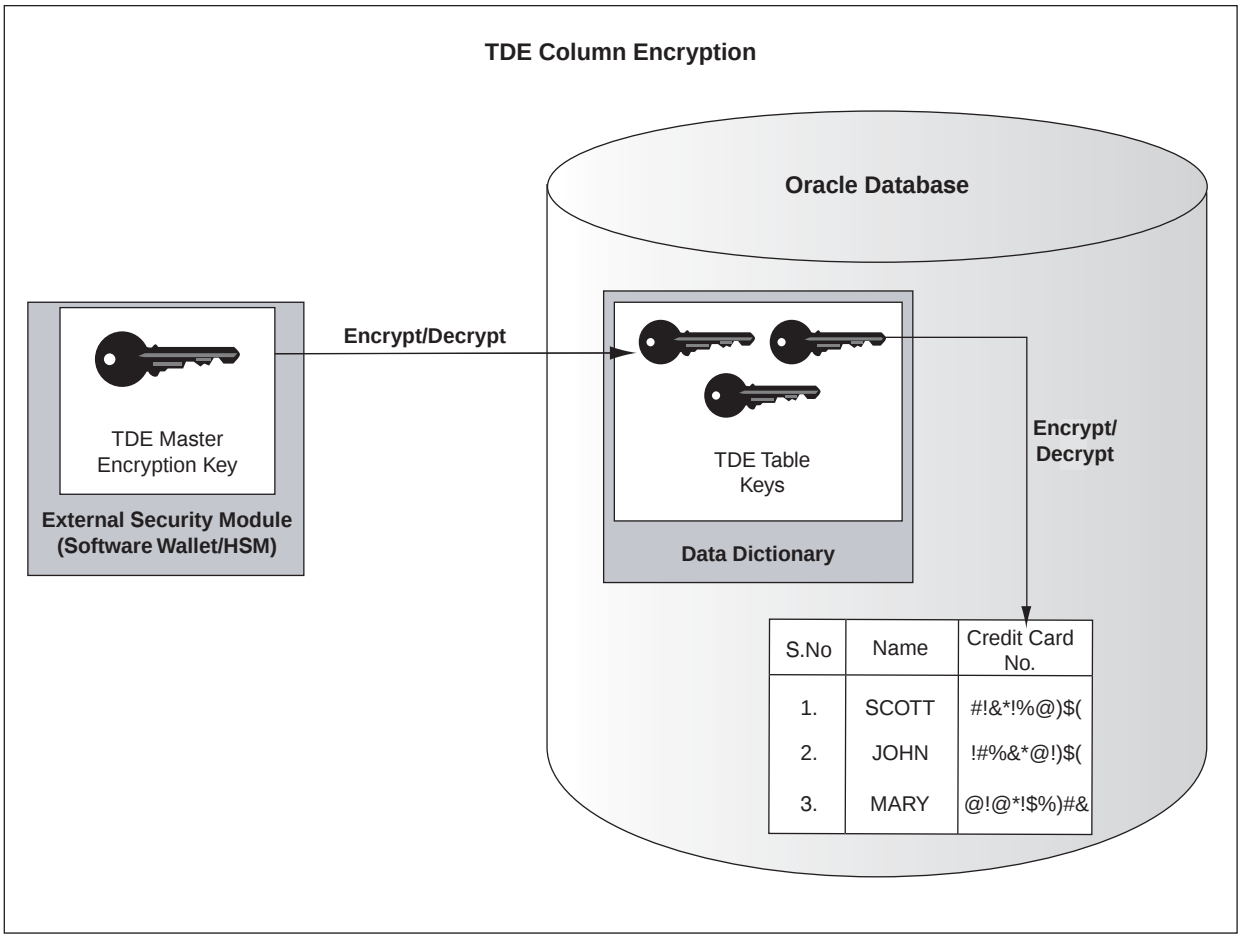
The following sections discuss TDE column encryption and TDE tablespace encryption:

- TDE Column Encryption
- TDE Tablespace Encryption

TDE Column Encryption

TDE column encryption is used to protect confidential data, such as credit card and social security numbers, stored in table columns. TDE column encryption uses the two-tiered, key-based architecture to transparently encrypt and decrypt sensitive table columns. The TDE master encryption key is stored in an external security module, which can be an Oracle wallet or Hardware Security Module (HSM). This master encryption key is used to encrypt the table key, which in turn is used to encrypt and decrypt data in the table column. Figure 3–1shows an overview of the TDE column encryption process.

Figure 3–1 TDE Column Encryption Overview



As shown in Figure 3–1, the master encryption key 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 or Hardware Security Module (HSM), as described in this chapter. Storing the master encryption key in this way prevents its unauthorized use.

Using an external security module (wallet / HSM) separates ordinary program functions from encryption operations, making it possible to divide duties between database administrators and security administrators. Security is enhanced because the wallet password can be unknown to the database administrator, requiring the security administrator to provide the password.

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

TDE Tablespace Encryption

TDE tablespace encryption enables you to encrypt an entire tablespace. All objects created in the encrypted tablespace are automatically encrypted. TDE tablespace encryption is useful if you want to secure sensitive data in tables. You do not need to perform a granular analysis of each table column to determine the columns that need encryption.

In addition, TDE tablespace encryption takes advantage of bulk encryption and caching to provide enhanced performance. While the actual performance impact on applications can vary, the performance overhead is roughly estimated to be in between 5% and 8%.

TDE tablespace encryption is a good alternative to TDE column encryption if your tables contain sensitive data in multiple columns, or if you want to protect the entire table and not just individual columns.

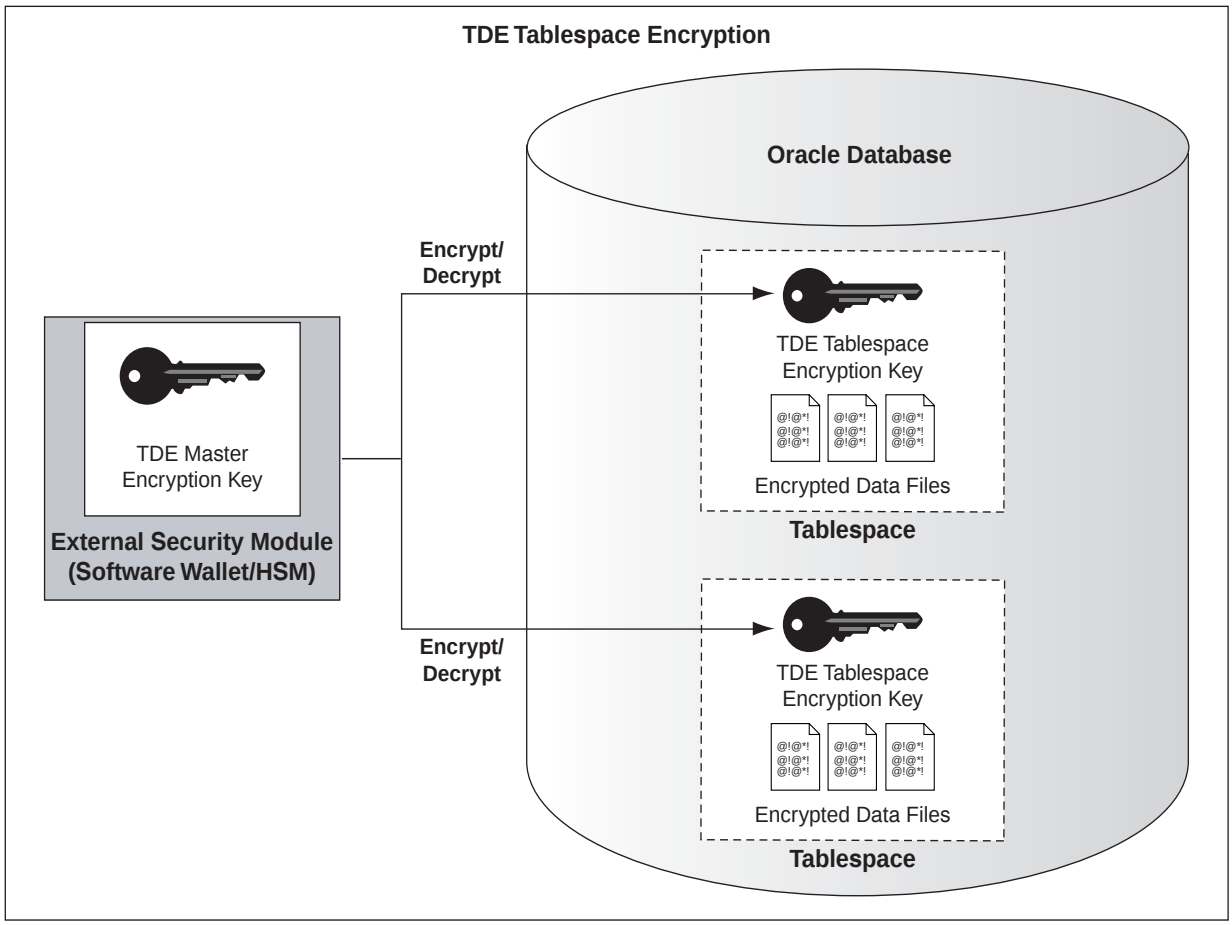
TDE tablespace encryption encrypts all data stored in an encrypted tablespace. This includes internal large objects (`LOB`s) such as `BLOB`s and `CLOB`s. TDE tablespace encryption does not encrypt data that is stored outside the tablespace. For example, `BFILE` data is not encrypted as it is stored outside the database. If you create a table with a `BFILE` column in an encrypted tablespace, then this particular column will not be encrypted. However, SecureFile LOBs are supported from Oracle Database 11*g* Release 1 (11.1).

All data in an encrypted tablespace is stored in encrypted format on the disk. Data is transparently decrypted for an authorized user having the necessary privileges to view or modify the data. A database user or application does not need to know if the data in a particular table is encrypted on the disk. In the event that the data files on a disk or backup media gets stolen, the data is not compromised.

TDE tablespace encryption uses the two-tiered, key-based architecture to transparently encrypt (and decrypt) tablespaces. The TDE master key is stored in an external security module (Oracle Wallet or HSM). This TDE master key is used to encrypt the TDE tablespace encryption key, which in turn is used to encrypt and decrypt data in the tablespace.

Figure 3–2 shows an overview of the TDE tablespace encryption process.

Figure 3–2 TDE Tablespace Encryption



Note: The encrypted data is protected during operations like `JOIN` and `SORT`. This means that the data is safe when it is moved to temporary tablespaces. Data in undo and redo logs is also protected.

TDE tablespace encryption also allows index range scans on data in encrypted tablespaces. This is not possible with TDE column encryption.

Oracle Database 11*g* Release 2 (11.2) implements the following enhancements to TDE tablespace encryption:

- A unified master encryption key is used for both TDE column encryption and TDE tablespace encryption.
- You can reset the unified master encryption key. This provides enhanced security and helps meet security and compliance requirements.

Using Transparent Data Encryption

The following sections discuss using Transparent Data Encryption (TDE):

- Enabling Transparent Data Encryption
- Setting and Resetting the Master Encryption Key
- Opening and Closing the Encrypted Wallet

- Encrypting Columns in Tables
- Encrypting Entire Tablespaces
- Using Hardware Security Modules with TDE
- Using Transparent Data Encryption with Oracle RAC

Enabling Transparent Data Encryption

TDE column encryption was first introduced in Oracle Database 10*g* release 2 (10.2). To use this feature, you must be running Oracle Database 10*g* release 2 (10.2) or higher.

TDE tablespace encryption was introduced in Oracle Database 11g release 1 (11.1). To use this feature, you must be running Oracle Database 11g release 1 (11.1) or higher.

Note: Oracle Database 11*g* Release 1 (11.1) and higher versions ensure greater security by protecting data in temporary tablespaces during operations such as `JOIN` and `SORT`. The data in temporary tablespaces stays encrypted during these operations.

To start using TDE, 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 TDE. Oracle strongly recommends that you use a separate wallet to store the master encryption key.

Specifying a Wallet Location for Transparent Data Encryption

If you wish to use a wallet specifically for TDE, then you must specify a wallet location in the `sqlnet.ora` file by using the `ENCRYPTION_WALLET_LOCATION` parameter. Oracle recommends that you use the `ENCRYPTION_WALLET_LOCATION` parameter to specify a wallet location for TDE.

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

Using Wallets with Automatic Login Enabled

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 auto login wallet.

You can also choose to create a local auto login wallet. Local auto login wallets cannot be moved to another computer. They must be used on the host on which they are created.

> **See Also:**
>
> "Using an Auto Login Wallet" on page 3-25 for more information on auto login wallets.

Setting and Resetting the Master Encryption Key

The master encryption key is stored in an external security module, and it is used to protect the table keys and tablespace encryption keys. By default, the master encryption key is a random key generated by Transparent Data Encryption (TDE). It can also be an existing key pair from a PKI certificate designated for encryption. To use

TDE with PKI key pairs, the issuing certificate authority must be able to issue X.509v3 certificates with the key usage field marked for encryption.

> **Note:** PKI-based encryption does not work with TDE tablespace encryption or hardware security modules. To know more about hardware security modules, refer to "Using Hardware Security Modules with TDE" on page 3-20.

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 encryption key may result in greater performance degradation when accessing encrypted columns in the database.

Use the `ALTER SYSTEM` command to set or reset (`rekey`) the master encryption key. The following sections discuss setting and resetting the master encryption key.

Setting the Master Encryption Key

Before you can encrypt or decrypt database columns or tablespaces, you must generate a master encryption key. Oracle Database 11*g* Release 2 (11.2) uses the same master encryption key for both TDE column encryption and TDE tablespace encryption.

To set the master encryption key, use the following command:

```
SQL> 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 Oracle wallet. Use this parameter if you intend to use your PKI private key as your master encryption key. This parameter has no default setting. Enclose the certificate_ID in double quotation marks (" ").

  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 encryption keys by TDE are shown.

- *`password`* is the mandatory wallet password for the security module, with no default setting. It is case sensitive. Enclose the password string in double quotation marks (" ").

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

The wallet location specified by the `ENCRYPTION_WALLET_LOCATION` parameter, in the `sqlnet.ora` parameter file, is used to create the master encryption key. If the `ENCRYPTION_WALLET_LOCATION` parameter is not present in the `sqlnet.ora` file, then the `WALLET_LOCATION` value is used. A new wallet is created if one does not exist already.

If no wallet location is specified in the `sqlnet.ora` file, then the default database wallet location is used. The default database wallet location is *`ORACLE_BASE`*`/admin/`*`DB_UNIQUE_NAME`*`/wallet` or *`ORACLE_HOME`*`/admin/`*`DB_UNIQUE_NAME`*`/wallet`. Here, *`DB_UNIQUE_NAME`* is the unique name of the database specified in the initialization parameter file.

If an existing auto login wallet is present at the expected wallet location, then a new wallet is not created.

Resetting the Master Encryption Key

Reset / Regenerate the master encryption key only if it has been compromised or as per the security policies of the organization. You should back up the wallet before resetting the master encryption key.

Frequent master encryption key regeneration does not necessarily enhance system security. Security modules can store a large number of keys. However, this number is not unlimited. Frequent master encryption key regeneration can exhaust all available storage space.

To reset the master encryption key, use the SQL syntax as shown in "Setting the Master Encryption Key" on page 3-7.

> **Note:** If you are resetting the master encryption key for a wallet that has auto login enabled, then you must ensure that both the auto login wallet, identified by the `.sso` file, and the encryption wallet, identified by the `.p12` file, are present before issuing the command to reset the master encryption key.

The `ALTER SYSTEM SET ENCRYPTION KEY` command is a data definition language (DDL) command requiring the `ALTER SYSTEM` privilege, and it automatically commits any pending transactions. Example 3–1 shows a sample usage of this command.

Example 3–1 Setting or Resetting the Master Encryption Key To Use a PKI-Based Private Key

```
SQL> ALTER SYSTEM SET ENCRYPTION KEY "j23lm781098dhb345sm" IDENTIFIED BY
"password";
```

Here, `j23lm781098dhb345sm` is the certificate ID and *`password`* is the wallet password.

For PKI-based keys, **certificate revocation lists** are not enforced as enforcing certificate revocation may lead to losing access to all encrypted information in the database. However, you cannot use the same certificate to create the master key again.

Opening and Closing the Encrypted Wallet

The database must load the master encryption key into memory before it can encrypt or decrypt columns / tablespaces. Opening the wallet allows the database to access the master encryption key. Use the following `ALTER SYSTEM` command to explicitly open the wallet:

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "password"
```

where *`password`* is the password to open the wallet. Enclose the password string in double quotation marks (" ").

> **Note:** The password to open the wallet is the password that you specify for creating the master encryption key. This is discussed under "Setting the Master Encryption Key" on page 3-7.

Once the wallet has been opened, it remains open until you shut down the database instance, or close it explicitly by issuing the following command:

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET CLOSE IDENTIFIED BY "password"
```

Closing the wallet disables all encryption and decryption operations. Any attempt to encrypt / decrypt data or access encrypted data results in the following error:

```
ORA-28365: wallet is not open
```

Each time you restart a database instance, you must issue the `ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "password"` command to reenable encryption and decryption operations.

Note: Auto login wallets are opened automatically and do not need to be opened explicitly.

In case an auto login wallet needs to be closed, it can be closed with the following command:

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET CLOSE
```

No password is required to close an auto login wallet.

If the user 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 3–2 shows an example of each usage case.

Example 3–2 Opening the External Security Module Wallet with ALTER SYSTEM

```
SQL> --Successfully opening the wallet
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "U83j10LLt8v";
Wallet opened.

SQL> --Trying to open a wallet that is already open
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "U83j10LLt8v";
ERROR at line 1:
ORA-28354: wallet already open

SQL> --Trying to open the wallet with an incorrect password
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "U93j10LLt8v";
ERROR at line 1:
ORA-28353: failed to open wallet
```

Encrypting Columns in Tables

The following sections discuss using TDE column encryption:

- Creating Tables with Encrypted Columns
- Encrypting Columns in Existing Tables
- Creating an Index on an Encrypted Column
- Adding or Removing Salt from an Encrypted Column
- Changing the Encryption Key or Algorithm for Tables Containing Encrypted Columns
- Data Types That Can Be Encrypted with TDE Column Encryption

- Restrictions on Using TDE Column Encryption

Creating Tables with 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
- Creating a Table with an Encrypted Column Using a Nondefault Algorithm and No Salt
- Using the NOMAC Parameter to Save Disk Space and Improve Performance
- Creating an Encrypted Column in an External Table

Creating a Table with an Encrypted Column By default, TDE uses the AES encryption algorithm with a 192-bit key length (AES192). If you encrypt a table column without specifying an algorithm, the column is encrypted using the AES192 algorithm.

TDE adds **salt** to cleartext before encrypting it. This makes it harder for attackers to steal data through a brute force attack. TDE also adds a Message Authentication Code (MAC) to the data for integrity checking. The SHA-1 integrity algorithm is used by default.

Note: If there are multiple encrypted columns in a table, then all these columns must use the same pair of encryption and integrity algorithms.

Salt is specified at the column level. This means that an encrypted column in a table can choose not to use salt irrespective of whether other encrypted columns in the table use salt or not.

Example 3–3 creates a new table with an encrypted column. The column is encrypted using the default encryption algorithm (AES192). Salt and MAC are added by default.

Example 3–3 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 Nondefault Algorithm and No Salt By default, TDE adds **salt** to cleartext before encrypting it. This makes it harder for attackers to steal data through a brute force attack. However, if you plan to index the encrypted column, you must use NO SALT.

TDE also enables you to specify a nondefault encryption algorithm. You can choose from one of the following algorithms:

- 3DES168
- AES128
- AES192 (default)

- AES256

Example 3–4 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 (' ').

The empID and salary columns will both use the 3DES168 encryption algorithm. This is because all encrypted columns in a table must use the same encryption algorithm. The salary column will use salt by default. The empID column will not use salt as the NO SALT option has been specified for it.

Example 3–4 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'
);
```

Using the NOMAC Parameter to Save Disk Space and Improve Performance The NOMAC parameter enables you to skip the integrity check performed by TDE. This saves 20 bytes of disk space per encrypted value. If the number of rows and encrypted columns in the table is large, then this adds up to a significant amount of disk space.

The NOMAC parameter also reduces the performance overheads associated with TDE. Using the NOMAC parameter causes the integrity check to be skipped during encryption and decryption operations. This saves processing cycles and leads to faster performance.

Note: TDE uses the SHA-1 integrity algorithm by default. All encrypted columns in a table must use the same integrity algorithm. If you already have a table column using the SHA-1 algorithm, then you cannot use the NOMAC parameter to encrypt another column in the same table.

You can change the integrity algorithm used by all encrypted columns in a table using the ALTER TABLE. . . . REKEY. . . command. See Example 3–6 for an example.

Example 3–5 creates a table with an encrypted column. The empID column is encrypted using the NOMAC parameter.

Example 3–5 Using the NOMAC parameter in a CREATE TABLE statement

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

Example 3–6 shows how to change the integrity algorithm for encrypted columns in a table. The encryption algorithm is set to 3DES168 and the integrity algorithm is set to SHA-1. The second ALTER TABLE statement sets the integrity algorithm to NOMAC.

Example 3–6 Changing the Integrity Algorithm for a Table

```
SQL> ALTER TABLE EMPLOYEE REKEY USING '3DES168' 'SHA-1';

Table altered.
SQL> ALTER TABLE EMPLOYEE REKEY USING '3DES168' 'NOMAC';

Table altered.
```

Creating an Encrypted Column in an External Table The external table feature enables you to access data in external sources as if the data were in a database table. External tables can be updated using the ORACLE_DATAPUMP access driver.

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

To encrypt specific columns in an external table, use the ENCRYPT clause when defining those columns. A system generated key is used to encrypt the columns. For example, the following definition encrypts the ssn column using the 3DES168 algorithm:

```
CREATE TABLE emp_ext (
    first_name,
    ....
    ssn ENCRYPT USING '3DES168',
    ....
...
...
```

If you plan to move your external table to a new location, then you cannot use a randomly generated key to encrypt the columns. This is because the randomly generated key will not be available at the new location.

For such scenarios, you should specify a password while encrypting the columns. After you move the data, you can use the same password to regenerate the key required to access encrypted column data at the new location.

Table partition exchange also requires a password-based table key.

Example 3–7 creates an external table using a password to create the table key.

Example 3–7 Creating a New External Table with a Password-Generated Table 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 Language Reference* about CREATE TABLE, ENCRYPT, and the rules for passwords.

Encrypting Columns 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 an Encrypted Column to an Existing Table
- Encrypting an Unencrypted Column
- Disabling Encryption on a Column

Adding an Encrypted Column to an Existing Table To add an encrypted column to an existing table, you use the ALTER TABLE ADD command, specifying the new column with the ENCRYPT clause. Example 3–8 adds an encrypted column, ssn, to an existing table, called employee.

Example 3–8 Adding an Encrypted Column to an Existing Table

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

The ssn column is encrypted with the default AES192 algorithm. Salt and MAC are added by default.

You can choose to encrypt the column using a different algorithm. You can also specify NO SALT, if you wish to index the column.

Encrypting an Unencrypted Column To encrypt an unencrypted column, use the ALTER TABLE MODIFY command, specifying the unencrypted column with the ENCRYPT clause. Example 3–9 encrypts the first_name column in the employee table.

Example 3–9 Encrypting an Unencrypted Column

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

The first_name column is encrypted with the default AES192 algorithm. Salt is added to the data, by default.

You can choose to encrypt the column using a different algorithm. You can also specify NO SALT, if you wish to index the column. You can also choose to skip integrity checks by using the NOMAC parameter. Example 3–10 encrypts the first_name column in the employee table using the NOMAC parameter.

Example 3–10 Using the NOMAC parameter in an ALTER TABLE statement

```
SQL> ALTER TABLE employee MODIFY (first_name ENCRYPT 'NOMAC');
```

Disabling Encryption on a Column You may want to disable encryption for reasons of compatibility or performance. To disable column encryption, use the ALTER TABLE MODIFY command with the DECRYPT clause. Example 3–11 decrypts the first_name column in the employee table.

Example 3–11 Turning Off Column Encryption

```
SQL> 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 3–12 shows how to create an index on a column that has been encrypted without salt.

Example 3–12 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.

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. This ensures that the same plaintext data does not always translate to the same encrypted text. Salt removes the one common method attackers use to steal data, namely, matching patterns of encrypted text. Adding salt requires an additional 16 bytes of storage, per encrypted data value.

To add or remove salt from encrypted columns, use the `ALTER TABLE MODIFY` command. Example 3–13 encrypts the `first_name` column using salt. If the `first_name` column was encrypted without salt earlier, then this command reencrypts it using salt.

Example 3–13 Adding Salt to an Encrypted Column

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

Example 3–14 removes salt from the `first_name` column. If you need to index a column that was encrypted using salt, then you can use this command to remove the salt before indexing.

Example 3–14 Removing Salt from an Encrypted Column

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

Changing the Encryption Key or Algorithm for Tables Containing Encrypted Columns

Each table can have only one table key for its columns. You can regenerate the table key with the `ALTER TABLE` command. You can also choose to use a different encryption algorithm for the new table key.

Example 3–15 regenerates the table key for the `employee` table.

Example 3–15 Changing the Encryption Key on Tables Containing Encrypted Columns

```
SQL> ALTER TABLE employee REKEY;
```

Example 3–16 regenerates the table key for the `employee` table using the `3DES168` algorithm.

Example 3–16 Changing the Encryption Key and Algorithm on Tables Containing Encrypted Columns

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

Data Types That Can Be Encrypted with TDE Column Encryption

The following data types can be encrypted using this feature:

- BINARY_DOUBLE
- BINARY_FLOAT
- CHAR
- DATE
- INTERVAL DAY TO SECOND
- INTERVAL YEAR TO MONTH
- LOBs (Internal LOBs and SECUREFILE LOBs Only)
- NCHAR
- NUMBER
- NVARCHAR2
- RAW
- TIMESTAMP (includes TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE)
- VARCHAR2

You cannot encrypt a column if the encrypted column size becomes greater than the size allowed by the data type of the column. Table 3–1 shows the maximum allowable sizes for various data types.

Table 3–1 Maximum Allowable Size for Data Types

Data Type	Maximum Size
CHAR	1932 bytes
VARCHAR2	3932 bytes
NVARCHAR2	1966 bytes
NCHAR	966 bytes

Note: TDE tablespace encryption does not have these data type restrictions.

Restrictions on Using TDE Column Encryption

TDE column encryption encrypts and decrypts data at the SQL layer. Oracle Database utilities and features that bypass the SQL layer cannot leverage the services provided by TDE column encryption. Do not use TDE column encryption with the following database features:

- Index types other than B-tree
- Range scan search through an index
- External large objects (BFILE)

- Synchronous Change Data Capture
- Transportable Tablespaces
- Original import/export utilities

In addition, you cannot use TDE column encryption to encrypt columns used in foreign key constraints.

See Also:

- "Export and Import of Tables with Encrypted Columns" on page 3-27
- "Data Types That Can Be Encrypted with TDE Column Encryption" on page 3-15

Note: Oracle Database 10*g* release 2 (10.2) TDE did not support large object (`LOB`) data types such as `BLOB` and `CLOB`. Oracle Database 11*g* TDE supports internal large object data types such as `BLOB` and `CLOB`. However, you cannot encrypt external `LOB`s (`BFILE`).

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

See Also: "DBMS_CRYPTO" in *Oracle Database PL/SQL Packages and Types Reference*

TDE protects data stored on disk/media. It does not protect data in transit. Use Oracle Advanced Security network encryption solutions discussed in Chapter 2, "Configuration and Administration Tools Overview"to encrypt data over the network.

Encrypting Entire Tablespaces

In order to use TDE tablespace encryption, you must be running Oracle Database 11*g* release 1 (11.1) or higher. If you have upgraded from an earlier release, the compatibility for the database must have been set to 11.0.0 or higher.

To use the enhanced tablespace encryption features in Oracle Database 11*g* Release 2 (11.2), the compatibility for the database must be set to 11.2 or higher.

Note: Advancing the database compatibility, using the `COMPATIBLE` initialization parameter, is an irreversible change.

The following steps discuss using TDE tablespace encryption:

- Setting the Tablespace Master Encryption Key
- Opening the Oracle Wallet
- Creating an Encrypted Tablespace
- Restrictions on Using TDE Tablespace Encryption

Setting the Tablespace Master Encryption Key

Before you can encrypt or decrypt tablespaces, you must generate or set a master encryption key. The tablespace master encryption key is stored in an external security module and is used to encrypt the TDE tablespace encryption keys.

Check to ensure that the ENCRYPTION_WALLET_LOCATION (or WALLET_LOCATION) parameter in the sqlnet.ora file points to the correct software wallet location. For example:

```
ENCRYPTION_WALLET_LOCATION=
  (SOURCE=(METHOD=FILE)(METHOD_DATA=
   (DIRECTORY=/app/wallet)))
```

Oracle Database 11*g* Release 2 (11.2) uses the same master encryption key for both TDE column encryption and TDE tablespace encryption. When you issue the ALTER SYSTEM SET ENCRYPTION KEY command, a unified master encryption key is created for both TDE column encryption and TDE tablespace encryption. Creating a master encryption key is discussed under "Setting the Master Encryption Key" on page 3-7.

If you were already using TDE in Oracle Database 10*g* release 2 (10.2), and have upgraded the database to 11*g* Release 2 (11.2), then you must reissue the ALTER SYSTEM SET ENCRYPTION KEY command to create a unified master encryption key.

If you were already using TDE tablespace encryption in Oracle Database 11*g* release 1 (11.1), and have upgraded the database to 11*g* release 2 (11.2), then you have separate master encryption keys for TDE column encryption and TDE tablespace encryption. You must create a unified master encryption key by reissuing the ALTER SYSTEM SET ENCRYPTION KEY command.

Resetting the Tablespace Master Encryption Key

Oracle Database 11*g* Release 2 (11.2) uses a unified master encryption key for both TDE column encryption and TDE tablespace encryption. When you reset (rekey) the master encryption key for TDE column encryption, the master encryption key for TDE tablespace encryption also gets reset.

The ALTER SYSTEM SET ENCRYPTION KEY command resets the tablespace master encryption key. Resetting the master encryption key is discussed under "Setting and Resetting the Master Encryption Key" on page 3-6.

Opening the Oracle Wallet

Before you can create an encrypted tablespace, the Oracle wallet containing the tablespace master encryption key must be open. The wallet must also be open before you can access data in an encrypted tablespace. Opening the Oracle wallet has been discussed under "Opening and Closing the Encrypted Wallet" on page 3-8.

> **Note:** The security administrator needs to open the Oracle wallet after starting the Oracle instance. A restart of the Oracle instance requires the security administrator to open the wallet again.

The security administrator also needs to open the wallet before performing database recovery operations. This is because background processes may require access to encrypted redo and undo logs. When performing database recovery, the wallet must be opened before opening the database. This is illustrated in the following statements:

```
SQL> STARTUP MOUNT;
```

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "password";
SQL> ALTER DATABASE OPEN;
```

You can also choose to use auto login wallets, if your environment does not require the extra security provided by a wallet that needs to be explicitly opened. .

Creating an Encrypted Tablespace

The CREATE TABLESPACE command enables you to create an encrypted tablespace. The permanent_tablespace_clause enables you to choose the encryption algorithm and the key length for encryption. The ENCRYPT keyword in the storage_clause encrypts the tablespace. The following syntax illustrates this:

```
CREATE
   [ BIGFILE | SMALLFILE ]
   { permanent_tablespace_clause
   | temporary_tablespace_clause
   | undo_tablespace_clause
   } ;
```

Where,

```
permanent_tablespace_clause=
TABLESPACE tablespace
.........
ENCRYPTION [USING algorithm]
.........
storage_clause
.........
```

Where,

```
storage_clause=
.........
[ENCRYPT]
.........
```

Here:

algorithm can have one of the following values:

- 3DES168
- AES128
- AES192
- AES256

The key lengths are included in the names of the algorithms themselves. If no encryption algorithm is specified, the default encryption algorithm is used. The default encryption algorithm is AES128.

Note:

- The ENCRYPTION keyword in the permanent_tablespace_clause is used to specify the encryption algorithm. The ENCRYPT keyword in the storage_clause actually encrypts the tablespace.
- For security reasons, a tablespace cannot be encrypted with the NO SALT option.

> **See Also:** *Oracle Database SQL Reference Guide* for the CREATE TABLESPACE command syntax.

Example 3–17 creates a tablespace called securespace. The tablespace is encrypted using the 3DES algorithm. The key length is 168 bits.

Example 3–17 Creating an Encrypted Tablespace

```
CREATE TABLESPACE securespace
DATAFILE '/home/user/oradata/secure01.dbf'
SIZE 150M
ENCRYPTION USING '3DES168'
DEFAULT STORAGE(ENCRYPT);
```

Example 3–18 creates a tablespace called securespace2. As no encryption algorithm is specified, the default encryption algorithm (AES128) is used. The key length is 128 bits.

Example 3–18 Creating an Encrypted Tablespace

```
CREATE TABLESPACE securespace2
DATAFILE '/home/user/oradata/secure01.dbf'
SIZE 150M
ENCRYPTION
DEFAULT STORAGE(ENCRYPT);
```

The following data dictionary views maintain information about the encryption status of a tablespace. You can query these views to verify that a tablespace has been encrypted:

- DBA_TABLESPACES: The ENCRYPTED column indicates whether a tablespace is encrypted
- USER_TABLESPACES: The ENCRYPTED column indicates whether a tablespace is encrypted

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

You cannot encrypt an existing tablespace. However, you can import data into an encrypted tablespace using the Oracle Data Pump utility. You can also use SQL commands like CREATE TABLE...AS SELECT... or ALTER TABLE...MOVE... to move data into an encrypted tablespace. The CREATE TABLE...AS SELECT... command enables you to create a table from an existing table. The ALTER TABLE...MOVE... command enables you to move a table into the encrypted tablespace.

> **See Also:** *Oracle Database SQL Language Reference* for more details on the CREATE TABLE and ALTER TABLE commands.

Restrictions on Using TDE Tablespace Encryption

TDE tablespace encryption encrypts / decrypts data during read / write operations, as opposed to TDE column encryption, which encrypts / decrypts data at the SQL layer. This means that most restrictions that apply to TDE column encryption, such as data type restrictions and index type restrictions, are not applicable to TDE tablespace encryption.

The following list includes the restrictions that apply to TDE tablespace encryption:

- External Large Objects (BFILEs) cannot be encrypted using TDE tablespace encryption. This is because these files reside outside the database.
- To perform import and export operations, use Oracle Data Pump.

Using Hardware Security Modules with TDE

A hardware security module (HSM) is a physical device that provides secure storage for encryption keys. It also provides secure computational space (memory) to perform encryption and decryption operations. HSM is a more secure alternative to the Oracle wallet.

TDE can use HSM to provide enhanced security for sensitive data. An HSM is used to store the master encryption key used for TDE. The key is secure from unauthorized access attempts as the HSM is a physical device and not an operating system file. All encryption and decryption operations that use the master encryption key are performed inside the HSM. This means that the master encryption key is never exposed in insecure memory.

Using HSM involves an initial setup of the HSM device. You also need to configure TDE to use HSM. Once the initial setup is done, HSM can be used just like an Oracle software wallet. The following steps discuss configuring and using hardware security modules:

1. Set the ENCRYPTION_WALLET_LOCATION Parameter in the sqlnet.ora File
2. Copy the PKCS#11 Library to Its Correct Path
3. Set Up the HSM
4. Generate a Master Encryption Key for HSM-Based Encryption
5. Reconfigure the Software Wallet (Optional)
6. Ensure that the HSM Is Accessible
7. Encrypt and Decrypt Data

Set the ENCRYPTION_WALLET_LOCATION Parameter in the sqlnet.ora File

The ENCRYPTION_WALLET_LOCATION parameter specifies the location of the Oracle wallet. You need to change this parameter to reflect the fact that an HSM is to be used in place of the software wallet.

Use the following steps to set the ENCRYPTION_WALLET_LOCATION parameter:

1. Open the sqlnet.ora file. This file is located in the $ORACLE_HOME/network/admin directory.
2. Add the ENCRYPTION_WALLET_LOCATION parameter to the sqlnet.ora file, as follows:

   ```
   ENCRYPTION_WALLET_LOCATION=(SOURCE=(METHOD=HSM))
   ```

   If the ENCRYPTION_WALLET_LOCATION parameter is already present in the sqlnet.ora file, then change the METHOD value to HSM:

   ```
   ENCRYPTION_WALLET_LOCATION=
     (SOURCE=(METHOD=HSM)(METHOD_DATA=
       (DIRECTORY=/app/wallet)))
   ```

> **Note:** If a DIRECTORY value is present in the ENCRYPTION_WALLET_LOCATION parameter, then make sure that you do not delete it. Although HSM does not require a DIRECTORY value, the value is used to locate your old software wallet when migrating to HSM-based transparent data encryption. Also, the DIRECTORY value might be required by tools, such as Recovery Manager (RMAN), to locate the software wallet.

3. Save and close the file.

Copy the PKCS#11 Library to Its Correct Path

Your HSM vendor supplies you with an associated PKCS#11 library. You should copy this library to the specified directory structure to ensure that the database is able to find this library. Use the following directory structures for UNIX and Windows respectively:

```
/opt/oracle/extapi/[32,64]/hsm/{VENDOR}/{VERSION}/libapiname.ext

%SYSTEM_DRIVE%\oracle\extapi\[32,64]\hsm\{VENDOR}\{VERSION}\libapiname.ext
```

Here:

`[32,64]` specifies whether the supplied binary is 32-bits or 64-bits

VENDOR stands for the name of the vendor supplying the library

VERSION refers to the version of the library. This should preferably be in a format, *number.number.number*

apiname requires no special format. However, the *apiname* must be prefixed with the word `lib`, as illustrated in the syntax.

.ext needs to be replaced by the extension of the library file. This extension is `.so` on Unix.

> **Note:** Only one PKCS#11 library is supported at a time. If you wish to use an HSM from a new vendor, then you should replace the PKCS#11 library from the earlier vendor with the library from the new vendor.

Set Up the HSM

Your HSM vendor should have provided you the instructions to set up the HSM interface. Use your HSM management interface and the instructions provided by your vendor to set up the HSM. Create the user account and password that would be used by the database to interact with the HSM.

> **Note:** The HSM is set up by the HSM administrator or the security administrator responsible for managing TDE.

Generate a Master Encryption Key for HSM-Based Encryption

To start using HSM-based encryption, you need to create a master encryption key that will be stored inside the HSM. The master encryption key is used to encrypt or decrypt table keys inside the HSM.

Use the following command to create the master encryption key:

```
SQL> ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "user_Id:password" [MIGRATE
USING "wallet_password"]
```

Here:

user_Id is the user Id created for the database using the HSM management interface

password is the password created for the user Id using the HSM management interface. Enclose the *user_Id:password* string in double quotation marks (" ").

wallet_password is the password required to open an existing Oracle wallet on the file system. Enclose the *wallet_password* string in double quotation marks (" ").

Note: The `user_Id` and `password` are not created automatically. You must set these up using the HSM management interface before issuing the `ALTER SYSTEM SET ENCRYPTION KEY` command. This is different from the procedure used for an Oracle wallet. An Oracle wallet requires no prior setup before issuing the `ALTER SYSTEM SET ENCRYPTION KEY` command.

If you are already using transparent data encryption and not using HSM, then you need to use the `MIGRATE USING` *wallet_password* clause in the preceding command. This decrypts the existing table keys and reencrypts them with the newly created, HSM-based, master encryption key.

Note: If the database contains columns encrypted with a public key, then the columns are decrypted and reencrypted with an AES symmetric key generated by HSM-based transparent data encryption.

Reconfigure the Software Wallet (Optional)

This step is applicable if you have exported encrypted data or created encrypted backups using the software wallet. Tools like Oracle Data Pump and Recovery Manager require access to the old software wallet to perform decryption and encryption operations on data exported or backed up using the software wallet.

You can use either of the following approaches to reconfigure the software wallet:

- Change the wallet password to the HSM *userId:password* string. Here:

  user_Id is the user Id created for the database using the HSM management interface

  password is the password created for the user Id using the HSM management interface. Enclose the *user_Id:password* string in double quotation marks (" ").

  Use Oracle Wallet Manager or the `orapki` command-line utility to change the password for the software wallet. SQL*Plus cannot be used to change the wallet password.

  See Also: "Changing the Password" on page 9-13 for more details on changing the wallet password

- You can alternatively choose to use an auto login wallet. The auto login wallet is identified by a file with the `.sso` extension. Use an auto login wallet only if your

environment does not require the extra security provided by a wallet that needs to be explicitly opened.

You can also choose to create a local auto login wallet. Local auto login wallets cannot be moved to another computer. They must be used on the host on which they are created.

See Also:

- "Using Auto Login" on page 9-14 for information about enabling auto login using Oracle Wallet Manager
- "Creating, Viewing, and Modifying Wallets with orapki" on page F-2 for information about enabling auto login and local auto login using the orapki command-line utility

Ensure that the HSM Is Accessible

The security administrator must make sure that the HSM is accessible to the database before any encryption or decryption can be performed. This is analogous to opening the Oracle wallet. Use the following command to make the HSM accessible:

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "user_Id:password"
```

Here:

`user_Id` is the user Id created for the database using the HSM management interface

`password` is the password created for the user Id using the HSM management interface

Enclose the *`user_Id:password`* string in double quotation marks (" ")

Note: Access to the HSM needs to reenabled every time the database instance is restarted.

The security administrator can disable access to the HSM using the `ALTER SYSTEM SET ENCRYPTION WALLET CLOSE IDENTIFIED BY "`*`user_Id`*`:`*`password`*`"` command. This disables all encryption and decryption operations in the HSM. A database user or application cannot perform any operation involving encrypted data until the wallet has been reopened. For example, the following operations will fail if the HSM is not accessible:

- `SELECT` data from an encrypted column
- `INSERT` data into on an encrypted column
- `CREATE` a table with encrypted column(s)
- `ALTER` the encryption properties of a column
- `CREATE` an encrypted tablespace

Encrypt and Decrypt Data

HSM use is transparent to the end user. The commands to create a table with encrypted columns, access encrypted data, or decrypt data are the same regardless of whether the master encryption key resides in an Oracle wallet or HSM.

Using Transparent Data Encryption with Oracle RAC

Oracle Database 11*g* Release 2 (11.2) enables Oracle Real Application Clusters (Oracle RAC) nodes to share the wallet. This eliminates the need to manually copy and synchronize the wallet across all nodes. Oracle recommends that you create the wallet on a shared file system. This allows all instances to access the same shared wallet.

Any wallet operation, like opening or closing the wallet, performed on any one Oracle RAC instance is applicable for all other Oracle RAC instances. This means that when you open and close the wallet for one instance, then it opens and closes for all Oracle RAC instances.

When using a shared file system, you need to ensure that the `ENCRYPTION_WALLET_LOCATION` or `WALLET_LOCATION` parameter for all Oracle RAC instances point to the same shared wallet location. The security administrator also needs to ensure security of the shared wallet by assigning appropriate directory permissions.

A master key rekey performed on one instance is applicable for all instances. When a new Oracle RAC node comes up, it is aware of the current wallet open or close status.

Using a Non-Shared File System to Store the Wallet

If you are not using a shared file system to store the wallet, then you need to copy the wallet to all nodes after a master key rekey. If you need to reset the master encryption key for the database, then use the following steps:

1. Reset the master encryption key on the first Oracle RAC node. Use the following command: See "Setting and Resetting the Master Encryption Key" on page 3-6 for more information.
2. Copy the wallet with the new master encryption key from the first node to all other nodes.
3. Close and reopen the wallet on any one node. Use the following commands:

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET CLOSE IDENTIFIED BY "password";
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "password";
```

> **Note:** Any wallet operation, like opening or closing the wallet, performed on any one Oracle RAC instance is applicable for all other Oracle RAC instances. This is true even if you are not using a shared file system.

All Oracle RAC nodes are now configured to use the new master encryption key.

Managing Transparent Data Encryption

This section contains these topics:

- Oracle Wallet Management
- Backup and Recovery of Master Encryption Keys
- Export and Import of Tables with Encrypted Columns
- Performance and Storage Overheads
- Security Considerations

- Using Transparent Data Encryption in a Multi-Database Environment
- Replication in Distributed Environments
- Compression and Data Deduplication of Encrypted Data
- Transparent Data Encryption with OCI
- Transparent Data Encryption in a Multi-Database Environment
- Transparent Data Encryption Data Dictionary Views

Oracle Wallet Management

Transparent Data Encryption (TDE) stores the master encryption key in an Oracle wallet. The wallet can also be an auto login wallet that allows access to encrypted data without requiring a security administrator to explicitly open the wallet.

Specifying a Separate Wallet for Transparent Data Encryption

When determining which wallet to use, TDE first attempts to use the wallet specified by the parameter `ENCRYPTION_WALLET_LOCATION`. If the parameter is not set, then it attempts to use the wallet specified by the parameter `WALLET_LOCATION`. If this fails as well, then TDE looks for a wallet at the default database location.

Oracle strongly recommends that you use a separate wallet for storing master encryption keys used by TDE. To designate a separate wallet, set the `ENCRYPTION_WALLET_LOCATION` parameter in the `sqlnet.ora` file to point to the wallet used exclusively by TDE.

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

Using an Auto Login Wallet

You can create an auto login wallet with Oracle Wallet Manager or the `orapki` command-line utility. The auto login wallet allows convenient access to encrypted data across database instance restarts.

> **Note:** You should not remove the `PKCS#12` wallet (`ewallet.p12` file) after the auto login wallet (`.sso` file) has been created. You need the `PKCS#12` wallet to regenerate/rekey the master encryption key in future.

TDE uses an auto login wallet only if it is available at the correct location (`ENCRYPTION_WALLET_LOCATION`, `WALLET_LOCATION`, or default wallet location), and the SQL command to open an encrypted wallet has not already been executed. If an auto login wallet is being used, you must not use the `ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "`*`password`*`"` command.

> **See Also:**
>
> - "Using Auto Login" on page 9-14 for information about enabling auto login using Oracle Wallet Manager
> - "Creating, Viewing, and Modifying Wallets with orapki" on page F-2 for information about enabling auto login and local auto login using the orapki command-line utility

Creating Wallets

When you create the master encryption key using the `ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "password"` command, TDE checks to see if a wallet exists in the default or specified location. If no wallet exists, then a wallet is created automatically.

In addition to the SQL command, you can also use Oracle Wallet Manager to create wallets. Oracle Wallet Manager is a full-featured tool that allows you to create wallets and to view and modify their content.

You can also use the `orapki` command like utility to create wallets.

See Also:

- Chapter 9, "Using Oracle Wallet Manager" for more information about Oracle Wallet Manager
- "Creating, Viewing, and Modifying Wallets with orapki"

Backup and Recovery of Master Encryption 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

You cannot access any encrypted data without the master encryption key. As the master encryption key is stored in the Oracle wallet, the wallet should be periodically backed up in a secure location. You must back up a copy of the wallet whenever a new master encryption key is set.

The Oracle wallet should not be backed up with the encrypted data. The wallet should be backed up separately. This is especially true when using the auto login wallet, which does not require a password to open. In case the backup tape gets lost, a malicious user should not be able to get both the encrypted data and the wallet.

Recovery Manager (RMAN) does not back up the wallet as part of the database backup. When using a media manager like Oracle Secure Backup (OSB) with RMAN, OSB automatically excludes auto-open wallets (the `cwallet.sso` files). However, encryption wallets (the `ewallet.p12` files) are not excluded automatically. It is a good practice to add the following exclude dataset statement to your OSB configuration:

```
exclude name *.p12
```

This instructs OSB to exclude the encryption wallet from the backup set.

If you lose the wallet that stores the master encryption 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 that the master encryption key was reset, then no additional action needs to be taken.

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

Backup and Recovery of PKI Key Pair

TDE column encryption supports the use of PKI asymmetric key pairs as master encryption 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. TDE 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 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:

```
SQL> 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

The following points are important when exporting tables containing encrypted columns:

- Sensitive data should remain unintelligible during transport
- Authorized users should be able to decrypt the data after it is imported at the destination

You can use the Oracle Data Pump utility to export and import tables containing encrypted columns. Oracle Data Pump makes use of the ENCRYPTION parameter to enable encryption of data in dump file sets. The ENCRYPTION parameter allows the following values:

- ENCRYPTED_COLUMNS_ONLY: Encrypted columns are written to the dump file set in encrypted format
- DATA_ONLY: All data is written to the dump file set in encrypted format
- METADATA_ONLY: All metadata is written to the dump file set in encrypted format
- ALL: All data and metadata is written to the dump file set in encrypted format
- NONE: Encryption is not used for dump file sets

The following steps discuss exporting and importing tables with encrypted columns using ENCRYPTION=ENCRYPTED_COLUMNS_ONLY:

1. You should ensure that the encryption wallet is open, before attempting to export tables containing encrypted columns. This is because the encrypted columns need to be decrypted using the table keys, which in turn requires access to the master encryption key. The columns are reencrypted using a password, before they are exported.
2. Use the ENCRYPTION_PASSWORD parameter to specify a password that is used to encrypt column data in the export dump file set. The following example exports the employee_data table:

```
expdp hr TABLES=employee_data DIRECTORY=dpump_dir
DUMPFILE=dpcd2be1.dmp ENCRYPTION=ENCRYPTED_COLUMNS_ONLY
ENCRYPTION_PASSWORD=PWD2encrypt

Password: password_for_hr
```

3. When importing data into the target database, you need to specify the same password. The password is used to decrypt the data. Data is reencrypted with the new table keys generated in the target database. The target database must have the wallet open to access the master encryption key. The following example imports the employee_data table:

```
impdp hr TABLES=employee_data DIRECTORY=dpump_dir DUMPFILE=dpcd2be1.dmp
ENCRYPTION_PASSWORD=PWD2encrypt

Password: password_for_hr
```

Oracle Data Pump functionality has been enhanced in Oracle Database 11*g* Release 2 (11.2). You can encrypt entire dump sets, as opposed to encrypting just transparent data encryption columns. The ENCRYPTION_MODE parameter enables you to specify the encryption mode.

ENCRYPTION_MODE=DUAL encrypts the dump set using the master key stored in the wallet and the password provided. The following example uses dual encryption mode:

```
expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr_enc.dmp
ENCRYPTION=all ENCRYPTION_PASSWORD=PWD2encrypt
ENCRYPTION_ALGORITHM=AES256 ENCRYPTION_MODE=dual

Password: password_for_hr
```

While importing, you can use either the password or the wallet master key to decrypt the data. If the password is not supplied, then the master key in the wallet is used to decrypt the data. The wallet must be present, and open, at the target database. The open wallet is also required to reencrypt column encryption data at the target database.

You can use ENCRYPTION_MODE=TRANSPARENT to transparently encrypt the dump file set with the master encryption key stored in the wallet. A password is not required in this case. The wallet must be present, and open, at the target database, for successful decryption during import. The open wallet is also required to reencrypt column encryption data at the target database.

See Also:

- "Overview of Data Pump", "Data Pump Export", and "Data Pump Import" in the *Oracle Database Utilities Guide* for details on using Oracle Data Pump and the associated encryption parameters.
- "Creating an Encrypted Column in an External Table" on page 3-12

Performance and Storage Overheads

The overhead associated with Transparent Data Encryption (TDE) can be categorized into the following:

- Performance Overheads
- Storage Overheads

Performance Overheads

TDE tablespace encryption has small associated overheads. While the actual performance impact on applications can vary, it is roughly estimated to be in between 5% and 8%.

TDE column encryption affects performance only when data is retrieved from or inserted into an encrypted column. No reduction in performance occurs for operations involving unencrypted columns, even if these columns are in a table containing encrypted columns.

Accessing data in encrypted columns involves small overheads. The overhead associated with encrypting or decrypting a common attribute, such as credit card number, is estimated to be around 5%. This means that a `SELECT` operation (involves decryption) or an `INSERT` operation (involves encryption) would take roughly 5% more time than what it takes with clear text data.

The total performance overhead 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.

> **See Also:** "Using the NOMAC Parameter to Save Disk Space and Improve Performance" on page 3-11

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.

A table can temporarily become inaccessible for write operations while encryption is being enabled, table keys are being rekeyed, or the encryption algorithm is being changed. You can use online table redefinition to ensure that the table is available for write operations during such procedures.

> **See Also:** "Redefining Tables Online" in *Oracle Database Administrator's Guide*

If TDE column encryption is being enabled on a very large table, then the redo log size might need to be increased to accommodate the operation.

It has also been observed that encrypting an indexed column takes more time than encrypting a column without indexes. If you need to encrypt a column that has an index built on it, you can try dropping the index, encrypting the column with `NO SALT`, and then re-creating the index.

If you index an encrypted column, then the index is created on the encrypted values. When you query for a value in the encrypted column, Oracle transparently encrypts the value used in the SQL query. It then performs an index lookup using the encrypted value.

Note: If you need to perform range scans over indexed, encrypted, columns, then you should use TDE tablespace encryption in place of TDE column encryption.

Storage Overheads

TDE tablespace encryption has no storage overheads. However, TDE column encryption has some associated storage overheads. Encrypted column data needs more storage space than clear text data. In addition, TDE pads out encrypted values to multiples of 16 bytes. This means that if a credit card number requires 9 bytes for storage, then an encrypted credit card value will require an additional 7 bytes.

Each encrypted value is also associated with a 20-byte integrity check. This is not applicable if you have encrypted columns using the `NOMAC` parameter. Also, if data has been encrypted with salt, then each encrypted value requires an additional 16 bytes of storage.

The maximum storage overhead for each encrypted value is 52 bytes.

See Also: "Using the NOMAC Parameter to Save Disk Space and Improve Performance" on page 3-11

Security Considerations

Security considerations for Transparent Data Encryption (TDE) operate within the broader arena of total system security. As a security administrator, you must identify the levels of risk to be addressed and the degrees of sensitivity of 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, a separate wallet for TDE, and protected backup procedures for encrypted data. Having a separate wallet for TDE permits auto-login for other Oracle components but preserves password protection for the TDE wallet.

Additional security considerations apply to normal database and network operations when using TDE. Encrypted column data stays encrypted in the data files, undo logs, redo logs, and the buffer cache of the system global area (SGA). However, data is decrypted during expression evaluation, making it possible for decrypted data to appear in the swap file on the disk. Privileged operating system users can potentially view this data.

Column values encrypted using TDE are stored in the data files in encrypted form. However, these data files may still contain some clear-text fragments, called ghost copies, left over by past data operations on the table. This is similar to finding data on the disk after a file has been deleted by the operating system.

Old clear-text fragments may be present for some time until the database overwrites the blocks containing such values. If privileged operating system users bypass the access controls of the database, they might be able to directly access these values in the data file holding the tablespace. You can use the following procedure to minimize this risk:

1. Create a new tablespace in a new data file. You can use the `CREATE TABLESPACE` statement.
2. Move the table containing encrypted columns to the new tablespace. You can use the `ALTER TABLE.....MOVE` statement. Repeat this step for all objects in the original tablespace.

3. Drop the original tablespace. You can use the `DROP TABLESPACE` *`tablespace`* `INCLUDING CONTENTS KEEP DATAFILES` statement. Oracle recommends that you securely delete data files using platform specific utilities.
4. Use platform and file system specific utilities to securely delete the old data file. Examples of such utilities include `shred` (on Linux) and `sdelete` (on Windows).

Using Transparent Data Encryption in a Multi-Database Environment

If there are multiple Oracle databases installed on the same server (for example, databases sharing the same Oracle binary but using different data files), then each database must access its own Transparent Data Encryption wallet. Wallets are not designed to be shared between databases. By design, there must be one wallet per database. You cannot use the same wallet for more than one database.

To configure the `sqlnet.ora` file for a multi-database environment, use one of the following options:

1. If the databases share the same Oracle home, then keep the `sqlnet.ora` file in the default location, which is in the *`ORACLE_HOME`*`/network/admin` directory.

   In this case, it is ideal to use the default location. Ensure that the `sqlnet.ora` file has no `WALLET_LOCATION` or `ENCRYPTION_WALLET_LOCATION` entries. Transparent Data Encryption accesses the wallet from the default `sqlnet.ora` location if these two entries are not in the `sqlnet.ora` file.
2. If Option 1 is not feasible for your site, then you can specify the wallet location based on an environment variable setting, such as `ORACLE_SID`. For example:

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE =
  (METHOD = FILE)
   (METHOD_DATA =
    (DIRECTORY = /home/oracle/wallet/$ORACLE_SID)
```

3. If Options 1 and 2 are not feasible, then use separate `sqlnet.ora` files, one for each database. Ensure that you have correctly set the `TNS_ADMIN` environment variable to point to the correct database configuration. See SQL*Plus User's Guide and Reference for more information and examples of setting the `TNS_ADMIN` variable.

Caution: Using a wallet from another database can cause partial or complete data loss.

Replication in Distributed Environments

Oracle Data Guard supports Transparent Data Encryption (TDE). If the primary database uses TDE, then each standby database in a Data Guard configuration must have a copy of the encryption wallet from the primary database. If you reset the master encryption key in the primary database, then the wallet containing the master encryption key needs to be copied to each standby database.

Encrypted data in log files remains encrypted when data is transferred to the standby database. Encrypted data also stays encrypted during transit.

> **See Also:** Appendix C in the *Oracle Data Guard Concepts and Administration Guide* for more information about the use of TDE with logical standby databases

TDE works with SQL*Loader direct path loads. The data loaded into encrypted columns is transparently encrypted during the direct path load.

Materialized views work with TDE tablespace encryption. You can create both materialized views and materialized view logs in encrypted tablespaces.

Materialized views also work with TDE column encryption. However, materialized view logs cannot contain encrypted columns.

> **See Also:** "Materialized View Concepts and Architecture" in the *Oracle Database Advanced Replication Guide* for more information on materialized views

Compression and Data Deduplication of Encrypted Data

After you have encrypted data, you cannot compress it. Any compression that you need to perform must take place before you encrypt the data. In Transparent Data Encryption column encryption, because much less data is encrypted, the post-encryption compression functions are not affected as much as with Transparent Data Encryption tablespace encryption.

You can use the following solutions to handle compression and data deduplication of encrypted data:

- To compress database tables before the encryption takes place, use the Advanced Compression Option of Oracle Recovery Manager.
- If you have many copies of very similar or even identical data stored on a single disk, then consider using storage-based data deduplication. To use data deduplication, use the Oracle SecureFiles LOB deduplication functionality on individual tables.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* for more information about the Advanced Compression Option
> - *Oracle Database SecureFiles and Large Objects Developer's Guide* for information about SecureFiles LOB storage

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 in a Multi-Database Environment

If there are multiple Oracle databases installed on the same server (for example, databases sharing the same Oracle binary but using different data files), then each database must access its own Transparent Data Encryption keystore. Wallets are not designed to be shared between databases. By design, there must be one wallet per database. You cannot use the same wallet for more than one database.

To configure the `sqlnet.ora` file for a multi-database environment, use one of the following options:

1. If the databases share the same Oracle home, then keep the `sqlnet.ora` file in the default location, which is in the *`ORACLE_HOME`*`/network/admin` directory.

   In this case, it is ideal to use the default location. Ensure that the `sqlnet.ora` file has no `WALLET_LOCATION` or `ENCRYPTION_WALLET_LOCATION` entries.

Transparent Data Encryption accesses the wallet from the default `sqlnet.ora` location if these two entries are not in the `sqlnet.ora` file.

2. If Option 1 is not feasible for your site, then you can specify the wallet location based on an environment variable setting, such as `ORACLE_SID`. For example:

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE =
  (METHOD = FILE)
   (METHOD_DATA =
    (DIRECTORY = /home/oracle/wallet/$ORACLE_SID)
```

3. If Options 1 and 2 are not feasible, then use separate `sqlnet.ora` files, one for each database. Ensure that the `TNS_ADMIN` environment variable is correctly set to point to the correct database configuration. See *SQL*Plus User's Guide and Reference* for more information and examples of setting the `TNS_ADMIN` variable.

Caution: Using a keystore from another database can cause partial or complete data loss.

Transparent Data Encryption Data Dictionary Views

The following data dictionary views maintain information about encryption details, tablespaces, and wallet details:

- `ALL_ENCRYPTED_COLUMNS`

  The `ALL_ENCRYPTED_COLUMNS` view displays encryption information about encrypted columns in the tables accessible to the current user. Table 3–2 lists the information included in this view:

Table 3–2 Description of the ALL_ENCRYPTED_COLUMNS Data Dictionary View

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the table
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the table
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the column
ENCRYPTION_ALG	VARCHAR2(29)		Encryption algorithm used to protect secrecy of data in this table: ■ 3 Key Triple DES 168 bits key ■ AES 128 bits key ■ AES 192 bits key ■ AES 256 bits key
SALT	VARCHAR2(3)		Indicates whether the column is encrypted with SALT (YES) or not (NO)
INTEGRITY_ALG	VARCHAR2(12)		Integrity algorithm used for the table: ■ SHA-1 ■ NOMAC

- `DBA_ENCRYPTED_COLUMNS`

The DBA_ENCRYPTED_COLUMNS view displays encryption information for all encrypted columns in the database. The view details are the same as the ALL_ENCRYPTED_COLUMNS view.

- USER_ENCRYPTED_COLUMNS

  The USER_ENCRYPTED_COLUMNS view displays encryption information for encrypted table columns in the user's schema. The view details are the same as the ALL_ENCRYPTED_COLUMNS view, except for the OWNER column. The OWNER column is not included, as data from only tables owned by the user are displayed.

- V$ENCRYPTED_TABLESPACES

  The V$ENCRYPTED_TABLESPACES view displays information about the tablespaces that are encrypted. Table 3–3 lists the information included in this view:

Table 3–3 Description of the V$ENCRYPTED_TABLESPACES View

Column	Datatype	Description
TS#	NUMBER	Tablespace number
ENCRYPTIONALG	VARCHAR2(7)	Encryption algorithm: ■ NONE ■ 3DES168 ■ AES128 ■ AES192 ■ AES256
ENCRYPTEDTS	VARCHAR2(3)	Indicates whether the tablespace is encrypted (YES) or not (NO)

- V$WALLET

  The V$WALLET view displays metadata information for a PKI certificate, which may be used as a master key for TDE. Table 3–4 summarizes the information included in this view.

Table 3–4 Description of the V$WALLET View

Column	Datatype	Description
CERT_ID	VARCHAR2(52)	A unique certificate identifier value used to specify a particular PKI certificate for use as the master key
DN	VARCHAR2(255)	Distinguished name of a particular PKI certificate
SERIAL_NUM	VARCHAR2(40)	Unique serial number assigned to a certificate by the issuer or signer
ISSUER	VARCHAR2(255)	Distinguished name of the Certificate Authority or issuer that issued and signed the certificate
KEYSIZE	NUMBER	Size of the PKI key associated with the certificate

Table 3–4 (Cont.) Description of the V$WALLET View

Column	Datatype	Description
STATUS	VARCHAR2(16)	Current status of the certificate: ■ UNUSED ■ IN USE ■ USED This column allows the user to identify whether a certificate is currently in use or has already been used for transparent database encryption.

- V$ENCRYPTION_WALLET

  V$ENCRYPTION_WALLET displays information on the status of the wallet and the wallet location for TDE. Table 3–5 summarizes the information included in this view.

Table 3–5 Description of the V$ENCRYPTION_WALLET View

Column	Datatype	Description
WRL_TYPE	VARCHAR2(20)	Type of the wallet resource locator (for example, FILE)
WRL_PARAMETER	VARCHAR2(4000)	Parameter of the wallet resource locator (for example, absolute filename if WRL_TYPE = FILE)
STATUS	VARCHAR2(9)	Status of the wallet: ■ OPEN ■ CLOSED ■ UNDEFINED ■ OPEN_NO_MASTER_KEY

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

Example: Getting Started with TDE Column Encryption and TDE Tablespace Encryption

This section uses a tutorial approach to help you get started with TDE column encryption and TDE tablespace encryption. We illustrate the following tasks using sample scenarios:

- Prepare the Database for Transparent Data Encryption
- Create a Table with an Encrypted Column
- Create an Index on an Encrypted Column
- Alter a Table to Encrypt an Existing Column
- Create an Encrypted Tablespace
- Create a Table in an Encrypted Tablespace

Prepare the Database for Transparent Data Encryption

In order to start using Transparent Data Encryption (TDE), let us first prepare the database by specifying an Oracle wallet location and setting the master encryption key. The following steps prepare the database to use TDE:

1. Specify an Oracle Wallet Location in the sqlnet.ora File
2. Create the Master Encryption Key
3. Open the Oracle Wallet

Specify an Oracle Wallet Location in the sqlnet.ora File

Open the `sqlnet.ora` file located in `$ORACLE_HOME/network/admin`. Enter the following line at the end of the file:

```
ENCRYPTION_WALLET_LOCATION=
  (SOURCE=(METHOD=FILE)(METHOD_DATA=
   (DIRECTORY=/app/wallet)))
```

Save the changes and close the file.

> **Note:** You can choose any directory for the encrypted wallet, but the path should not point to the standard obfuscated wallet (`cwallet.sso`) created during the database installation.

Create the Master Encryption Key

Next, we need to create the master encryption key, which is used to encrypt the table keys. Enter the following commands to create the master encryption key:

```
SQL> ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "Easy2rem";
```

The preceding command achieves the following:

- If no encrypted wallet is present in the directory specified, an encrypted wallet is created (`ewallet.p12`), the wallet is opened, and the master encryption key for TDE is created / re-created.
- If an encrypted wallet is present in the directory specified, the wallet is opened, and the master encryption key for TDE is created / re-created.

> **Note:**
>
> - The master encryption key should only be created once, unless you want to reencrypt your data with a new encryption key.
> - Only users with the `ALTER SYSTEM` privilege can create a master encryption key or open the wallet.

Open the Oracle Wallet

Every time the database is shut down, the Oracle wallet is closed. You can also explicitly close the wallet.

You need to make sure that the Oracle wallet is open before you can perform any encryption or decryption operation. Use the following command to open the wallet containing the master encryption key:

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "Easy2rem";
```

Note: The password used with the preceding command is the same that you used to create the master encryption key. This becomes the password to open the wallet and make the master encryption key accessible.

Create a Table with an Encrypted Column

We can now create tables with encrypted columns. Let us create a table called `cust_payment_info`. This table contains a column called `credit_card_number`. The `credit_card_number` column contains sensitive data, which we would like to encrypt. Use the following command to create the table:

```
CREATE TABLE cust_payment_info
  (first_name VARCHAR2(11),
  last_name VARCHAR2(10),
  order_number NUMBER(5),
  credit_card_number VARCHAR2(16) ENCRYPT NO SALT,
  active_card VARCHAR2(3));
```

The table is created in the default tablespace of the user that issues this command. The `credit_card_number` column is encrypted without SALT. All data entered for the `credit_card_number` column would be encrypted on disk. Any user with access to the `credit_card_number` data can view the decrypted data. A database user or application need not be aware if the contents of a particular column are encrypted on the disk.

You can now enter data into the table. The following example adds some sample data to the `cust_payment_info` table:

```
INSERT INTO cust_payment_info VALUES
  ('Jon', 'Oldfield', 10001, '5446959708812985','YES');
INSERT INTO cust_payment_info VALUES
  ('Chris', 'White', 10002, '5122358046082560','YES');
INSERT INTO cust_payment_info VALUES
  ('Alan', 'Squire', 10003, '5595968943757920','YES');
INSERT INTO cust_payment_info VALUES
  ('Mike', 'Anderson', 10004, '4929889576357400','YES');
INSERT INTO cust_payment_info VALUES
  ('Annie', 'Schmidt', 10005, '4556988708236902','YES');
INSERT INTO cust_payment_info VALUES
  ('Elliott', 'Meyer', 10006, '374366599711820','YES');
INSERT INTO cust_payment_info VALUES
  ('Celine', 'Smith', 10007, '4716898533036','YES');
INSERT INTO cust_payment_info VALUES
  ('Steve', 'Haslam', 10008, '340975900376858','YES');
INSERT INTO cust_payment_info VALUES
  ('Albert', 'Einstein', 10009, '310654305412389','YES');
```

All data entered into the `credit_card_number` column is stored on the disk in encrypted form.

Create an Index on an Encrypted Column

You can create an index on an encrypted column if it has been encrypted without salt. Let us create an index on the `credit_card_number` column. The following command creates an index on the `credit_card_number` column:

```
CREATE INDEX cust_payment_info_idx ON cust_payment_info (credit_card_number);
```

Alter a Table to Encrypt an Existing Column

You can use the `ALTER TABLE` command to alter an existing table. Let us alter a table called `employees` with no encrypted columns. The following command describes the employees table:

```
SQL> DESC employees
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 FIRSTNAME                                          VARCHAR2(11)
 LASTNAME                                           VARCHAR2(10)
 EMP_SSN                                            VARCHAR2(9)
 DEPT                                               VARCHAR2(20)
```

The following command encrypts the `emp_ssn` column in the employees table:

```
SQL> ALTER TABLE employees MODIFY (emp_ssn ENCRYPT);
```

The following command describes the altered `employees` table:

```
SQL> DESC employees
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 FIRSTNAME                                          VARCHAR2(11)
 LASTNAME                                           VARCHAR2(10)
 EMP_SSN                                            VARCHAR2(9) ENCRYPT
 DEPT                                               VARCHAR2(20)
```

All existing data in the `emp_ssn` column will now be encrypted on the disk. Data would be transparently decrypted for users, who otherwise have access to the data.

Create an Encrypted Tablespace

TDE tablespace encryption enables you to encrypt an entire tablespace. All data stored in the tablespace is encrypted by default. Thus, if you create any table in an encrypted tablespace, it is encrypted by default. You do not need to perform a granular analysis of each table column to determine the columns that need encryption.

Let us create an encrypted tablespace to store encrypted tables. The following command creates an encrypted tablespace called `securespace`:

```
SQL> CREATE TABLESPACE securespace
  2  DATAFILE '/home/oracle/oracle3/product/11.1.0/db_1/secure01.dbf'
  3  SIZE 150M
  4  ENCRYPTION
  5  DEFAULT STORAGE(ENCRYPT);
Tablespace created.
```

Create a Table in an Encrypted Tablespace

If we create a table in an encrypted tablespace, then all data in the table is stored in encrypted form on the disk. The following command creates a table called, `customer_info_payment` in an encrypted tablespace called, `securespace`.

```
SQL> CREATE TABLE customer_payment_info
  2  (first_name VARCHAR2(11),
  3  last_name VARCHAR2(10),
```

```
  4  order_number NUMBER(5),
  5  credit_card_number VARCHAR2(16),
  6  active_card VARCHAR2(3))TABLESPACE securespace;
Table created.
```

Troubleshooting Transparent Data Encryption

This section lists common error messages that you may encounter while configuring and using Transparent Data Encryption (TDE). It also lists the common causes of these error messages and possible solutions for them.

ORA-28330: encryption is not allowed for this data type

Cause: Data type was not supported for column encryption.

Action: None

ORA-28331: encrypted column size too long for its data type

Cause: column was encrypted and for VARCHAR2, the length specified was > 3932; for CHAR, the length specified was > 1932; for NVARCHAR2, the length specified was > 1966; for NCHAR, the length specified was > 966;

Action: Reduce the column size.

ORA-28332: cannot have more than one password for the encryption key

Cause: More than one password was specified in the user command.

Action: None

ORA-28333: column is not encrypted

Cause: An attempt was made to rekey or decrypt an unencrypted column.

Action: None

ORA-28334: column is already encrypted

Cause: An attempt was made to encrypt an encrypted column.

Action: None

ORA-28335: referenced or referencing FK constraint column cannot be encrypted

Cause: encrypted columns were involved in the referential constraint

Action: None

ORA-28336: cannot encrypt SYS owned objects

Cause: An attempt was made to encrypt columns in a table owned by SYS.

Action: None

ORA-28337: the specified index may not be defined on an encrypted column

Cause: Index column was either a functional, domain, or join index.

Action: None

ORA-28338: cannot encrypt indexed column(s) with salt

Cause: An attempt was made to encrypt index column with salt.

Action: Alter the table and specify column encrypting without salt.

ORA-28339: missing or invalid encryption algorithm

Cause: Encryption algorithm was missing or invalid in the user command.

Action: Must specify a valid algorithm.

ORA-28340: a different encryption algorithm has been chosen for the table

Cause: Existing encrypted columns were associated with a different algorithm.

Action: No need to specify an algorithm, or specify the same one for the existing encrypted columns.

ORA-28341: cannot encrypt constraint column(s) with salt

Cause: An attempt was made to encrypt constraint columns with salt.

Action: Encrypt the constraint columns without salt.

ORA-28342: integrity check fails on column key

Cause: Encryption metadata may have been improperly altered.

Action: None

ORA-28343: fails to encrypt data

Cause: data or encryption metadata may have been improperly altered or the security module may not have been properly setup

Action: None

ORA-28344: fails to decrypt data

Cause: data or encryption metadata may have been improperly altered or the security module may not have been properly setup

Action: None

ORA-28345: cannot downgrade because there exists encrypted column

Cause: An attempt was made to downgrade when there was an encrypted column in the system.

Action: Decrypt these columns before attempting to downgrade.

ORA-28346: an encrypted column cannot serve as a partitioning column

Cause: An attempt was made to encrypt a partitioning key column or create partitioning index with encrypted columns.

Action: The column must be decrypted.

ORA-28347: encryption properties mismatch

Cause: An attempt was made to issue an ALTER TABLE EXCHANGE PARTITION | SUBPARTITION command, but encryption properties were mismatched.

Action: Make sure encryption algorithms and columns keys are identical. The corresponding columns must be encrypted on both tables with the same salt and non-salt flavor.

ORA-28348: index defined on the specified column cannot be encrypted

Cause: An attempt was made to encrypt a column which is in a functional index, domain index, or join index.

Action: drop the index

ORA-28349: cannot encrypt the specified column recorded in the materialized view log

Cause: An attempt was made to encrypt a column which is already recorded in the materialized view log.

Action: drop the materialized view log

ORA-28350: cannot encrypt the specified column recorded in CDC synchronized change table

Cause: An attempt was made to encrypt a column which is already recorded in CDC synchronized change table.

Action: drop the synchronized change table

ORA-28351: cannot encrypt the column of a cluster key

Cause: An attempt was made to encrypt a column of the cluster key. A column of the cluster key in a clustered table cannot be encrypted.

Action: None

ORA-28353: failed to open wallet

Cause: The database was unable to open the security module wallet due to an incorrect wallet path or password It is also possible that a wallet has not been created.

Action: Execute the command again using the correct wallet password or verifying a wallet exists in the specified directory. If necessary, create a new wallet and initialize it.

ORA-28354: wallet already open

Cause: The security module wallet has already been opened.

Action: None

ORA-28356: invalid open wallet syntax

Cause: The command to open the wallet contained improper spelling or syntax.

Action: If attempting to open the wallet, verify the spelling and syntax and execute the command again.

ORA-28357: password required to open the wallet

Cause: A password was not provided when executing the open wallet command.

Action: Retry the command with a valid password.

ORA-28358: improper set key syntax

Cause: The command to set the master key contained improper spelling or syntax.

Action: If attempting to set the master key for Transparent Database Encryption, verify the spelling and syntax and execute the command again.

ORA-28359: invalid certificate identifier

Cause: The certificate specified did not exist in the wallet.

Action: Query the V$WALLET fixed view to find the proper certificate identifier for certificate to be used.

ORA-28361: master key not yet set

Cause: The master key for the instance was not set.

Action: Execute the ALTER SYSTEM SET KEY command to set a master key for the database instance.

ORA-28362: master key not found

Cause: The required master key required could not be located. This may be caused by the use of an invalid or incorrect wallet.

Action: Check wallet location parameters to see if they specify the correct wallet. Also, verify that an SSO wallet is not being used when an encrypted wallet is intended.

ORA-28363: buffer provided not large enough for output

Cause: A provided output buffer is too small to contain the output.

Action: Check the size of the output buffer to make sure it is initialized to the proper size.

ORA-28364: invalid wallet operation

Cause: The command to operate the wallet contained improper spelling or syntax.

Action: Verify the spelling and syntax and execute the command again.

ORA-28365: wallet is not open

Cause: The security module wallet has not been opened.

Action: Open the wallet.

ORA-28366: invalid database encryption operation

Cause: The command for database encryption contained improper spelling or syntax.

Action: Verify the spelling and syntax and execute the command again.

ORA-28367: wallet does not exist

Cause: The Oracle wallet has not been created or the wallet location parameters in sqlnet.ora specifies an invalid wallet path.

Action: Verify that the WALLET_LOCATION or the ENCRYPTION_WALLET_LOCATION parameter is correct and that a valid wallet exists in the path specified.

ORA-28368: cannot auto-create wallet

Cause: The database failed to auto create an Oracle wallet. The Oracle process may not have proper file permissions or a wallet may already exist.

Action: Confirm that proper directory permissions are granted to the Oracle user and that neither an encrypted or obfuscated wallet exists in the specified wallet location and try again.

ORA-28369: cannot add files to encryption-ready tablespace when offline

Cause: You attempted to add files to an encryption-ready tablespace when all the files in the tablespace were offline.

Action: Bring the tablespace online and try again

ORA-28370: ENCRYPT storage option not allowed

Cause: You attempted to specify the ENCRYPT storage option. This option may only be specified during CREATE TABLESPACE.

Action: Remove this option and retry the statement.

ORA-28371: ENCRYPTION clause and/or ENCRYPT storage option not allowed

Cause: You attempted to specify the ENCRYPTION clause or ENCRYPT storage option for creating TEMP or UNDO tablespaces.

Action: Remove these options and retry the statement.

ORA-28372: missing ENCRYPT storage option for encrypted tablespace

Cause: You attempted to specify ENCRYPTION property for CREATE TABLESPACE without specifying ENCRYPT storage option to encrypt the tablespace.

Action: Add ENCRYPT storage option and retry the statement.

ORA-28373: missing ENCRYPTION clause for encrypted tablespace

Cause: You attempted to specify storage option ENCRYPT in CREATE TABLESPACE without specifying ENCRYPTION property to encrypt the tablespace.

Action: Add ENCRYPTION clause and retry the statement.

ORA-28374: typed master key not found in wallet

Cause: You attempted to access encrypted tablespace or redo logs with a typed master key not existing in the wallet.

Action: Copy the correct Oracle Wallet from the instance where the tablespace was created.

ORA-28375: cannot perform cross-endianism conversion on encrypted tablespace

Cause: You attempted to perform cross-endianism conversion on encrypted tablespace.

Action: Cross-endianism conversion on encrypted tablespace is not supported.

ORA-28376: cannot find PKCS11 library

Cause: The HSM vendor's library cannot be found.

Action: Place the HSM vendor's library in the following directory structure: For Unix like system:
/opt/oracle/extapi/[32,64]/hsm/{VENDOR}/{VERSION}/lib<apiname>.<ext>
For Windows systems:
%SYSTEM_DRIVE%\oracle\extapi\[32,64]\hsm\{VENDOR}\{VERSION}\lib<apin/ / ame>.<ext> [32, 64] - refers to 32bit or 64bit binary. {VENDOR} - The name of the vendor supplying the library. {VERSION} - Version of the library, preferably in num#.num#.num# for/ / mat.

ORA-28377: No need to migrate from wallet to HSM

Cause: There are either no encrypted columns or all column keys are already encrypted with the HSM master key.

Action: No action required.

ORA-28378: Wallet not open after setting the Master Key

Cause: The Master Key has been set or reset. However, wallet could not be reopened successfully.

Action: Reopen the wallet.

Transparent Data Encryption Reference Information

This section includes the following topics:

- Supported Encryption and Integrity Algorithms
- Quick Reference: Transparent Data Encryption SQL Commands

Supported Encryption and Integrity Algorithms

By default, Transparent Data Encryption (TDE) 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 3–4 on page 3-11 for the correct syntax when choosing the `NO SALT` parameter for the SQL `ENCRYPT` clause
- "Changing the Encryption Key or Algorithm for Tables Containing Encrypted Columns" on page 3-14 for syntax examples when setting a different algorithm with the SQL `ENCRYPT` clause

Table 3–6 lists the supported encryption algorithms.

Table 3–6 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.

Quick Reference: Transparent Data Encryption SQL Commands

Table 3–7 provides a summary of the SQL commands you can use to implement and manage transparent data encryption.

Table 3–7 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 encryption key: set or reset	`ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "password";`
Master encryption key: set or reset to use PKI certificate	`ALTER SYSTEM SET ENCRYPTION KEY "certificate_ID" IDENTIFIED BY "password";`
Wallet: open to access master encryption key	`ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "password";`

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 Negotiation
- How To Configure Data Encryption and Integrity

Oracle Advanced Security 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.

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

- 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.

Advanced Encryption Standard

Oracle Advanced Security supports the Federal Information Processing Standard (FIPS) encryption algorithm, Advanced Encryption Standard (AES). 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 supports 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. The DES algorithm uses a 56-bit key length. DES has been largely replaced, as a standard, by AES.

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 11*g* Release 2 (11.2), 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 11*g* Release 2 (11.2) 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 Negotiation

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
- 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 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.

Oracle Database servers and clients are set to ACCEPT encrypted connections out of the box. This means that you can enable the desired encryption and integrity settings for a connection pair by configuring just one side of the connection, server-side or client-side.

So, for example, if there are many Oracle clients connecting to an Oracle database, you can configure the required encryption and integrity settings for all these connections by making the appropriate sqlnet.ora changes at the server end. You do not need to implement configuration changes for each client separately.

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.

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:

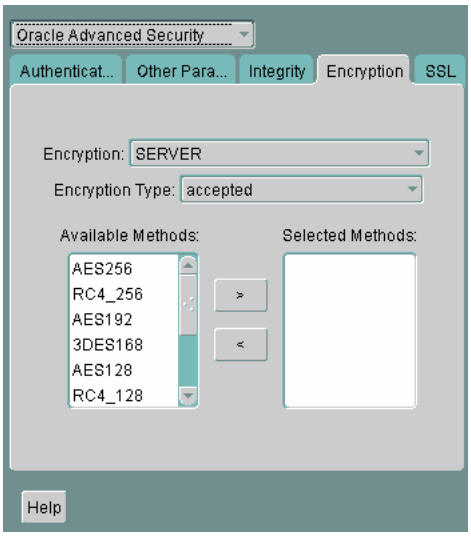
- Appendix A, "Data Encryption and Integrity Parameters", for valid encryption algorithms
- Oracle Net Manager online help, for more detailed configuration information

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



1. Click the **Encryption** tab.
2. Select **CLIENT** or **SERVER** option from the Integrity box.
3. From the Encryption Type list, select one of the following:
   - REQUESTED
   - REQUIRED
   - ACCEPTED
   - REJECTED
4. (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.
5. 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.
6. Select **File**, **Save Network Configuration**. The `sqlnet.ora` file is updated.

7. 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

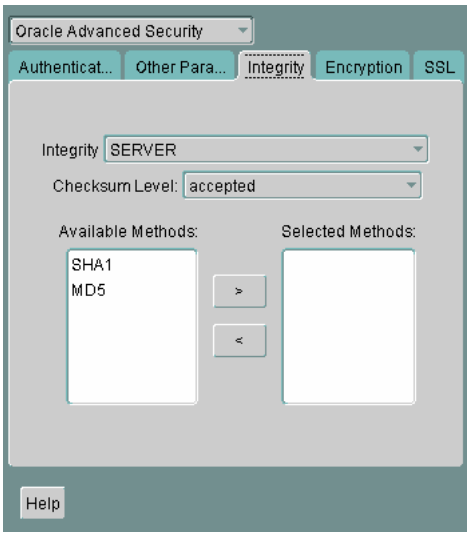
Configuring Integrity on the Client and the Server

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



1. Click the **Integrity** tab.
2. Depending upon which system you are configuring, select the **Server** or **Client** from the **Integrity** box.
3. From the **Checksum Level** list, select one of the following checksum level values:
   - REQUESTED
   - REQUIRED
   - ACCEPTED
   - REJECTED
4. 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.
5. Select **File**, **Save Network Configuration**. The `sqlnet.ora` file is updated.
6. 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 Authentication, 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 authentication, 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 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

As 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 authentication, encryption, and integrity algorithms, for use with thin clients. Oracle Advanced Security provides the following features for Thin JDBC:

- Strong Authentication
- 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 11*g* Release 2 (11.2) to older versions of Oracle databases with Oracle Advanced Security installed

The Oracle JDBC Thin driver supports Oracle Advanced Security SSL implementation and third party authentication methods such as RADIUS and Kerberos. Thin JDBC support for authentication methods like RADIUS, Kerberos, and SSL were introduced in Oracle Database 11*g* Release 1 (11.1).

The Oracle Advanced Security Java implementation provides Java versions of the following encryption algorithms:

- `AES256`: AES 256-bit key
- `AES192`: AES 192-bit key
- `AES128`: AES 128-bit key
- `3DES168`: 3-key 3DES
- `3DES112`: 2-key 3DES
- `DES56C`: DES 56-bit key CBC
- `DES40C`: DES 40-bit key CBC
- `RC4_256`: RC4 256-bit key
- `RC4_128`: RC4 128-bit key
- `RC4_56`: RC4 56-bit key
- `RC4_40`: RC4 40-bit key

> **Note:** In the preceding list of algorithms, CBC refers to the Cipher Block Chaining mode.

Thin JDBC support for the Advanced Encryption Standard (AES) has been newly introduced in Oracle Database 11*g* Release 2 (11.2).

In addition, this implementation provides data integrity checking for Thin JDBC using Secure Hash Algorithm (`SHA1`) and Message Digest 5 (`MD5`). Thin JDBC support for `SHA1` was introduced in Oracle Database 11*g* release 1 (11.1).

> **See Also:** *Oracle Database JDBC Developer's Guide and Reference* for details on configuring authentication, encryption, and integrity for thin JDBC clients.

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 OCI clients. The client and server negotiate encryption algorithms, generate random numbers, use Diffie-Hellman to exchange session keys, and use the Oracle Password Protocol, 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

The Java cryptography code is *obfuscated*. 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.

All JDBC connection properties including the ones pertaining to Oracle Advanced Security are defined as constants in the `oracle.jdbc.OracleConnection` interface. The following list enumerates some of these connection properties:

- Client Encryption Level:
  CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL
- Client Encryption Selected List:
  CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES
- Client Integrity Level:
  CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL
- Client Integrity Selected List:
  CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES
- Client Authentication Service:
  CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES

See Also: *Oracle Database JDBC Developer's Guide and Reference* for detailed information on configuration parameters and configuration examples

Client Encryption Level: CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL

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

Table 5–1 CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL Parameter Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	`REJECTED`; `ACCEPTED`; `REQUESTED`; `REQUIRED`
Default Value	`ACCEPTED`
Syntax	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL,`*`level`*`);` where `prop` is an object of the `Properties` class
Example	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL,"REQUIRED");` where `prop` is an object of the `Properties` class

Client Encryption Selected List: CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES

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

Table 5–2 CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES Parameter Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	`AES256` (AES 256-bit key), `AES192` (AES 192-bit key), `AES128` (AES 128-bit key), `3DES168` (3-key 3DES), `3DES112` (2-key 3DES), `DES56C` (DES 56-bit key CBC), `DES40C` (DES 40-bit key CBC), `RC4_256` (RC4 256-bit key), `RC4_128` (RC4 128-bit key), `RC4_56` (RC4 56-bit key), `RC4_40` (RC4 40-bit key)
Syntax	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES,`*`algorithm`*`);` where `prop` is an object of the `Properties` class
Example	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES, "( AES256, AES192 )");` where `prop` is an object of the `Properties` class

Client Integrity Level: CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL

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

Table 5–3 CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL Parameter Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	`REJECTED`; `ACCEPTED`; `REQUESTED`; `REQUIRED`
Default Value	`ACCEPTED`
Syntax	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL,`*`level`*`);` where `prop` is an object of the `Properties` class
Example	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL,"REQUIRED");` where `prop` is an object of the `Properties` class

Client Integrity Selected List: CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES

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

Table 5–4 CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES Parameter Attributes

Attribute	Description
Parameter Type	String
Parameter Class	Static
Permitted Values	`MD5`, `SHA1`
Syntax	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES,` *`algorithm`*`);` where `prop` is an object of the `Properties` class
Example	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES,"( MD5, SHA1 )");` where `prop` is an object of the `Properties` class

Client Authentication Service: CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES

This parameter determines the authentication service to be used. Table 5–5 describes this parameter's attributes.

Table 5–5 CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES Parameter Attributes

Attribute	Description
Parameter Type	String

Table 5–5 (Cont.) CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES Parameter Attributes

Attribute	Description
Parameter Class	Static
Permitted Values	`RADIUS, KERBEROS, SSL`
Syntax	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES,`*`authentication`*`);` where `prop` is an object of the `Properties` class
Example	`prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES,"(RADIUS, KERBEROS, SSL)");` where `prop` is an object of the `Properties` class

AnoServices Constants

The `oracle.net.ano.AnoServices` interface has been updated in this release to include the names of all the encryption, authentication, and checksum algorithms supported by the JDBC Thin driver. The following constants have been added to the oracle.net.ano.AnoServices interface:

```
// ---- SUPPORTED ENCRYPTION ALG -----
public static final String ENCRYPTION_RC4_40 = "RC4_40";
public static final String ENCRYPTION_RC4_56 = "RC4_56";
public static final String ENCRYPTION_RC4_128 = "RC4_128";
public static final String ENCRYPTION_RC4_256 = "RC4_256";
public static final String ENCRYPTION_DES40C = "DES40C";
public static final String ENCRYPTION_DES56C = "DES56C";
public static final String ENCRYPTION_3DES112 = "3DES112";
public static final String ENCRYPTION_3DES168 = "3DES168";
public static final String ENCRYPTION_AES128 = "AES128";
public static final String ENCRYPTION_AES192 = "AES192";
public static final String ENCRYPTION_AES256 = "AES256";
// ---- SUPPORTED INTEGRITY ALG ----
public static final String CHECKSUM_MD5 = "MD5";
public static final String CHECKSUM_SHA1 = "SHA1";
// ---- SUPPORTED AUTHENTICATION ADAPTORS ----
public static final String AUTHENTICATION_RADIUS = "RADIUS";
public static final String AUTHENTICATION_KERBEROS = "KERBEROS";
```

You can use these constants to set the encryption, integrity, and authentication parameters. Example 5–1 illustrates one such scenario.

Example 5–1 Using AnoServices Constants in JDBC Client Code

```
import java.sql.*;
import java.util.Properties;
import oracle.jdbc.*;
import oracle.net.ano.AnoServices;
/**
 * JDBC thin driver demo: new security features in 11gR1.
 *
 * This program attempts to connect to the database using the JDBC thin
 * driver and requires the connection to be encrypted with either AES256 or AES192
 * and the data integrity to be verified with SHA1.
 *
 * In order to activate encryption and checksumming in the database you need to
```

```
 * modify the sqlnet.ora file. For example:
 *
 *   SQLNET.ENCRYPTION_TYPES_SERVER = (AES256,AES192,AES128)
 *   SQLNET.ENCRYPTION_SERVER = accepted
 *   SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER= (SHA1)
 *   SQLNET.CRYPTO_CHECKSUM_SERVER = accepted
 *
 * This output of this program is:
 *   Connection created! Encryption algorithm is: AES256, data integrity algorithm
 *   is: SHA1
 *
 */
public class DemoAESAndSHA1
{
  static final String USERNAME= "hr";
  static final String PASSWORD= "hr";
  static final String URL =
"jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=somehost.us.example.c
om)(PORT=5561))"
  +"(CONNECT_DATA=(SERVICE_NAME=itydemo.regress.rdbms.dev.us.example.com)))";

  public static final void main(String[] argv)
  {
    DemoAESAndSHA1 demo = new DemoAESAndSHA1();
    try
    {
      demo.run();
    }catch(SQLException ex)
    {
      ex.printStackTrace();
    }
  }
  void run() throws SQLException
  {
    OracleDriver dr = new OracleDriver();
    Properties prop = new Properties();
    // We require the connection to be encrypted with either AES256 or AES192.
    // If the database doesn't accept such a security level, then the connection
    // attempt will fail.
    prop.setProperty(

OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL,AnoServices.ANO_REQ
UIRED);
    prop.setProperty(
      OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES,
      "( " + AnoServices.ENCRYPTION_AES256 + "," +AnoServices.ENCRYPTION_AES192 +
")");
    // We also require the use of the SHA1 algorithm for data integrity checking.
    prop.setProperty(

OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL,AnoServices.ANO_REQUI
RED);
    prop.setProperty(
      OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES,
      "( " + AnoServices.CHECKSUM_SHA1 + " )");

    prop.setProperty("user",DemoAESAndSHA1.USERNAME);
    prop.setProperty("password",DemoAESAndSHA1.PASSWORD);
    OracleConnection oraConn =
 (OracleConnection)dr.connect(DemoAESAndSHA1.URL,prop);
```

```
    System.out.println("Connection created! Encryption algorithm is:
"+oraConn.getEncryptionAlgorithmName()
    +", data integrity algorithm is: "+oraConn.getDataIntegrityAlgorithmName());

    oraConn.close();
  }

}
```

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.

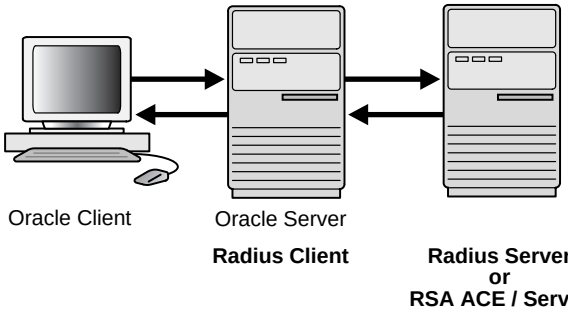
From the user's perspective, the entire authentication process is transparent. When the user seeks access to an Oracle database server, the Oracle database server, acting as the RADIUS client, notifies the RADIUS server. The RADIUS server:

- 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



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:

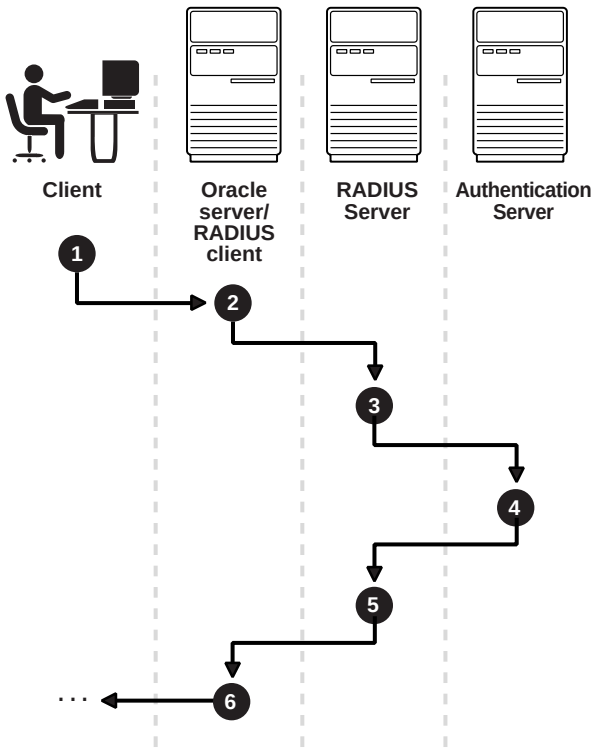
- Synchronous Authentication Mode

- 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.

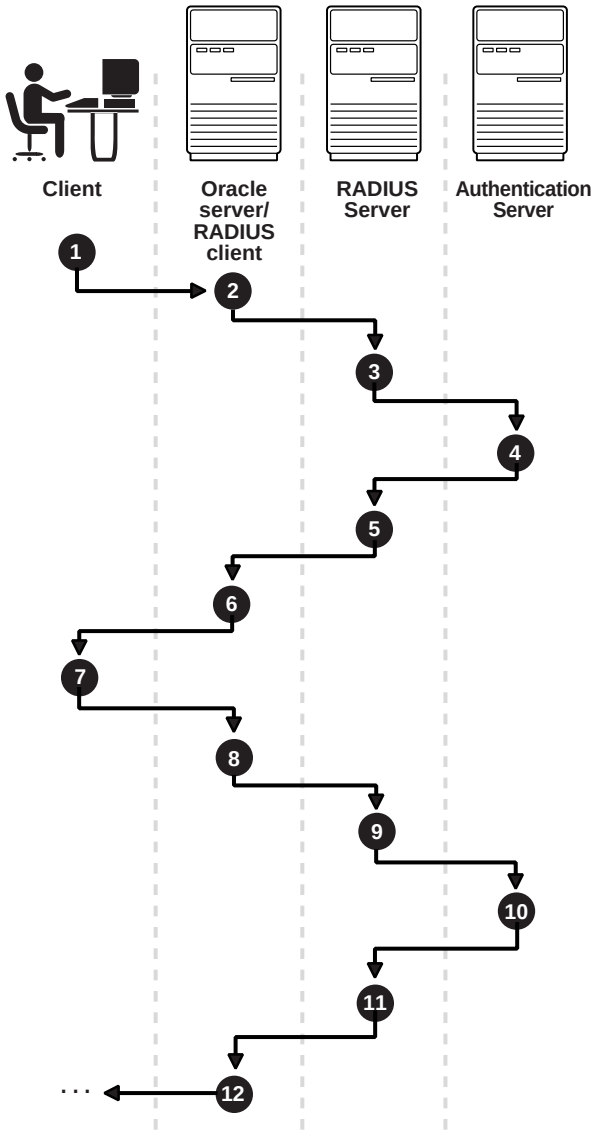
Challenge-Response (Asynchronous) Authentication Mode

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 initiates 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 into a smart card reader that reads the smart card. The smart card is a plastic card, like a credit card, with an embedded integrated circuit for storing information.

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, authorization, 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

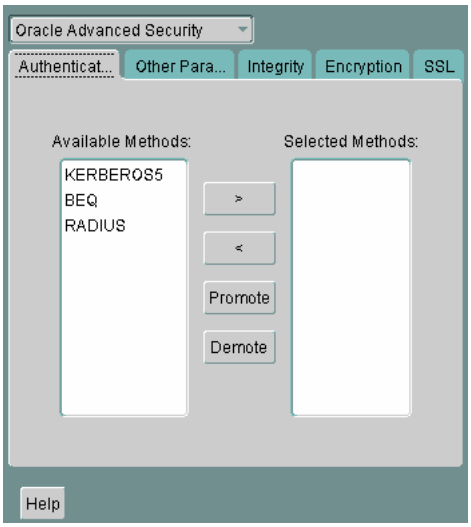
Unless otherwise indicated, perform these configuration tasks by using Oracle Net Manager or by using any text editor to modify the `sqlnet.ora` file.

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



1. Click the **Authentication** tab.
2. From the Available Methods list, select **RADIUS**.
3. Select the right-arrow (>) to move RADIUS to the **Selected Methods** list. Move any other methods you want to use in the same way.
4. 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.
5. 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 less than or equal to 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.

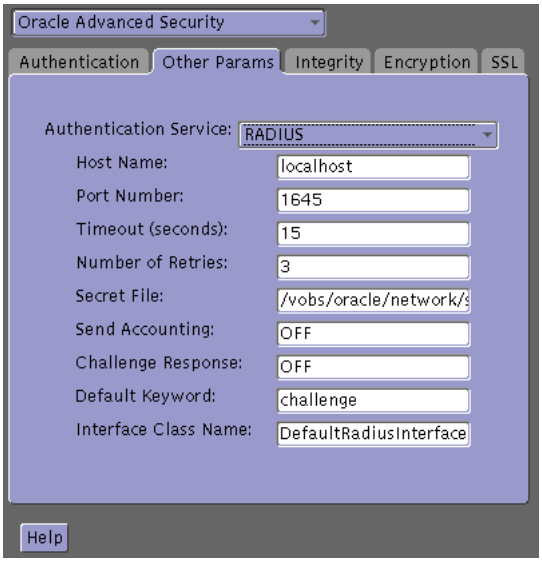
> **See Also:** The RADIUS server administration documentation, for information about obtaining the secret key

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



1. From the Authentication Service list, select **RADIUS**.
2. In the **Host Name** field, accept the **localhost** as the default primary RADIUS server, or enter another host name.
3. Ensure that the default value of the **Secret File** field is valid.
4. 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}
```

Note: The `IP_address` can either be an Internet Protocol Version 4 (IPv4) or Internet Protocol Version 6 (IPv6) address. The RADIUS adapter supports both IPv4 and IPv6 based servers.

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:

```
OS_AUTHENT_PREFIX=""
```

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.
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.

Field	Description
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.

---

1. 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).
2. From the Authentication Service list, select **RADIUS**.
3. In the **Challenge Response** field, enter **ON** to enable challenge-response.
4. 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.

---

1. 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`
2. 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 User's Guide*

2. Enter the same *username* 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.EXAMPLE.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.

1. 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@database_alias
Enter password: password
```

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 Security 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 Key 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:** Your Kerberos version 5 source distribution for notes about building and installing 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 DNS name of the system on which Oracle Database is running.
REALM	This is the name of the Kerberos realm with which the service principal is registered. 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 `EXAMPLE.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.example.com` and the realm is `EXAMPLE.COM`. The principal name then is:

```
oracle/dbserver.example.com@EXAMPLE.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.example.com@EXAMPLE.COM` to the list of server principals known by Kerberos, enter the following:

```
kadmin.local:addprinc -randkey oracle/dbserver.example.com@EXAMPLE.COM
```

Task 3: Extract a Service Key Table from Kerberos

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

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

1. Enter the following to extract the service key table:

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

Entry for principal oracle/dbserver.example.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 key 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 default realm of the Kerberos server. `kadmin.local` is connected to the Kerberos server running on the `localhost`.

3. If the Kerberos service key table is on the same system as the Kerberos client, you can move it. If the service key 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 key 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 key 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:

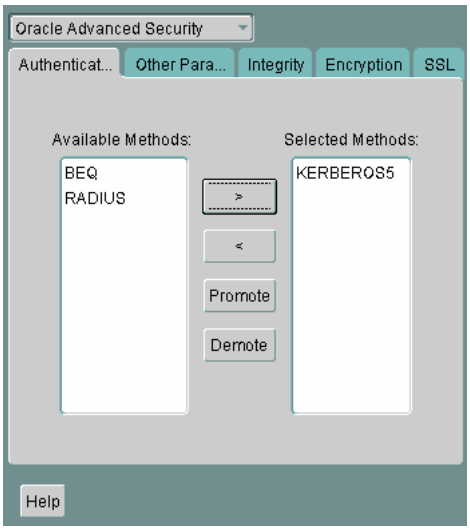
- Step 1: Configure Kerberos on the Client and on the Database Server
- Step 2: Set the Initialization Parameters
- Step 3: Set sqlnet.ora Parameters (optional)

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)



1. Click the **Authentication** tab.
2. From the Available Methods list, select **KERBEROS5**.
3. Move KERBEROS5 to the Selected Methods list by clicking the right arrow (>).
4. 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.
5. 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

As Kerberos user names can be long, and Oracle user names are limited to 30 characters, Oracle recommends that you set the value of `OS_AUTHENT_PREFIX` to null in the initialization parameter file.

```
OS_AUTHENT_PREFIX=""
```

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

Note: Oracle Database 11*g* Release 2 (11.2) enables you to create external database users that have Kerberos user names of more than 30 characters. See "Task 9: Create an Externally Authenticated Oracle User" on page 7-8 for more information.

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 use the following formats to specify a value for `SQLNET.KERBEROS5_CC_NAME`:
	■ `SQLNET.KERBEROS5_CC_NAME=`*`complete_path_to_cc_file`*
	For example:
	`SQLNET.KERBEROS5_CC_NAME=/tmp/kcache`
	`SQLNET.KERBEROS5_CC_NAME=D:\tmp\kcache`
	■ `SQLNET.KERBEROS5_CC_NAME=FILE:`*`complete_path_to_cc_file`*
	For example:
	`SQLNET.KERBEROS5_CC_NAME=FILE:/tmp/kcache`
	■ `SQLNET.KERBEROS5_CC_NAME=OSMSFT:`
	Use this value if you are running Windows and using a Microsoft KDC.
	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`

Attribute	Description
Parameter:	SQLNET.KERBEROS5_CONF_MIT=[TRUE|FALSE]
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

Support for Credential Cache Type 4 Format

Oracle Database now supports and recognizes the credential cache type 4 format. This feature is useful for those environments that use newer versions of MIT Kerberos 5 (1.3.x and above) utilities.

To use this feature, you need to set the following parameter in the sqlnet.ora file:

```
SQLNET.KERBEROS5_CONF_MIT = TRUE
```

Your Kerberos configuration file (krb5.conf) should have the following settings:

```
...
[libdefaults]
...
kdc_timesync = 1
ccache_type = 4
```

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 EXAMPLE.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@EXAMPLE.COM": (password does not display)
Re-enter password for principal: "krbuser@EXAMPLE.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@EXAMPLE.COM" IDENTIFIED EXTERNALLY;
SQL> GRANT CREATE SESSION TO "KRBUSER@EXAMPLE.COM";
```

If the user's Kerberos principal name is longer than 30 characters, and up to 1024 characters, then create the user as follows:

```
SQL> CREATE USER db_user_name IDENTIFIED EXTERNALLY AS 'kerberos_principal_name'
```

For example:

```
SQL> CREATE USER KRBUSER IDENTIFIED EXTERNALLY AS 'KerberosUser@EXAMPLE.COM';
```

Note: The database administrator should ensure that two database users are not identified externally by the same Kerberos principal name.

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@EXAMPLE.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

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 2wld6h20m30s` 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.
`-e`	Specifies a number representing the Kerberos encryption type to use. This option can be used to request a particular Kerberos encryption type key for the session. If you specify more than one encryption type, then the KDC chooses the common and strongest encryption type from the list. The following values are allowed: ■ 1 for `DES-CBC-CRC` ■ 3 for `DES-CBC-MD5` ■ 16 for `DES3-CBC-SHA1` ■ 18 for `AES256-CTS` ■ 23 for `RC4-HMAC` The following example requests for the `DES-CBC-CRC` and `DES3-CBC-SHA1` encryption types: `okinit -e 1 -e 16 krbuser@REALM` Note that you can repeat the option to request multiple encryption types.
`-?`	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/EXAMPLE.COM@EXAMPLE.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 User'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

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.example.com`.

- **`krb.conf`** file

  For example:

```
SALES3854.US.EXAMPLE.COM
SALES3854.US.EXAMPLE.COM sales3854.us.example.com admin server
```

- **`krb5.conf`** file

  For example:

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

- **`krb5.realms`** file

  For example:

```
us.example.com SALES.US.EXAMPLE.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 environment, ensure that the first `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.example.com`, then use Active Directory to create a user with the user name `sales3854.us.example.com` and the password `oracle`.

   Note: Do not create a user as `host/hostname.dns.com`, such as `oracle/sales3854.us.example.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.

1. 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.example.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.

2. 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://technet.microsoft.com/hi-in/windowsserver/2000/bb735396(en-us).aspx`

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 7-8 to create an externally authenticated Oracle user. Ensure that the username is created in all uppercase characters. For example, `ORAKRB@SALES.US.EXAMPLE.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.

Note: The user does not need to explicitly request for an initial ticket, using the `okinit` command, when using the Windows native cache.

If the Oracle client is running on Windows 2000 or later, the Kerberos ticket is automatically retrieved when the user logs in to Windows.

See Also: Microsoft documentation for details on the `Kerbtray.exe` utility, which can be used to display Kerberos ticket information for a system

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.
  - Ensure that the `TNS_ADMIN` environment variable is pointing to the directory containing the `sqlnet.ora` configuration file.
- 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

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

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 session key 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). The CA periodically publishes

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. Oracle Internet Directory
2. **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 8-23 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 11*g* Release 2 (11.2) 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 11*g* Release 2 (11.2) 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, SafeNET and nCipher devices are certified with Oracle Advanced Security

---

See Also: "Configuring Your System to Use Hardware Security Modules" on page 8-32 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

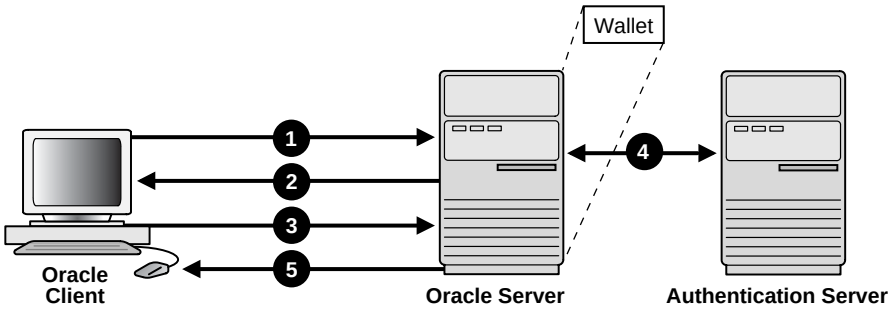
Figure 1–4 on page 1-10, which displays the Oracle Advanced Security implementation architecture, shows that Oracle Advanced Security operates at the **session layer** on top of SSL and uses TCP/IP at the **transport layer**. This separation of functionality lets you employ SSL concurrently with other 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.

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.
- Multi-threaded clients currently cannot use SSL.

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 8-32 for information about improving SSL performance with hardware accelerators
- "Enabling SSL" on page 8-8

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 SSL Cipher Suites

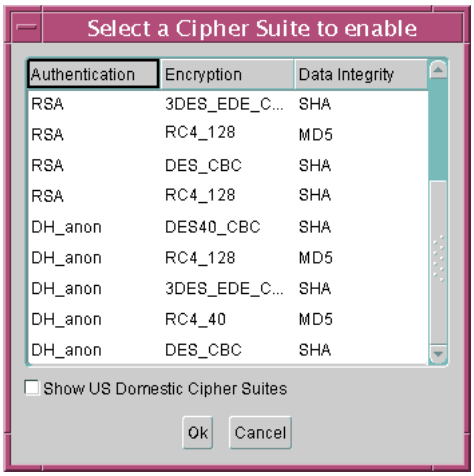
Cipher Suites	Authentication	Encryption	Data Integrity
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES EDE CBC	SHA-1
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4 128	SHA-1
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4 128	MD5
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES CBC	SHA-1
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH anon	3DES EDE CBC	SHA-1
SSL_DH_anon_WITH_RC4_128_MD5	DH anon	RC4 128	MD5
SSL_DH_anon_WITH_DES_CBC_SHA	DH anon	DES CBC	SHA-1
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:

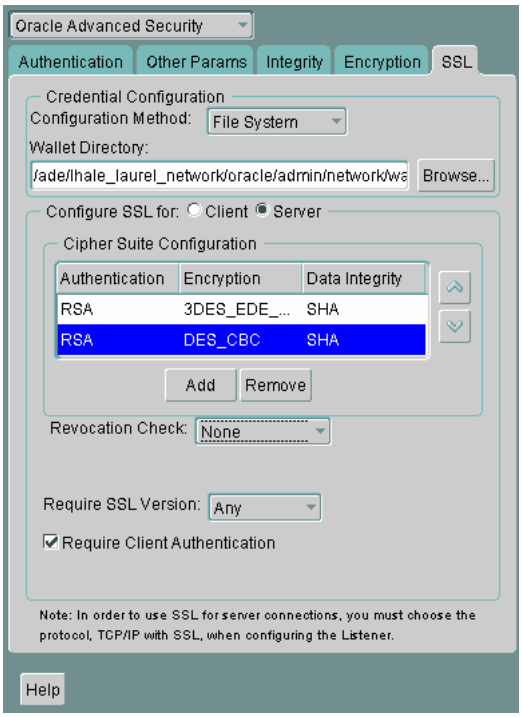
1. Click **Add**. A dialog box displays available cipher suites (Figure 8–2).

Figure 8–2 SSL Cipher Suites Window



1. 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)



1. Use the up and down arrows to prioritize the cipher suites.
2. 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` or the `listener.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)

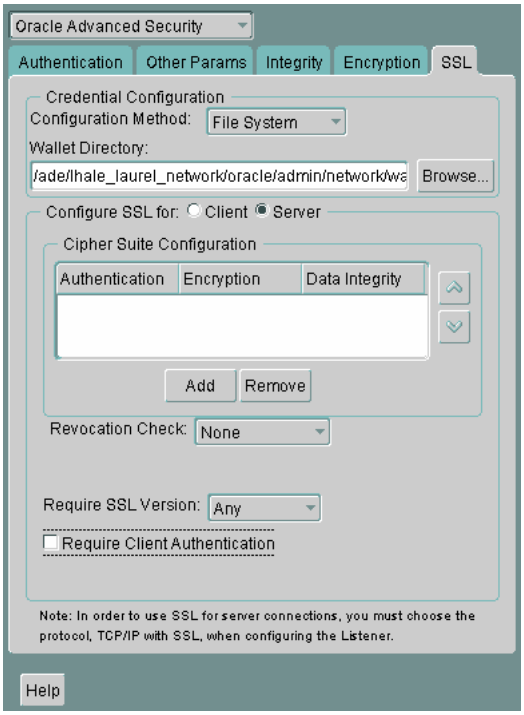
The `SSL_CLIENT_AUTHENTICATION` parameter in the `sqlnet.ora` file controls whether the client is authenticated using SSL. The default value is `TRUE`.

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** (Figure 8–4).
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 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 8-23 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 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.

1. 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.
2. 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.example.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

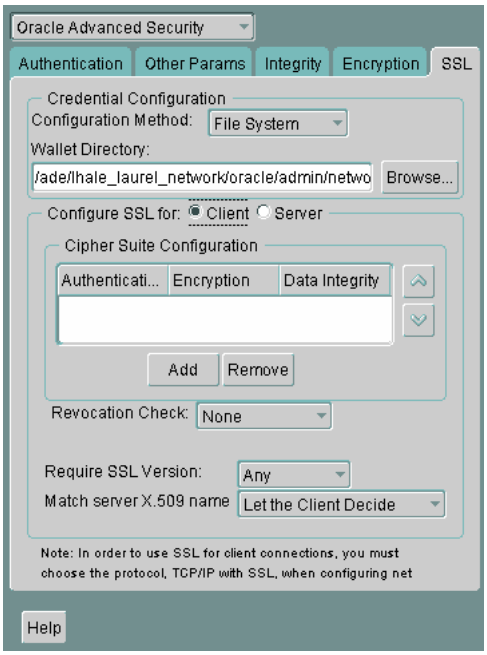
```
LISTENER=
(DESCRIPTION_LIST=
(DESCRIPTION=
(ADDRESS= (PROTOCOL = tcps) (HOST = finance_server) (PORT = 1575))))
```

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)



1. Click the **SSL** tab.
2. Select **Configure SSL for: Client**.
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.
4. 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 recommends selecting YES for this option so that connections are refused when there is a mismatch.

5. 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 8-32 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.

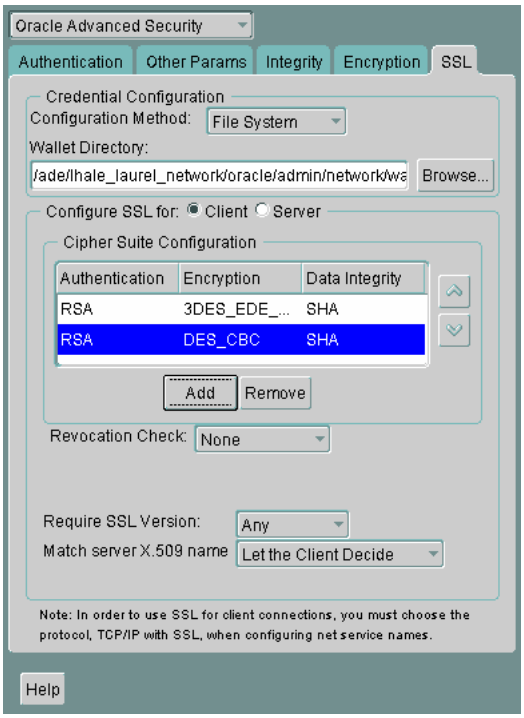
Table 8–1 lists the SSL cipher suites supported in the current release of Oracle Advanced Security. These cipher suites are set by default when you install Oracle Advanced Security. The table also lists the authentication, encryption, and data integrity types each cipher suite uses.

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)



1. Use the up and down arrows to prioritize the cipher suites.
2. 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@net_service_name
Enter password: password
```

> **See Also:** "Certificate Validation with Certificate Revocation Lists" on page 8-23 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.

If the auto login feature is not being used, then enter the correct password.

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.

  If the error still persists, then enable Oracle Net tracing and attempt the connection again. Contact Oracle customer support with the trace output.

  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 8-21

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 8-27

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 8-28

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.

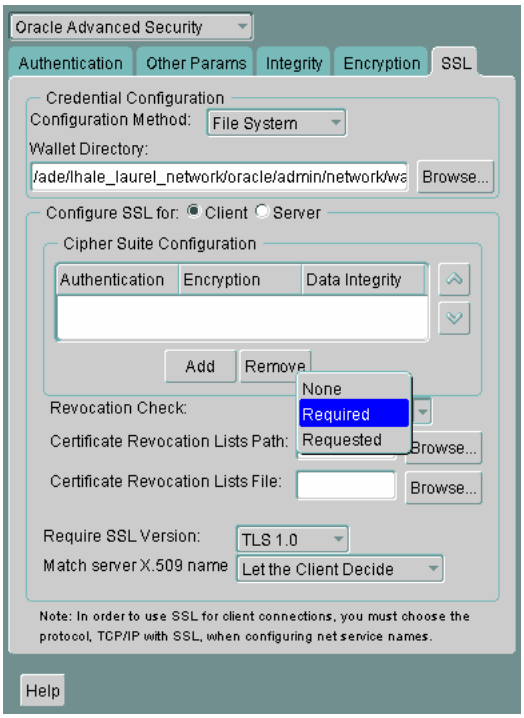
Configuring Certificate Validation with Certificate Revocation Lists

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.

---

1. (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 8-27

---

1. (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.

---

1. 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 8-30 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 to 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.example.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, "Listing CRLs Stored in Oracle Internet Directory" on page 8-29.

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 "Uploading CRLs to Oracle Internet Directory" on page 8-28 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 8-28 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 8-27
   - CRLs stored in the directory, refer to "Uploading CRLs to Oracle Internet Directory" on page 8-28

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 Distribution Point (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 8-27

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 8-28

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

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 cryptographic processing that frees your server to respond to other requests
- Secure private key storage on the device
- Allow key administration 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

  The following platform-specific PKCS#11 library is required:

  - `libcknfast.so` library for UNIX 32-Bit
  - `libcknfast-64.so` library for UNIX 64-Bit
  - `cknfast.dll` library for Windows

> **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:

- `/opt/nfast` for UNIX
- `C:\nfast` for Windows

The nCipher PKCS #11 library is located at the following location for typical installations:

- `/opt/nfast/toolkits/pkcs11/libcknfast.so` for UNIX 32-Bit
- `/opt/nfast/toolkits/pkcs11/libcknfast-64.so` for UNIX 64-Bit
- `C:\nfast\toolkits\pkcs11\cknfast.dll` for Windows

> **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).

Configuring Your System to Use SafeNET Hardware Security Modules

Hardware security modules made by SafeNET Incorporated 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

> **Note:** You must contact your SafeNET representative to obtain certified hardware and software to use with Oracle Advanced Security.

Oracle Components Required To Use a SafeNET Luna SA Hardware Security Module

To use a SafeNET Luna SA hardware security module, you need the following components

- SafeNET Luna SA Hardware Security Module
- Supporting SafeNET Luna SA PKCS #11 library

The following platform-specific PKCS#11 library is required:

- `libCryptoki2.so` library for UNIX
- `cryptoki.dll` library for Windows

> **Note:** You must contact your SafeNET 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 a SafeNET hardware security module with Oracle Advanced Security.

About Installing a SafeNET Hardware Security Module

To use the secure accelerator, you must provide the absolute path to the directory that contains the SafeNET PKCS #11 library (including the library name) when you create the wallet using Oracle Wallet Manager. This enables the library to be loaded at runtime. Typically, the SafeNET Luna SA client is installed at the following location:

- `/usr/lunasa` for UNIX
- `C:\Program Files\LunaSA` for Windows

The SafeNET Luna SA PKCS #11 library is located at the following location for typical installations:

- `/usr/lunasa/lib/libCryptoki2.so` for UNIX
- `C:\Program Files\LunaSA\cryptoki2.dll` for Windows

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 and SafeNET documentation for more information about troubleshooting nCipher and SafeNET devices

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 key**s 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 **certificate**s 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 pairs** 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 a user from accidentally overwriting 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 9-14

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, SafeNET and nCipher hardware have 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) *$ORACLE_HOME*/owm/wallets/*username* if the ORACLE_HOME environment variable has been set.

  ./owm/wallets/*username* if the ORACLE_HOME environment variable is not set.

- (WINDOWS) *ORACLE_HOME*\owm\wallets*username* if the ORACLE_HOME environment variable has been set.

  .\owm\wallets*username* if the ORACLE_HOME environment variable is not set.

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, encrypted with the 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 Login**. 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
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: "Exporting a User Certificate Request" on page 9-20

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 certificates**, 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 system 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

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 All 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

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@net_service_name
Enter password: password
```

For example:

```
% sqlplus hr@emp
Enter password: password
```

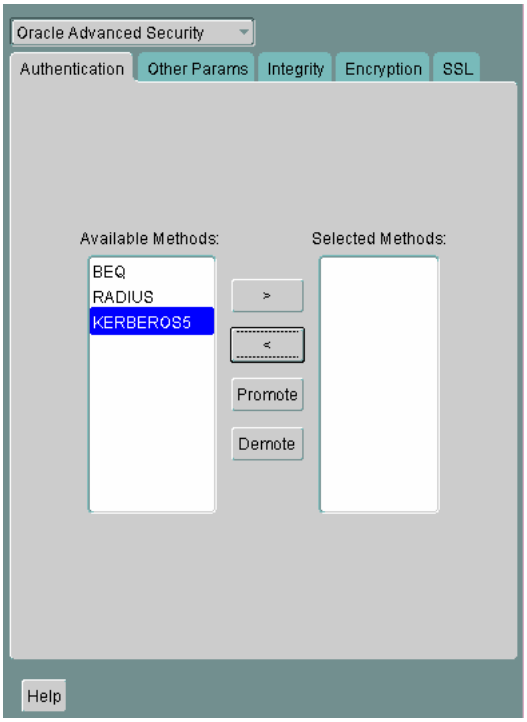
Note: You can configure multiple authentication methods, including both externally authenticated users and password authenticated users, on a single database.

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 = (KERBEROS5, RADIUS)
   ```

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
- 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)
```

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 User'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 =
                                     /etc/ORACLE/WALLETS/oracle)))
```

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.example.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

See Also:

- Chapter 4, "Configuring Network Data Encryption and Integrity for Oracle Servers and Clients"
- "About Activating Encryption and Integrity" on page 4-4

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*])

Table A–6 (Cont.) SQLNET.ENCRYPTION_TYPES_SERVER Parameter Attributes

Attribute	Description
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).
Default Setting	If no algorithms are defined in the local `sqlnet.ora` file, all installed algorithms are used in a negotiation.

Table A–7 (Cont.) SQLNET.ENCRYPTION_TYPES_CLIENT Parameter Attributes

Attribute	Description
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	■ SHA1: 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	■ SHA1: 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.

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`	`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
```

Initialization File Parameters

```
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. Parameters like `SSL_CIPHER_SUITES` and `SSL_VERSION` can also be configured using the `listener.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, 1.0 (for TLS), 2.0, and 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:

> **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`
- `ENCRYPTION_TYPES_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)
```

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

Oracle Advanced Security SSL adapter can be configured to run in FIPS mode by setting the `SSLFIPS_140` parameter to `TRUE` in the `fips.ora` file.

```
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 `fips.ora` file is either located in the *`$ORACLE_HOME`*`/ldap/admin` directory, or is pointed to by the `FIPS_HOME` environment variable. This procedure can be repeated in any Oracle home for any database server or client.

Note: The `SSLFIPS_140` parameter replaces the `SQLNET.SSLFIPS_140` parameter used in Oracle Database 10*g* Release 2 (10.2). The parameter needs to be set in the `fips.ora` file, and not the `sqlnet.ora` file.

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 following cipher suites 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_AES_256_CBC_SHA
- SSL_RSA_WITH_AES_128_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` or the `listener.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 8-10 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, from the command line. This enables you to automate these tasks using 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

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

The `orapki` utility provides a convenient, lightweight way to create signed certificates for testing purposes.

To create a signed certificate for testing purposes, use the following command:

```
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, use the following command:

```
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, Viewing, and Modifying 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, Viewing, and Modifying Wallets with orapki

This section contains the following topics:

- Creating a PKCS#12 Wallet
- Creating an Auto Login Wallet
- Viewing a Wallet

- Modifying the Password for a Wallet

Creating a PKCS#12 Wallet

To create an Oracle PKCS#12 wallet (`ewallet.p12`), use the following command:

```
orapki wallet create -wallet wallet_location [-pwd password]
```

This command prompts 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`.

> **Note:** For security reasons, Oracle recommends that you do not specify the password at the command line. You should supply the password when prompted to do so.

Creating an Auto Login Wallet

To create an auto login wallet (`cwallet.sso`) that does not need a password, use the following command:

```
orapki wallet create -wallet wallet_location -auto_login_only
```

This command creates an auto login wallet (`cwallet.sso`) that does not need a password to open. You can also modify or delete the wallet without using a password. File system permissions provide the necessary security for such auto login wallets.

You can also create an auto login wallet that is associated with a PKCS#12 wallet. The auto login wallet does not need a password to open. However, you must supply the password for the associated PKCS#12 wallet in order to modify or delete the wallet. Any update to the PKCS#12 wallet also updates the associated auto login wallet.

To create an auto login wallet (`cwallet.sso`) that is associated with a PKCS#12 wallet (`ewallet.p12`), use the following command:

```
orapki wallet create -wallet wallet_location -auto_login [-pwd password]
```

This command creates a wallet with auto login enabled (`cwallet.sso`) and associates it with a PKCS#12 wallet (`ewallet.p12`). The command prompts you to enter the password for the PKCS#12 wallet, if no password has been specified at the command line.

> **Note:** For security reasons, Oracle recommends that you do not specify the password at the command line. You should supply the password when prompted to do so.

If the `wallet_location` already contains a PKCS#12 wallet, then auto login is enabled for it. You must supply the password for the existing PKCS#12 wallet in order to enable auto login for it.

If the wallet_location does not contain a PKCS#12 wallet, then a new PKCS#12 wallet is created. You must specify a password for the new PKCS#12 wallet.

If you wish to turn the auto login feature off for a PKCS#12 wallet, then use Oracle Wallet Manager.

> **See Also:** "Using Auto Login" on page 9-14 for more information

You can also choose to create a local auto login wallet. Local auto login wallets cannot be moved to another computer. They must be used on the host on which they are created.

A local auto login wallet does not need a password to open. However, you must supply the password for the associated PKCS#12 wallet in order to modify or delete the wallet. Any update to the PKCS#12 wallet also updates the associated auto login wallet.

To create a local auto login wallet, use the following command:

```
orapki wallet create -wallet wallet_location -auto_login_local [-pwd password]
```

This command creates an auto login wallet (`cwallet.sso`) that is local to both the computer on which it is created and the user who created it. It associates it with a PKCS#12 wallet (`ewallet.p12`). The command prompts you to enter the password for the PKCS#12 wallet, if no password has been specified at the command line.

> **Note:** For security reasons, Oracle recommends that you do not specify the password at the command line. You should supply the password when prompted to do so.

Viewing a Wallet

To view an Oracle wallet, use the following command:

```
orapki wallet display -wallet wallet_location
```

This command 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.

Modifying the Password for a Wallet

To change the wallet password, use the following command:

```
orapki wallet change_pwd -wallet wallet_location [-oldpwd password ] [-newpwd
password]
```

This command changes the current wallet password to the new password. The command prompts you for the old and new passwords if no password is supplied at the command line.

> **Note:** For security reasons, Oracle recommends that you do not specify the password options at the command line. You should supply the password when prompted to do so.

Adding Certificates and Certificate Requests to Oracle Wallets with orapki

To add a certificate request to an Oracle wallet, use the following command:

```
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-6 for more information

To add a trusted certificate to an Oracle wallet, use the following command:

```
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, use the following command:

```
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, use the following command:

```
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.

> **Note:** For security reasons, Oracle recommends that you do not specify the password at the command line. You should supply the password when prompted to do so.

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]
```

> **Note:** For security reasons, Oracle recommends that you do not specify the password at the command line. You should supply the password when prompted to do so.

Exporting Certificates and Certificate Requests from Oracle Wallets with orapki

To export a certificate from an Oracle wallet, use the following command:

```
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, use the following command:

```
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/root -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 enables 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 8-28 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-11
- 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 8-28 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 8-28 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, use the following command:

```
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-13

To add trusted certificates, use the following command:

```
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, use the following command:

```
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, use the following command:

```
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|-auto_login_local]
```

- 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

- The `-auto_login_local` parameter creates a local auto login wallet, or it turns on local automatic login for the wallet specified with the `-wallet` option.

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, use the following command:

```
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, use the following command:

```
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

Entrust-enabled Oracle Advanced Security provides:

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.

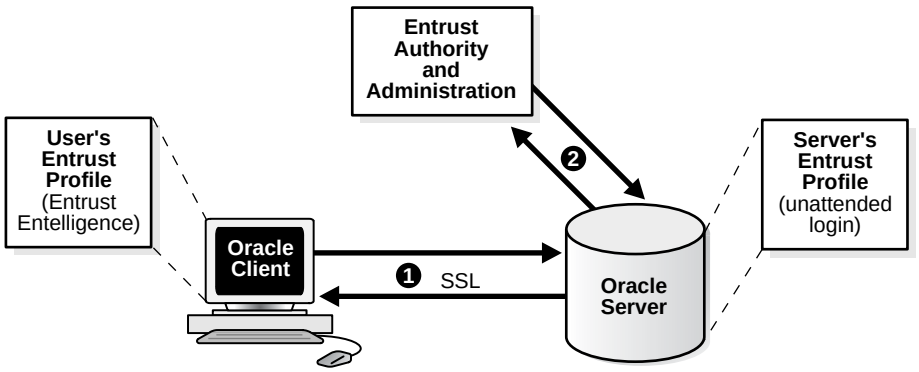
Entrust Authentication Process

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

1. The administrator enters the user's name and password.
2. The Entrust Authority creates the profile, or `.epf` file.
3. 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

1. The user receives a secure e-mail notification from the administrator that contains a reference number, authorization code, and expiration date.
2. The user navigates to the Create Entrust Profiles screen in Entrust Entelligence Desktop Manager as follows:

   Start, **Programs**, **Entrust**, **Entrust Profiles**, **Create Entrust Profiles**

3. 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 11*g* Release 2 (11.2), 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.

1. Select the Entrust icon on the system tray to open the Entrust_Login dialog box.
2. 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
```

1. 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.

2. 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)
)
)
```

1. 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.

2. Start the Oracle database instance.

Note: For all Windows environments, Oracle 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 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**

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 **certificates**. 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 name, net service name, or service name that resolves to a **connect descriptor**. 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 want to connect.

For example:

```
CONNECT username@connect_identifier
Enter password: password
```

connect string

Information the user passes to a **service** to connect, such as **user name**, password and **net service name**. For example:

```
CONNECT username@net_service_name
Enter password: password
```

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)

An older Federal Information Processing Standards encryption algorithm superseded by the Advanced Encryption Standard (AES).

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 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.

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.

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.

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.

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

A simple name for a service that resolves to a connect descriptor. Users initiate a connect request by passing a user name and password along with a net service name in a connect string for the service to which they want to connect:

```
CONNECT username@net_service_name
Enter password: password
```

Depending on your needs, net service names can be stored in a variety of places, including:

- Local configuration file, `tnsnames.ora`, on each client
- Directory server
- External naming service, such as NIS

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 **National Institute of Standards and Technology (NIST)**

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 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 domains** 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 chains**.

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 communicate 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).

The **Transport Layer Security (TLS)** protocol is the successor to the SSL protocol.

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 key table

In Kerberos authentication, a service key 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

A service ticket is trusted information used to authenticate the client, to a specific service or server, for a predetermined period of time. It is obtained from the **KDC** using the **initial ticket**.

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 **initial ticket** and **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.

Transport Layer Security (TLS)

An industry standard protocol for securing network connections. The TLS protocol is a successor to the SSL protocol. It provides authentication, encryption, and data integrity using public key infrastructure (PKI). The TLS protocol is developed by the Internet Engineering Task Force (IETF).

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

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W

X