

# Oracle® Cryptographic Toolkit

Programmer's Guide

Release 2.0.4

October 1997

Part No. A54082-02

---

Oracle® Cryptographic Toolkit Programmer's Guide

Part No. A54082-02

Release 2.0.4

Copyright © 1996, 1997, Oracle Corporation. All rights reserved.

Printed in the U.S.A

Primary Author: Gilbert Gonzalez

Contributing Authors: Andre Srinivasan, Richard Wessman

Contributors: Paul Lambert, Patricia Markee, Kendall Scott, Sandy Venning

**The programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle disclaims liability for any damages caused by such use of the Programs.**

This Program contains proprietary information of Oracle Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright patent and other intellectual property law. Reverse engineering of the software is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free.

If this Program is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

**Restricted Rights Legend** Programs delivered subject to the DOD FAR Supplement are 'commercial computer software' and use, duplication and disclosure of the Programs shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are 'restricted computer software' and use, duplication and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-14, Rights in Data -- General, including Alternate III (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.



This product contains security software from RSA Data Security, Inc. Copyright 1994 RSA Data Security, Inc. All rights reserved. This version supports International Security with RSA Public Key Cryptography, MD2, MD5, and RC4.

This product contains encryption and/or authentication engines from RSA Data Security, Inc. Copyright 1996 RSA Data Security, Inc. All rights reserved.

Oracle and SQL\*Plus are registered trademarks of Oracle Corporation, Redwood City, California. Oracle Security Server, Oracle Enterprise Manager, Oracle Call Interface, Net8, PL/SQL, and Oracle8 are trademarks of Oracle Corporation, Redwood City, California.

All other product or company names are used for identification purposes only, and may be trademarks of their respective owners.

---

# Preface

## Purpose

The *Oracle Cryptographic Toolkit Programmer's Guide* provides independent application programmers with programming interfaces to the services provided by the Oracle Security Server.

## Intended Audience

The *Oracle Cryptographic Toolkit Programmer's Guide* is designed to be used by both Oracle and non-Oracle application programmers who require an interface to the services provided by the Oracle Security Server. This document assumes that the reader is familiar with the functionality of the Oracle Security Server, as described in the *Oracle Security Server Guide*.

## Structure

This manual contains three parts, **seven** chapters, and **two** appendices.

### Part I

#### **Concepts**

The Concepts chapters contain the following information:

### Chapter 1

#### **Overview**

Provides definitions of the Oracle Security Server and the Oracle Cryptographic Toolkit and states the purpose of this Programmer's Guide

### Chapter 2

#### **Data Types**

Discusses public functions, data types, and data structures

### Chapter 3

#### **Concepts**

Discusses general security concepts and Oracle Cryptographic Toolkit concepts

Chapter 4	<b>Using the Oracle Cryptographic Toolkit</b> Shows you how to program using the Oracle Cryptographic Toolkit
Chapter 5	<b>Random Number Generator</b> Shows users how to generate random data for their applications
Part II	<b>Reference</b> The Reference chapters contain the following information:
Chapter 6	<b>OCI Functions for C</b> Describes each Oracle Call Interface (OCI) function in the Oracle Cryptographic Toolkit
Chapter 7	<b>PL/SQL Functions</b> Describes each PL/SQL function in the Oracle Cryptographic Toolkit
Part III	<b>Appendices</b> The Appendices contain reference information, including sample C programs, sample PL/SQL programs, and OCI - API function mappings.
Appendix A	<b>Sample PL/SQL Code</b> Contains sample PL/SQL programs
Appendix B	<b>OCI - API Mappings</b> Lists each OCI function that is directly mapped to an API function
Glossary	Lists terms, abbreviations, and definitions used in this guide

## Related Documents

For more information, see the following manuals:

- *Oracle8<sup>TM</sup> Server Application Developer's Guide*
- *Oracle Security Server<sup>TM</sup> Guide*
- *Programmer's Guide to the Oracle Call Interface<sup>TM</sup>*

## Conventions

The following conventions are used in this manual:

Convention	Meaning
monospace	Code examples and data type names are displayed in monospace font.
<i>italic</i>	Names of related manuals are displayed in italic font.



---

---

# Send Us Your Comments

## Oracle® Cryptographic Toolkit Programmer's Guide

Part No. A54082-02

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available).

You can send comments to us in the following ways

- electronic mail: [ossdoc@us.oracle.com](mailto:ossdoc@us.oracle.com)
- postal service:  
Oracle Corporation  
Documentation Manager: Enterprise Application Services  
500 Oracle Parkway  
Redwood City CA 94065  
USA

If you would like a reply, please give your name, address, and telephone number below.

---

---

---





---

---

# Contents

<b>Preface</b> .....	iii
----------------------	-----

<b>Send Us Your Comments</b> .....	vii
------------------------------------	-----

## **Part I Concepts**

### **1 Overview**

1.1	What is the Oracle Security Server?.....	2
1.1.1	Oracle Security Server Features .....	2
1.2	What is the Oracle Cryptographic Toolkit? .....	4
1.3	Oracle Cryptographic Toolkit Functional Layers.....	5
1.3.1	API Layer.....	5
1.3.2	Cryptographic Engine Functions.....	5
1.3.3	Persona/Identity Functions.....	6
1.3.4	Wallet Functions.....	6
1.4	Oracle Cryptographic Toolkit Elements .....	7
1.4.1	Identity.....	7
1.4.2	Trusted Identity .....	8
1.4.3	Persona.....	8
1.4.4	Wallet .....	9
1.5	Types of Interfaces .....	10
1.5.1	Oracle Call Interface .....	10
1.5.2	PL/SQL Interface .....	10

## 2 Data Types

2.1	Data Types.....	2
2.1.1	Name Prefixes .....	2
2.1.2	Crypto Engine State .....	2
2.1.3	Crypto Engine Functions.....	3
2.1.4	Identity Type .....	3
2.1.5	Cipher Types .....	3
2.1.6	TDU Formats.....	4
2.1.7	Validate State .....	4
2.1.8	Unique ID .....	4
2.1.9	Timestamp .....	4
2.2	Data Structures.....	5
2.2.1	nzttBufferBlock .....	5
2.2.2	nzttWallet .....	6
2.2.3	nzttPersona.....	6
2.2.4	nzttIdentity .....	6

## 3 Concepts

3.1	Security Concepts .....	2
3.2	Oracle Cryptographic Toolkit Concepts .....	4

## 4 Using the Oracle Cryptographic Toolkit

4.1	Basic Oracle Cryptographic Toolkit Program Flow .....	2
4.2	A Programming Example.....	2
4.2.1	Using the Oracle Cryptographic Toolkit.....	3
4.2.2	An Example: Generating a detached signature for an array of bytes.....	5

## 5 Random Number Generator

5.1	Overview.....	2
5.2	Functions.....	2
5.3	Example.....	2

## Part II Reference

## 6 OCI Functions for C

6.1	OCISecurityInitialize.....	2
6.2	OCISecurityTerminate.....	3
6.3	OCISecurityOpenWallet.....	4
6.4	OCISecurityCloseWallet.....	5
6.5	OCISecurityOpenPersona .....	6
6.6	OCISecurityClosePersona .....	7
6.7	OCISecuritySign .....	8
6.8	OCISecurityVerify .....	9
6.9	OCISecurityValidate .....	11
6.10	OCISecuritySignDetached.....	12
6.11	OCISecurityVerifyDetached .....	13
6.12	OCISecurityHash.....	15
6.13	OCISecuritySeedRandom.....	16
6.14	OCISecurityRandomBytes .....	17
6.15	OCISecurityRandomNumber .....	18
6.16	OCISecurityInitBlock .....	19
6.17	OCISecurityReuseBlock.....	20
6.18	OCISecurityPurgeBlock.....	21
6.19	OCISecuritySetBlock .....	22

## 7 PL/SQL Functions

7.1	General Purpose Procedures .....	2
7.1.1	Procedures Used by Applications That Use the Wallet.....	3
7.2	Digital Signature.....	7
7.2.1	Sign .....	8
7.2.2	Verify.....	9
7.2.3	SignDetached .....	10
7.2.4	VerifyDetached.....	11
7.3	Hash.....	12
7.3.1	KeyedHash.....	13
7.3.2	Hash .....	14
7.4	Random Number Generation.....	15

**Part III Appendices**

**A Sample PL/SQL Code**

A.1	Sample PL/SQL Program .....	2
-----	-----------------------------	---

**B OCI - API Mappings**

B.1	Mappings .....	2
B.1.1	Overview .....	2
B.1.2	OCI - API Mappings .....	2
B.2	OCI - API Mapping Exceptions .....	3

**Glossary**

**Index**

**Figures**

1-1	Relationship between Toolkit and Services.....	4
1-2	Identity .....	8
1-3	Persona.....	9
1-4	Wallet .....	9
4-1	Oracle Cryptographic Toolkit Program Flow .....	2



## Tables

2-1	Data Types .....	2
2-2	Data Structures and Descriptions.....	5
2-3	nzttBufferBlock.....	5
2-4	nzttWallet .....	6
2-5	nzttPersona.....	6
2-6	nzttIdentity .....	6
6-1	OCISecurityInitialize Handles.....	2
6-2	OCISecurityTerminate parameters.....	3
6-3	OCISecurityOpenWallet parameters .....	4
6-4	OCISecurityCloseWallet parameters .....	5
6-5	OCISecurityOpenPersona parameters.....	6
6-6	OCISecurityOpenPersona errors .....	6
6-7	OCISecurityClosePersona parameters.....	7
6-8	OCISecurityClosePersona errors .....	7
6-9	OCISecuritySign parameters .....	8
6-10	OCISecurityVerify parameters .....	9
6-11	OCISecurityVerify errors.....	10
6-12	OCISecurityValidate parameters.....	11
6-13	OCISecurityValidate errors .....	11
6-14	OCISecuritySignDetached parameters .....	12
6-15	OCISecuritySignDetached errors .....	12
6-16	OCISecurityVerifyDetached parameters.....	13
6-17	OCISecurityVerifyDetached errors .....	14
6-18	OCISecurityHash parameters.....	15
6-19	OCISecurityHash errors .....	15
6-20	OCISecuritySeedRandom parameters .....	16
6-21	OCISecurityRandomBytes parameters.....	17
6-22	OCISecurityRandomNumber parameters .....	18
6-23	OCISecurityInitBlock parameters.....	19
6-24	OCISecurityReuseBlock parameters.....	20
6-25	OCISecurityPurgeBlock parameters.....	21
6-26	OCISecuritySetBlock parameters .....	22
7-1	PL/SQL Procedure and Function Descriptions.....	1
7-2	PROCEDURE OpenWallet .....	2
7-3	PROCEDURE OpenWallet .....	3
7-4	PROCEDURE CloseWallet .....	3
7-5	PROCEDURE DestroyWallet .....	3
7-6	PROCEDURE StorePersona .....	4
7-7	PROCEDURE OpenPersona .....	4

7-8	PROCEDURE ClosePersona .....	4
7-9	PROCEDURE RemovePersona .....	4
7-10	PROCEDURE CreatePersona.....	4
7-11	PROCEDURE RemoveIdentity .....	5
7-12	CreateIdentity .....	5
7-13	AbortIdentity .....	5
7-14	StoreTrustedIdentity.....	6
7-15	Validate.....	6
7-16	Sign parameters for raw data.....	8
7-17	Sign parameters for string data.....	8
7-18	Verify parameters for raw data.....	9
7-19	Verify parameters for string data.....	9
7-20	SignDetached parameters for raw data.....	10
7-21	SignDetached parameters for string data.....	10
7-22	VerifyDetached parameters for raw data.....	11
7-23	VerifyDetached parameters for string data.....	11
7-24	KeyedHash parameters for raw data .....	13
7-25	KeyedHash parameters for string data .....	13
7-26	Hash parameters for raw data .....	14
7-27	Hash parameters for string data .....	14
7-28	SeedRandom parameters for numeric data.....	15
B-1	OCI Function Names and Descriptions .....	2



# Part I

---

## Concepts

Part I, Concepts, contains the following chapters:

- Chapter 1, “Overview”
- Chapter 2, “Data Types”
- Chapter 3, “Concepts”
- Chapter 4, “Using the Oracle Cryptographic Toolkit”
- Chapter 5, “Random Number Generator”



---

# Overview

This chapter provides an overview of the Oracle Cryptographic Toolkit. The following topics are discussed:

- “What is the Oracle Security Server?”
- “What is the Oracle Cryptographic Toolkit?”
- “Oracle Cryptographic Toolkit Functional Layers”
- “Oracle Cryptographic Toolkit Elements”
- “Types of Interfaces”

## 1.1 What is the Oracle Security Server?

The Oracle Security Server is a portable security service that provides a centralized global authentication and authorization framework. It provides enterprise security by using public key cryptography to authenticate users, control user access to data, and protect sensitive data. These functions are achieved through the use of public key cryptography for encryption, digital signatures, and user authentication.

The Oracle Security Server uses X.509 v1 certificates as its authentication mechanism. The X.509 v1 certificate is a standard format for digitally signed certificates that contain information such as a user's identity, authorizations, and public key information.

X.509 v1 certificates are used to access secure network systems. Users obtain certificates so they can identify themselves, present their access credentials, and obtain a secure network connection with other cryptographically secure users or systems.

### 1.1.1 Oracle Security Server Features

The Oracle Security Server supports the following features.

#### **Certificate Authority Capability**

Customers can create their own certificate authorities (CA), create certificates for their users, and manage user authorizations and roles using the Oracle Security Server.

A certificate authority is a trusted entity that certifies that other entities are who they say they are. The CA is something of an electronic notary service: it generates and validates electronic IDs in the form of certificates that are the equivalent of driver's licenses or passports. The CA uses its private key to sign each certificate: an entity that receives a certificate from the CA can trust that signature just as a person in real life can trust the written signature of a notary.

#### **X.509 v1 Certificate**

A certificate is a message, signed by the CA, stating that a specified public key belongs to someone or something with a specified name. Certificates prevent someone from using a phony key to impersonate another party and also enable parties to exchange keys without contacting a CA for each authentication. Distributing keys in certificates is as reliable as if the keys were obtained directly from the CA. Certificate-based authentication works even when the security database server is temporarily unavailable.

The authentication mechanism used by the Oracle Security Server is based on the International Telecommunications Union (ITU) X.509 v1 certificates. X.509 is a standard format for digitally signed certificates. It conveys a user's identity and public key data.

### **Certificate Revocation List (CRL)**

A certificate revocation list (CRL) is a data structure, signed and timestamped by a CA, that lists all of the certificates created by the CA that have not yet expired but are no longer valid. CRLs are used to revoke security privileges and for compromise management.

A party retrieving a certificate from the CA can check one or more CRLs to see whether that certificate has been revoked. However, since checking a CRL incurs significant overhead, users may want to make these checks only for documents that are especially important, or they may want to limit themselves to only random, or periodic, checks of the CRLs.

### **Certificate Management Services**

The Oracle Security Server Manager provides the user with a graphical user interface that is used to create, store, and revoke certificates.

### **Oracle Enterprise Manager Administration Tool**

The Oracle Security Server Manager is implemented as an Oracle Enterprise Manager applet. This applet is a graphical user interface to the command line version of the Oracle Security Server Manager.

### **Command Line Administration Tools**

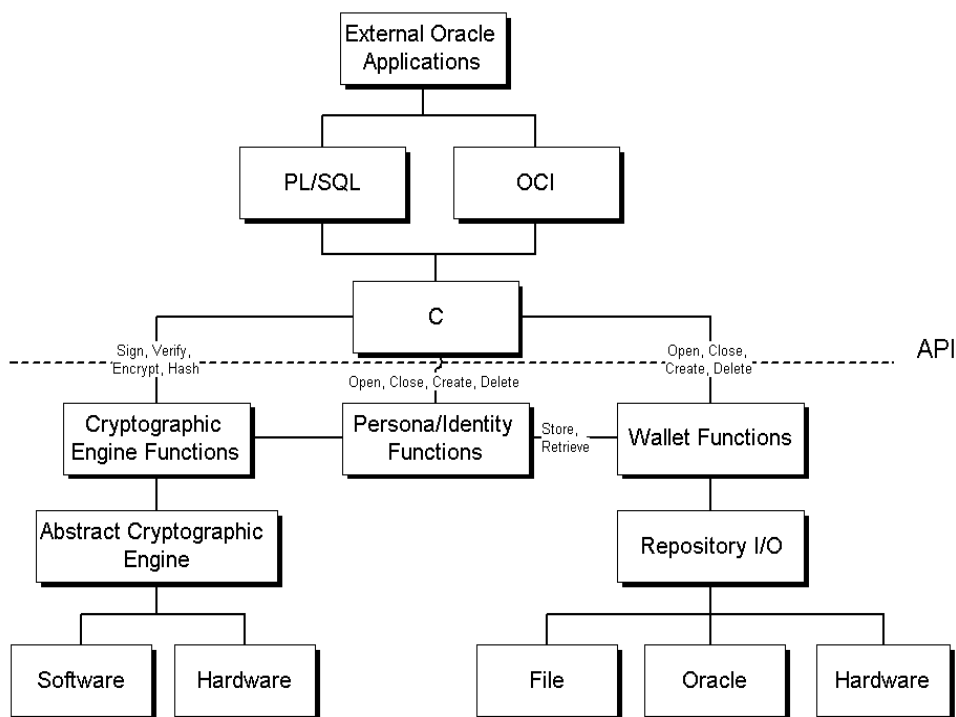
The Oracle Security Server Manager is also implemented as a set of command line tools. These command line tools give you access to the same Oracle Security Server features as the Oracle Enterprise Manager tool.

## 1.2 What is the Oracle Cryptographic Toolkit?

The Oracle Cryptographic Toolkit is an interface to the cryptographic services provided by the Oracle Security Server. It is intended to unify all cryptographic services, including the use, storage, retrieval, import, and export of credentials. This interface is used by both internal and external Oracle customers to add security enhancements to their applications. External customers can use either OCI or PL/SQL to access the Oracle Cryptographic Toolkit.

Refer to Figure 1–1, “Relationship between Toolkit and Services”, for an overview of who uses the Oracle Security Server and the Oracle Cryptographic Toolkit and how the two are related.

**Figure 1–1 Relationship between Toolkit and Services**



The Oracle Cryptographic Toolkit presents an abstraction that hides keys and X.509 v1 certificates from the application. The application, then, works with wallets, trusted identities, and personas. A wallet is a storage abstraction that can be located on the file system, in a database, or in a hardware device; a trusted identity is similar to a certificate; and a persona is a combination of a certificate and its associated private key.

## 1.3 Oracle Cryptographic Toolkit Functional Layers

The Oracle Cryptographic Toolkit is comprised of four functional layers: an API layer, a Cryptographic Engine Functions layer, a Persona/Identity Functions layer, and a Wallet Functions layer. Refer to Figure 1-1, “Relationship between Toolkit and Services”.

### 1.3.1 API Layer

The API layer contains three interfaces, or points of entry, into the Oracle Cryptographic Toolkit. The three points of entry are OCI, PL/SQL, and raw C (for Oracle internal customers only). The OCI and PL/SQL interfaces are actually wrappers around the raw C interface.

### 1.3.2 Cryptographic Engine Functions

The Cryptographic Services layer consists of all the cryptographic services available to the Oracle Security Server. These services include the use, storage, retrieval, import and export of credentials. This layer consists of two main components: a cryptographic engine and an abstract cryptographic engine.

Cryptographic engine functions are built on top of a set of primitives presented by the abstract cryptographic engine. The cryptographic engine issues a function call to the abstract cryptographic engine. After it issues the function call, the cryptographic engine verifies that the correct amount of memory is available for any output from the abstract cryptographic engine and that the cipher keys are available in the appropriate format. A cryptographic engine function provides a single interface to the application. Following is a list of cryptographic engine functions.

#### **Attached sign/verify**

The signature generated from a message is attached to that message. The Oracle Cryptographic Toolkit:

- supports both RSA and DSS signatures
- defines and supports an Oracle proprietary signature format

- will support industry standard signature formats such as PKCS #7 and W3C DSig blocks

#### **Detached sign/verify**

The signature generated from a message is kept separate from that message. The Oracle Cryptographic Toolkit:

- supports both RSA and DSS signatures
- defines and supports an Oracle proprietary signature format
- will support industry standard signature formats such as PKCS #7 and W3C DSig blocks

#### **Hash**

The cryptographic checksum of an entity. Both MD5 and SHA hash algorithms are supported.

#### **Keyed hash**

The cryptographic checksum of a message with an additional key folded in. Both MD5 and SHA hash algorithms are supported.

#### **Random Numbers**

Pseudo random number generation. The Oracle Cryptographic Toolkit generates random integers, random sequences of bytes, and allows the application to change the seed value.

### **1.3.3 Persona/Identity Functions**

The Wallet provides storage and retrieval of personas and identities for use with various cryptographic engine functions. In order for an application to call the cryptographic engine functions, the wallet must contain at least one persona. The Oracle Cryptographic Toolkit relies on the persona to carry specific information about what cryptographic algorithm to use with a cryptographic engine function. The application configures the persona for a particular purpose and then uses one or more cryptographic engine functions. The application can therefore treat a persona as a set of security contexts: one for each cryptographic engine function.

### **1.3.4 Wallet Functions**

The Wallet Functions layer implements one or more repositories referred to as wallets. A wallet implements a single way to store, retrieve, and use credentials that



can be located on a file system, a database, or a hardware device. Applications access one or more of these wallets to select personas and identities.

The wallet provides location transparency in two ways. First, the wallet can be located on a file system, in a database, or in a hardware device. Second, each credential stored in a wallet can exist as a typed reference rather than as the actual credential.

The Oracle Cryptographic Toolkit wallet interface becomes a wrapper around the wallet style interface presented by hardware devices. File-based wallets can be treated like a wallet when the format of their credentials are well known. For example, Oracle proprietary, Netscape, and Spyglass file based wallets can be treated as wallets.

In this release, only the default wallet is supported; it is located on a file system. The wallet's location is defined with the `oss.source_my_wallet` `SQLNET.ORA` parameter .

---

---

**Note:** The wallet must be created using the `osslogin` command line tool. Refer to Chapter 3, "Installing and Configuring the Oracle Security Server", in the *Oracle Security Server<sup>TM</sup> Guide*.

---

---

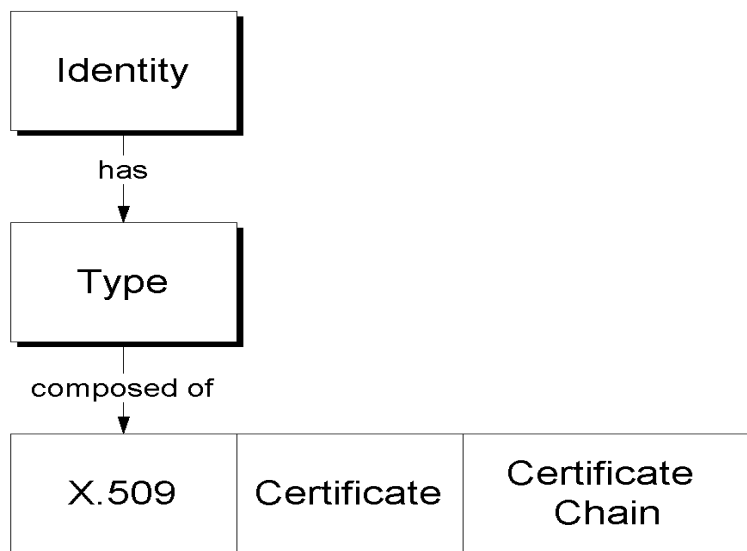
## 1.4 Oracle Cryptographic Toolkit Elements

The Oracle Cryptographic Toolkit works with the following basic elements:

- “Identity”
- “Trusted Identity”
- “Persona”
- “Wallet”

### 1.4.1 Identity

An identity is the public information for an entity. The identity of an object consists of the binding of a public key and other public information for that entity. Every identity has a type; for example, X.509 v1. Refer to Figure 1-2, “Identity”, for an illustration of the structure of an identity.

**Figure 1–2 Identity**

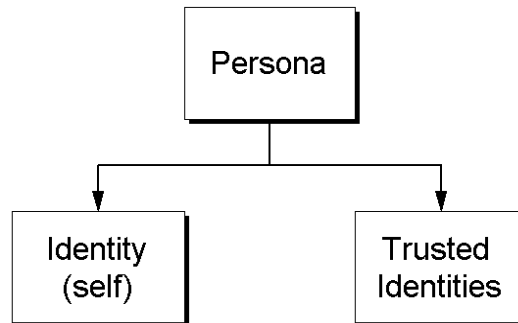
### 1.4.2 Trusted Identity

A trusted identity (or trust point) is an identity that is considered trustworthy. This trusted identity is then used to validate other identities. For example, an X.509 type trusted identity is a Certificate Authority.

### 1.4.3 Persona

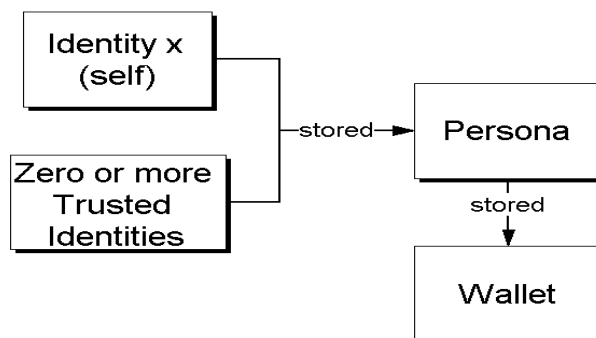
A persona contains an identity, the private information for an entity, a list of actions that can be performed (for example, DSS, RSA, or symmetric key encryption), a set of message formats, and a set of trusted identities. Each persona has a type that it inherits from its identity: for example, X.509 v1.

Refer to Figure 1–3, “Persona”, for an illustration of a persona.

**Figure 1–3 Persona**

#### 1.4.4 Wallet

The Oracle Cryptographic Toolkit also works with one or more repositories called wallets. Wallets are containers that store trusted identities and personas. Refer to Figure 1–4, “Wallet”, for an overview of the relationship between these elements.

**Figure 1–4 Wallet**

## 1.5 Types of Interfaces

The Oracle Cryptographic Toolkit is accessed using two types of interfaces: the Oracle Call Interface and the PL/SQL Interface.

### 1.5.1 Oracle Call Interface

Oracle client programs use the Oracle call interface to access Oracle Security Server functions. Refer to Chapter 6, “OCI Functions for C”, for detailed Oracle call interface programming information.

### 1.5.2 PL/SQL Interface

Oracle server programs use the Oracle PL/SQL interface to access Oracle Security Server functions. Refer to Chapter 7, “PL/SQL Functions”, for detailed PL/SQL interface programming information.

---

# Data Types

This chapter discusses Oracle Cryptographic Toolkit external datatype codes. The following topics are covered:

- “Data Types”
- “Data Structures”

## 2.1 Data Types

Each data type name and its corresponding data type prefix used in the Oracle Cryptographic Toolkit is listed as a subheading below. The table below each subheading lists the possible data type values and their corresponding descriptions.

### 2.1.1 Name Prefixes

Each data type used in the Oracle Cryptographic Toolkit has a unique prefix. Following is a list of Oracle Cryptographic Toolkit data type names and prefixes.

**Table 2–1 Data Types**

Data Type Name	Prefix Used
Crypto Engine State	nzttc <sub>es</sub> _
Crypto Engine Functions	nzttc <sub>ef</sub> _
Identity Type	nzttid <sub>enttype</sub> _
Cipher Types	nzttcip <sub>hertype</sub> _
TDU Formats	nzttd <sub>ufmt</sub> _
Validate State	nzt <sub>tvalstate</sub> _
Unique ID	nzt <sub>tid</sub> _
Timestamp	nzt <sub>tstamp</sub> _

### 2.1.2 Crypto Engine State

`nzttces` Enumerated type listing the current state of the cryptographic engine (CE).

States are:

NZTTCES_CONTINUE	Continue processing input
NZTTCES_END	End processing input
NZTTCES_RESET	Reset processing and skip generating output

### 2.1.3 Crypto Engine Functions

`nztacef` Enumerated type to show the cryptographic engine categories.

Types are:

<code>NZTTCEF_DETACHEDSIGNATURE</code>	Signature, detached from content
<code>NZTTCEF_SIGNATURE</code>	Signature, combined with content
<code>NZTTCEF_KEYEDHASH</code>	Keyed hash/checksum
<code>NZTTCEF_HASH</code>	Hash/checksum
<code>NZTTCEF_RANDOM</code>	Random byte generation
<code>NZTTCEF_LAST</code>	Used for array size

### 2.1.4 Identity Type

`nztIdentType` Enumerated type to indicate the type of identity.

Types are:

<code>NZTTIDENTTYPE_X509v1</code>	X.509v1
<code>NZTTIDENTTYPE_X509v3</code>	X509v3
<code>NZTTIDENTTYPE_SYMMETRIC</code>	Symmetric

### 2.1.5 Cipher Types

`nztCipherType` Enumerated type listing all possible cryptographic algorithms.

Types are:

<code>NZTTCIPHERTYPE_MD5</code>	MD5
<code>NZTTCIPHERTYPE_SHA</code>	SHA

### 2.1.6 TDU Formats

`nzttdufmt` Enumerated type listing all possible toolkit data unit (TDU) formats. Depending on the function and cipher used, some may not be available.

Types are:

<code>NZTTDUFMT_PKCS7</code>	PKCS7 format
<code>NZTTDUFMT_RSAPAD</code>	RSA padded format
<code>NZTTDUFMT_ORACLEv1</code>	Oracle v1 format

### 2.1.7 Validate State

`nztvalstate` Enumerated type listing states an identity can be in.

States are:

<code>NZTTVALSTATE_NONE</code>	Needs to be validated
<code>NZTTVALSTATE_GOOD</code>	Validated
<code>NZTTVALSTATE_REVOKED</code>	Failed to validate

### 2.1.8 Unique ID

`nzttid`

<code>nztID</code>	Unique IDs for personas and identities represented with 128 bits
--------------------	--

### 2.1.9 Timestamp

`nztstamp`

<code>nztTStamp</code>	Timestamp as a 32 bit quantity in UTC
------------------------	---------------------------------------



## 2.2 Data Structures

Following is a list of Oracle Cryptographic Toolkit data structures. Each data structure is listed along with a brief description.

**Table 2–2 Data Structures and Descriptions**

Name of Data Structure	Description
<code>nzttBufferBlock</code>	This is an output parameter block used to describe each buffer
<code>nzttWallet</code>	The Wallet structure contains a list of personas stored in that wallet and private wallet information
<code>nzttPersona</code>	The Persona structure contains information about a persona
<code>nzttIdentity</code>	The Identity structure contains information about an identity

### 2.2.1 `nzttBufferBlock`

A function uses an output parameter block to describe each buffer when that function needs to fill (and possibly grow) an output buffer. The `flags_nzttBufferBlock` member tells the function whether the buffer can be grown. The buffer is automatically reallocated when `flags_nzttBufferBlock` is 0.

The `buflen_nzttBufferBlock` member is set to the length of the buffer before the function is called and equals the length of the buffer when the function is finished. If `buflen_nzttBufferBlock` is 0, then the initial pointer stored in `buflen_nzttBufferBlock` is ignored.

The `usedlen_nzttBufferBlock` member is set to the length of the object stored in the buffer when the function is finished. If the initial buffer had a non zero length, then it is possible that the object length is shorter than the buffer length.

The `buffer_nzttBufferBlock` member is a pointer to the output object. Refer to Table 2–3, “`nzttBufferBlock`”.

**Table 2–3 `nzttBufferBlock`**

Type	Name	Description
<code>uword</code>	<code>flags_nzttBufferBlock</code>	Flags
<code>size_t</code>	<code>buflen_nzttBufferBlock</code>	Total length of buffer
<code>size_t</code>	<code>usedlen_nzttBufferBlock</code>	Length of buffer actually used
<code>ub1</code>	<code>*buffer_nzttBufferBlock</code>	Pointer to buffer

## 2.2.2 nzttWallet

The wallet structure contains one or more personas. Each of these personas contains its private key, its identity, and trusted third party identities. All identities are qualified with trust where the qualifier can indicate anything from untrusted to trusted for specific operations. Refer to Table 2–4, “nzttWallet”.

**Table 2–4** *nzttWallet*

Type	Name	Description
size_t	npersona_nzttWallet	Number of personas in the wallet
nzttPersona	list_nzttWallet	List of personas in the wallet
nzttWalletPrivate	private_nzttWallet	Private wallet information

## 2.2.3 nzttPersona

The persona structure contains information about a persona. Refer to Table 2–5, “nzttPersona”.

**Table 2–5** *nzttPersona*

Type	Name	Description
nzttIdentity	myidentity_nzttPersona	My identity
size_t	nidents_nzttPersona	Number of trusted identities
nzttIdentity	list_nzttPersona	List of trusted identities
nzttPersonaPrivate	private_nzttPersona	Opaque part of persona

## 2.2.4 nzttIdentity

The identity structure contains information about an identity. Refer to Table 2–6, “nzttIdentity”.

**Table 2–6** *nzttIdentity*

Type	Name	Description
size_t	aliaslen_nzttIdentity	Length of alias
text	alias_nzttIdentity	Alias
size_t	commentlen_nzttIdentity	Length of comment
text	comment_nzttIdentity	Comment
nzttIdentityPrivate	private_nzttIdentity	Opaque part of identity

This chapter discusses concepts behind the Oracle Cryptographic Toolkit. The following topics are discussed:

- “Security Concepts”
- “Oracle Cryptographic Toolkit Concepts”

## 3.1 Security Concepts

Following is a list of security concepts used in this document. Refer to Section 1.1.1, “Oracle Security Server Features”, for an explanation of how these concepts apply to the Oracle Cryptographic Toolkit.

### **Authentication**

The recipient of an authenticated message can be certain of the message’s origin (its sender). Authentication reduces the possibility that another person has impersonated the sender of the message.

### **Authorization**

The set of privileges available to an authenticated entity.

### **Certificate**

An entity’s public key signed by a trusted identity (certificate authority) in the form of a certificate. This certificate gives assurance that the entity’s information is correct and that the public key actually belongs to the entity.

### **Certificate Authority**

An application that creates identities by signing public key certificates and stores them in a database or a repository. The certificate authority signature certifies that the information in the certificate is correct and the public key actually belongs to the entity.

### **Confidentiality**

A function of cryptography. Confidentiality guarantees that only the intended recipient(s) of a message can view the message (decrypt the ciphertext).

### **Cryptography**

The act of writing and deciphering in a secret code resulting in secure messages.

### **Decryption**

The process of converting the contents of an encrypted message (ciphertext) back into its original readable format (plaintext).

### **Digital Signature**

A public key algorithm is used to sign the sender’s message with the sender’s private key. The digital signature means that the document is authentic, has not been

forged by another entity, has not been altered, and cannot be repudiated by the sender.

**Encryption**

The process of disguising the contents of a message and rendering it unreadable (ciphertext) to anyone but the intended recipient.

**Integrity**

The guarantee that the contents of the message received were not altered from the contents of the original message sent.

**Non-repudiation**

Undeniable proof of the origin, delivery, submission, or transmission of a message.

**Public-Key Encryption**

The process by which the sender of a message encrypts the message with the public key of the recipient. Upon delivery, the message is decrypted with the recipient's private key.

**Public/Private Key Pair**

Each private key has an associated public key that anyone can access. Data encrypted with a public key can be decrypted with its associated private key and vice versa. However, data encrypted with a public key cannot be decrypted with a public key.

**X.509**

The ISO authentication framework uses public key cryptography (X.509 protocols). X.509 has a structure for public key certificates. This framework allows for authentication across networks to occur.

## 3.2 Oracle Cryptographic Toolkit Concepts

Following is a list of Oracle Cryptographic Toolkit concepts. Refer to Section 1.3, “Oracle Cryptographic Toolkit Functional Layers” for information on how these concepts are implemented.

### **Cryptographic Engine**

A cryptographic engine (CE) is an implementation of cryptographic functions. The CE can be software based, such as RSA’s BSAFE, or it can be hardware based, such as a FORTEZZA card.

### **Detached Signature**

A detached signature gives you the ability to manipulate the message independently of the signature for that message. Use a detached signature to sign an object that can be used with or without signature verification (for example, applets and database rows).

### **Entity**

An entity is a person (physical or imaginary) or a process.

### **Enveloping**

Enveloping is the process of digitally signing a message for authentication and encrypting the message with the recipient’s public key for privacy. It provides both sender verification and message privacy.

### **Identity**

An identity is composed of the public key and any other public information for an entity. The public information may include user identification data: an e-mail address, for example.

### **Persona**

A persona is the combination of an identity (public information) and its associated private information. A persona’s type is inherited from that persona’s identity. A persona is always protected by a password associated with the wallet.

### **Personal Resource Locator**

The personal resource locator (PRL) acts as a reference to a group composed of a persona, its self-identity, and its trusted identities. It is a string in the format:

`type:parameters`

where **type** is one of the defined persona types and **parameters** is 0 or more parameters necessary to access the persona. The platform specific PRL can be specified with:

`default:`

to indicate that the persona is contained inside the wallet and can provide an additional protection key that is specific for this persona.

---

---

**Note:** The value of the platform specific PRL above is `default`, because only the default wallet is supported in this release of the Oracle Cryptographic Toolkit.

---

---

### Protection Set

A protection set is a list of tuples (elements) in the form ((cryptographic-function-1, format, algorithm(s), parameter(s)) (cryptographic-function-2, format, algorithm(s), parameter(s)), ...). It represents the current set of algorithms and message formats to be used with the cryptographic functions.

### Recipient Oriented Encryption

Recipient Oriented Encryption is the process of encrypting a message with a randomly generated symmetric key and then encrypting the encrypted message with the public key of the recipient.

### Signature

See “Digital Signature”.

### Symmetric Encryption

Symmetric Encryption is an encryption method where both of the communicating parties agree on a secret key (or algorithm) that can be used to both encrypt and decrypt a message.

### Toolkit Data Unit

A toolkit data unit (TDU) is an encoding of possibly formatted and/or cryptographically altered data that is created by an application using the Oracle Cryptographic Toolkit. The TDU is usually transferred to another application that, in turn, uses the Oracle Cryptographic Toolkit to decrypt the TDU back into data. The TDU is the

message granularity of the Oracle Cryptographic Toolkit, and it is transport independent.

### **Trust Point**

A trust point is a third party identity contained within a persona that is qualified with a level of trust. The trust point is used when an identity is being validated as the entity it claims to be.

### **Wallet**

A wallet implements the storage and retrieval of credentials for use with various cryptographic services. It represents a storage facility that is location and type transparent once it is opened. A Wallet Resource Locator provides all the necessary information to locate the wallet.

A Wallet Resource Locator (WRL) is a string in the format:

`type:parameters`

where **type** is one of the defined wallet types and **parameters** is 0, or more, parameters necessary to access the wallet. The platform specific WRL can be specified with:

`default:`

to quickly access the default wallet.

---

---

**Note:** The value of the platform specific WRL above is `default`, because only the default wallet is supported in this release of the Oracle Cryptographic Toolkit.

---

---



---

# Using the Oracle Cryptographic Toolkit

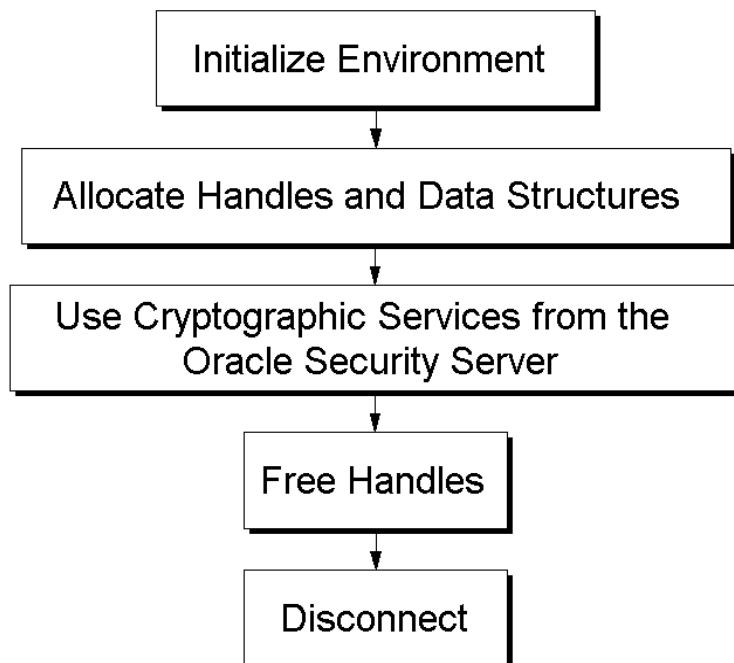
This chapter shows you how to program using the Oracle Cryptographic Toolkit. The following topics are discussed:

- “Basic Oracle Cryptographic Toolkit Program Flow”
- “A Programming Example”

## 4.1 Basic Oracle Cryptographic Toolkit Program Flow

The following section describes the typical program flow for those who want to use the Oracle Cryptographic Toolkit and provides program code examples for calling the available functions. Refer to Figure 4–1, “Oracle Cryptographic Toolkit Program Flow”, below, for an illustration of how a typical program flows using the Oracle Cryptographic Toolkit.

**Figure 4–1** *Oracle Cryptographic Toolkit Program Flow*



## 4.2 A Programming Example

This section first lists the programming steps to follow when you use the Oracle Cryptographic Toolkit. The balance of this chapter provides the following sample code for your use:

“An Example: Generating a detached signature for an array of bytes”

## 4.2.1 Using the Oracle Cryptographic Toolkit

Follow steps 1 - 5 to access the Oracle Security Server.

1. Once the OCI process has been initialized with `OCIInitialize` and the environment has been initialized with `OCIEnvInit` (refer to the *Programmer's Guide to the Oracle Call Interface*), the security handle can be created with `OCIHandleAlloc` and initialized with `OCISecurityInitialize`. The security handle is used with subsequent calls to the Oracle Cryptographic Toolkit.

```
...
OCIError      *error_handle = (OCIError *) NULL;
OCISecurity *security_handle = (OCISecurity *) NULL;
...

/*
 * The OCI process and environment have already been initialized.
 */

OCIHandleAlloc((dvoid *) env_handle, (dvoid **) &error_handle,
               (ub4) OCI_HTYPE_ERROR,
               (size_t) 0, (dvoid **) 0),

OCIHandleAlloc((dvoid *) env_handle,
               (dvoid **) &security_handle,
               (ub4) OCI_HTYPE_SECURITY, (size_t) 0,
               (dvoid **) 0);

OCISecurityInitialize(security_handle, error_handle);
```

2. Typically, an application will first need to open a wallet in order to get its persona and gain access to the list of trusted identities. The wallet location is specified through a Wallet Resource Locator (WRL), and if the contents have been protected with a password, the correct password must be provided as well.

```
...
nzttWallet wallet;
...

OCISecurityOpenWallet(security_handle, error_handle,
                     wrllen, wrl,
                     passlen, password,
                     &wallet)
```

3. Next, an application will choose a persona from the wallet and open it to prepare it for use.

```
...
nzttPersona *persona;
...

/*
 * Use the first persona in the wallet.
 */
persona = &wallet.list_nzttWallet[0];

OCISecurityOpenPersona(security_handle, error_handle, persona);
```

4. The application can now perform a cryptographic function such as signing some data:

```
...
nzttBufferBlock signature;
...

memset(&signature, 0, sizeof(signature));
OCISecuritySign(security_handle, error_handle, persona,
                NZTTES_END, strlen((char *)"Some data"),
                "Some data", &signature);
```

5. During termination, the application should call OCISecurityFree to deallocate the security handle once the wallet has been closed and the security subsystem has been terminated.

```
OCISecurityCloseWallet(security_handle, error_handle, &wallet);
OCISecurityTerminate(security_handle, error_handle);
OCISecurityFree((void *) security_handle, OCI_HTYPE_SECURITY);
```

## 4.2.2 An Example: Generating a detached signature for an array of bytes

The following code sample shows you how to generate a detached signature for an array of bytes. For brevity, errors are checked but are not displayed. Refer to Part III, “Appendices”, for a complete code example.

```
#include <oratypes.h>

#ifdef OCI_ORACLE
#include <oci.h>
#endif

#ifdef OCIDFN
#include <ocidfn.h>
#endif

#ifdef __STDC__
#include <ociap.h>
#else
#include <ocikp.h>
#endif

static text phrase[] = "This is a static text phrase";

int main(argc, argv)
int argc;
char *argv[];
{
    nzttWallet wallet;                /* Wallet structure */
    nzttBufferBlock signature;        /* Detached signature */
    nzttPersona *persona = (nzttPersona *)NULL; /* Persona used to sign */
    OCIEnv *env_handle = (OCIEnv *)NULL; /* OCI environment handle */
    OCIErr *error_handle = (OCIErr *)NULL; /* OCI error handle */
    OCISecurity *security_handle = (OCISecurity *)NULL; /* OCI security handle */

    /*
     * Clear out the wallet and signature structures so that if an
     * error occurs before they are used, they are not mistaken for
     * holding allocated memory.
     */
    memset(&wallet, 0, sizeof(wallet));
    memset(&signature, 0, sizeof(signature));
    /*
     * Initialize the OCI process.
     */
}
```

```
if (OCI_SUCCESS
    != OCIInitialize((ub4) OCI_DEFAULT, (dvoid *)0, (dvoid *(*())0),
                    (dvoid *(*())0, (void(*)())0))
{
    goto exit;
}

/*
 * Initialize the OCI environment.
 */
if (OCI_SUCCESS
    != OCIEnvInit((OCIEnv **)&env_handle, (ub4)OCI_DEFAULT, (size_t)0,
                  (dvoid **)&0))
{
    goto exit;
}

/*
 * Create an error handle.
 */
if (OCI_SUCCESS
    != OCIHandleAlloc((dvoid *)env_handle, (dvoid **)&error_handle,
                      (ub4)OCI_HTYPE_ERROR, (size_t)0, (dvoid **)&0))
{
    goto exit;
}

/*
 * Create a security handle
 */
if (OCI_SUCCESS
    != OCIHandleAlloc((dvoid *)env_handle, (dvoid **)&security_handle,
                      (ub4)OCI_HTYPE_SECURITY, (size_t)0, (dvoid **)&0))
{
    goto exit;
}

/*
 * Initialize the security subsystem.
 */
if (OCI_SUCCESS != OCISecurityInitialize(security_handle, error_handle))
{
    goto exit;
}
```

```

/*
 * Open the wallet. Since NZT_DEFAULT_WRL is used as the wallet
 * WRL, the platform specific default wallet will be used. Note,
 * as well, that this wallet has no password (NZT_NO_PASSWORD).
 */
if (OCI_SUCCESS
    != OCISecurityOpenWallet(security_handle, error_handle,
                             strlen(NZT_DEFAULT_WRL), NZT_DEFAULT_WRL,
                             strlen(NZT_NO_PASSWORD), NZT_NO_PASSWORD,
                             &wallet))
{
    goto exit;
}

/*
 * Use the first persona in the wallet.
 */
persona = &wallet->list_nzttWallet[0];

/*
 * Open the persona and prepare it for use.
 */
if (OCI_SUCCESS
    != OCISecurityOpenPersona(security_handle, error_handle, persona))
{
    goto exit;
}

/*
 * Create a detached signature for the phrase. This means that
 * when the signature is verified, the original phrase will need to
 * be provided since it is not attached to the signature. The
 * variable signature contains the output.
 */
if (OCI_SUCCESS
    != OCISecuritySignDetached(security_handle, error_handle, persona,
                                NZTTCES_END, strlen((char *)phrase),
                                phrase, &signature))
{
    goto exit;
}

exit:
DISCARD OCISecurityPurgeBlock(security_handle, error_handle, &signature);

```

```
DISCARD OCISecurityCloseWallet(security_handle, error_handle, &wallet);

/*
 * Free the various handles (if allocated). Delay freeing the error
 * handle so that errors can be generated until the last possible
 * moment.
 */
if (security_handle)
{
    DISCARD OCISecurityTerminate(security_handle, error_handle);
    DISCARD OCIHandleFree((dvoid *)security_handle, OCI_HTYPE_SECURITY);
}

if (error_handle)
{
    DISCARD OCIHandleFree((dvoid *)error_handle, OCI_HTYPE_ERROR);
}

if (env_handle)
{
    DISCARD OCIHandleFree((dvoid *)env_handle, OCI_HTYPE_ENV);
}

return 0;
}
```



---

# Random Number Generator

This chapter discusses the Oracle Cryptographic Toolkit random number generator. The following topics are covered:

- “Overview”
- “Functions”
- “Example”

## 5.1 Overview

The random number generator is built on top of the Oracle Cryptographic Toolkit. This tool is intended for users who want to generate random data for their applications.

## 5.2 Functions

The random number generator is composed of the following:

### **PROCEDURE Initialize (seed IN BINARY\_INTEGER)**

This procedure is used before the random number generator package is called. The procedure takes a seed which initializes the random number generator. The seed can be any value between -999999999 and 999999999.

---

---

**Note:** You must call this procedure before using any of the other procedures or functions. Otherwise, an exception will be raised.

---

---

### **PROCEDURE Seed (seed IN BINARY\_INTEGER)**

This procedure resets the seed used by the random number generator.

### **FUNCTION Random RETURN BINARY\_INTEGER**

The function returns a random number between -999999999 and 999999999.

### **PROCEDURE Terminate**

This procedure must be called when the package is no longer needed.

## 5.3 Example

The following code fragment is an example of how to use the random number generator package.

```
DECLARE
    i BINARY_INTEGER;
BEGIN
    dbms_random.initialize(19254);
    i := dbms_random.random;
    INSERT INTO some_table VALUES(i);
    dbms_random.terminate;
END;
```

---

---

**Note:** It is not currently possible to use the return value of RANDOM directly in a SQL statement. The following is not allowed, for example:

```
INSERT INTO some_table VALUES (DBMS_RANDOM.RANDOM);
```

---

---



# Part II

---

## Reference

Part II, Reference, contains the following chapters:

- “OCI Functions for C”
- “PL/SQL Functions”



---

## OCI Functions for C

This chapter describes each Oracle Call Interface (OCI) function in the Oracle Cryptographic Toolkit. Each OCI function description contains the following information:

Purpose	Describes what the function does
Parameter Descriptions	Lists a detailed description of each parameter name along with its description, mode, and type
Comments	Gives detailed information about the OCI function and includes an example
Errors	Lists some of the possible values returned by the function.

Refer to Chapter 2, OCI Programming Basics, in the *Programmer's Guide to the Oracle Call Interface<sup>TM</sup>* for an overview of the steps involved in calling OCI functions.

Refer to Appendix B, "OCI - API Mappings" for a list of OCI functions and the API functions to which they map.

## 6.1 OCISecurityInitialize

OCISecurityInitialize must be called after the user gets a security handle but before any security function is called.

### Error Handles

Error handles are passed as parameters to OCI calls. Error handles are allocated at the beginning of an OCI application. The following handles are passed:

**Table 6–1** *OCISecurityInitialize Handles*

Handle Type	Handle Name
OCISecurity	osshandle
OCIError	error_handle



## 6.2 OCI`SecurityTerminate`

### Purpose

OCI`SecurityTerminate` must be called after the user has finished using the security routines.

### Parameter Descriptions

Following is a list of parameters and their descriptions.

***Table 6–2 OCI`SecurityTerminate` parameters***

Parameter Name	Description
OCI <code>Security</code>	osshandle
OCI <code>Error</code>	error_handle

## 6.3 OCISecurityOpenWallet

### Purpose

OCISecurityOpenWallet opens a wallet based on the wallet resource locator (WRL).

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–3** *OCISecurityOpenWallet parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
wrllen	Length of wallet resource locator	[IN]	size_t
wallet_resource_locator	Wallet resource locator	[IN]	text
pwdlen	Length of password	[IN]	size_t
password	Password	[IN]	text
wallet	Initialized wallet structure	[IN]	nztWallet

### Comments

Defaults: The platform specific WRL default is used when the WRL is NZT\_DEFAULT\_WRL. Use the WRL type specific default (e.g., “oracle:”) when only the wallet type is specified.

A wallet is opened and its password is verified by hashing it and comparing the result with the password hash stored with the wallet. The list of personas and their associated identities is built and stored into the wallet structure.

Implication: An Oracle based wallet can be implemented either in a user’s private space or in world readable space.

## 6.4 OCISecurityCloseWallet

### Purpose

OCISecurityCloseWallet closes a wallet based on the wallet resource locator (WRL).

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–4** *OCISecurityCloseWallet parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCIErrror	error_handle		
wallet	Initialized wallet structure	[IN]	nztWallet

### Comments

Closing a wallet also closes all personas associated with that wallet. Any changes you have made to the persona will not automatically be saved.

Implication: An application can modify a persona, but the persona will revert to what it was in the wallet if it is not explicitly saved.

## 6.5 OCISecurityOpenPersona

### Purpose

OCISecurityOpenPersona opens a persona in a wallet.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–5** *OCISecurityOpenPersona parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
persona	Persona	{IN/OUT}	nzttnPersona

### Comments

A persona must be selected and opened before a cryptographic engine function can be used. The opened persona then initializes the protection set to either the system defaults or persona specific preferences. The opened persona also contains and maintains any state information necessary for the cryptographic engine functions.

### Returns

Following is a list of possible error codes returned by this function.

**Table 6–6** *OCISecurityOpenPersona errors*

Error	Explanation
NZERROR_TK_PASSWORD	Password failed to decrypt persona
NZERROR_TK_BADPRL	Persona resource locator did not work
NZERROR_RIO_OPEN	Could not open persona (see network trace file)

## 6.6 OCISecurityClosePersona

### Purpose

OCISecurityClosePersona closes a persona in a wallet.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–7** *OCISecurityClosePersona parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
persona	Persona	{IN/OUT}	nzttPersona

### Comments

A persona is not stored when it is closed; it only releases the memory associated with the crypto engine.

### Returns

Following is a list of possible error codes returned by this function.

**Table 6–8** *OCISecurityClosePersona errors*

Error	Explanation
NZERROR_OK	Success
NZERROR_TK_PASSWORD	Password failed to decrypt persona
NZERROR_TK_BADPRL	Persona resource locator did not work
NZERROR_RIO_OPEN	Could not open persona (see network trace file)

## 6.7 OCISecuritySign

**Purpose**

OCISecuritySign creates an attached signature.

**Parameter Descriptions**

Following is a list of parameters, their descriptions, modes, and types.

*Table 6–9 OCISecuritySign parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
persona	Open persona acting as signer	{IN}	nztPersona
signature_state	State of the signature	{IN}	nztTces
input_length	Length of this input part	{IN}	sizt_t
input	This input part	{OUT}	ub1
buffer_block	TDU buffer	{IN/OUT}	nztBufferBlock

**Comments**

This function generates a signature that consists of a cryptographic checksum of the data to be signed: encrypted with the private key of the signing persona. The original data is then attached to the signature.

## 6.8 OCISecurityVerify

### Purpose

OCISecurityVerify verifies an attached signature.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

*Table 6–10 OCISecurityVerify parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
persona	Persona	{IN}	nztPersona
signature_state	State of verification	{IN}	nztTces
siglen	TDU length	{IN}	size_t
signature	Token Data Unit	{IN}	ub1
extracted_message	Extracted message	{IN/OUT}	nztBufferBlock
verified	TRUE if signature is verified	{OUT}	boolean
validated	TRUE if signing identity validated	{OUT}	boolean
signing_party_identity	Identity of signing party	{OUT}	nztIdentity

### Comments

The data from the attached signature is used to generate a cryptographic checksum. Then the signature part of the attached signature is decrypted using the signing identity's public key. The two checksums are then compared to verify they are identical. The signing identity is also validated to verify that it can be trusted and that it has not expired.

## Returns

Following is a list of possible error codes returned by this function.

**Table 6–11** *OCISecurityVerify errors*

Error	Explanation
NZERROR_TK_CANTGROW	Needed to grow output buffer but could not
NZERROR_TK_NOTOPEN	Persona is not open
NZERROR_TK_NOTSUPP	Function not supported with persona



## 6.9 OCI`SecurityValidate`

### Purpose

OCI`SecurityValidate` validates an identity.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–12** OCI`SecurityValidate` parameters

Parameter Name	Description	Mode	Type
OCI <code>Security</code>	osshandle		
OCI <code>Error</code>	error_handle		
persona	Persona	{IN}	nzttnPersona
identity	Identity	{IN}	nzttnIdentity
validated	TRUE if identity was validated	{OUT}	boolean

### Comments

An identity is validated for trust and to verify that it has not expired.

### Returns

Following is a list of possible error codes returned by this function.

**Table 6–13** OCI`SecurityValidate` errors

Error	Explanation
NZ <code>ERROR_TK_NOTOPEN</code>	Persona is not open
NZ <code>ERROR_TK_NOTSUPP</code>	Function not supported with persona

## 6.10 OCISecuritySignDetached

### Purpose

OCISecuritySignDetached generates a detached signature.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–14** *OCISecuritySignDetached parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
persona	Persona	{IN}	nzttnPersona
signature_state	State of signature	{IN}	nzttnces
input_length	Length of this input part	{IN}	size_t
input	This input part	{IN}	ub1
signature	TDU buffer	{IN/OUT}	nzttnBufferBlock

### Comments

The function is identical to OCISecuritySign, but the data to be signed is not attached to the signature. It generates a signature that consists of a cryptographic checksum of the data to be signed, encrypted with the private key of the signing persona.

### Returns

Following is a list of possible error codes returned by this function.

**Table 6–15** *OCISecuritySignDetached errors*

Error	Explanation
NZERROR_TK_NOTSUPP	Function not supported with persona

## 6.11 OCISecurityVerifyDetached

### Purpose

OCISecurityVerifyDetached verifies a detached signature.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

*Table 6–16 OCISecurityVerifyDetached parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCIErrror	error_handle		
persona	Persona	{IN}	nztPerson
signature_state	State of signature	{IN}	nztces
data_length	Length of data	{IN}	size_t
data	Data	{IN}	ub1
siglen	Input TDU length	{IN}	size_t
signature	Input TDU	{IN}	ub1
verified	TRUE if signature is verified	{OUT}	boolean
validated	TRUE if signing identity validated	{OUT}	boolean
signing_party_identity	Identity of signing party	{OUT}	nztIdentity

### Comments

This function is identical to OCISecurityVerify, except the signature does not contain the data that will allow it to be verified. The data is provided by the application calling the function.

## Returns

Following is a list of possible error codes returned by this function.

**Table 6–17** *OCISecurityVerifyDetached errors*

Error	Explanation
NZERROR_TK_NOTOPEN	Persona is not open
NZERROR_TK_NOTSUPP	Function not supported with persona

## 6.12 OCISecurityHash

### Purpose

OCISecurityHash generates a hash.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–18** *OCISecurityHash parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCIErrror	error_handle		
persona	Persona	{IN}	nzttnPersona
hash_state	State of hash	{IN}	nzttnces
input	Length of this input	{IN}	size_t
input_length	This input	{IN}	ub1
hash	Output TDU	{IN/OUT}	nzttnBufferBlock

### Comments

This hash is a cryptographic hash, or checksum, of the input.

### Returns

Following is a list of some of the possible error codes returned by this function.

**Table 6–19** *OCISecurityHash errors*

Error	Explanation
NZERROR_TK_NOTSUPP	Function not supported with persona

## 6.13 OCISecuritySeedRandom

### Purpose

OCISecuritySeedRandom supplies a seed to the Oracle Cryptographic Toolkit.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–20** *OCISecuritySeedRandom parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
persona			nztPersonna
seed_length			size_t
seed			ub1

## 6.14 OCISecurityRandomBytes

### Purpose

OCISecurityRandomBytes generates a buffer block for random bytes.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–21** *OCISecurityRandomBytes parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
persona	Persona	{IN}	nzttnPersona
number_of_bytes_desired	Number of bytes desired	{IN}	size_t
random_bytes	Buffer block for bytes	{IN/OUT}	nzttnBufferBlock

## 6.15 OCISecurityRandomNumber

**Purpose**

OCISecurityRandomNumber generates a random number.

**Parameter Descriptions**

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–22** *OCISecurityRandomNumber parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
persona	Persona	{IN}	nztPersona
random_number_ptr	Number	{OUT}	uword



## 6.16 OCI`SecurityInitBlock`

### Purpose

OCI`SecurityInitBlock` initializes a buffer block.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–23** OCI`SecurityInitBlock` parameters

Parameter Name	Description	Mode	Type
OCI <code>Security</code>	osshandle		
OCI <code>Error</code>	error_handle		
buffer_block	Buffer block	{IN/OUT}	nztbBufferBlock

### Comments

The buffer block is initialized to be empty (all members are set to zero/NULL). This block is allocated to memory as needed.

## 6.17 OCISecurityReuseBlock

**Purpose**

OCISecurityReuseBlock reuses a previously initialized, and possibly used, block.

**Parameter Descriptions**

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–24** *OCISecurityReuseBlock parameters*

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
buffer_block	Buffer block	{IN/OUT}	nztBufferBlock

**Comments**

This function sets the used length member of the buffer block to zero (0). It will cause a block to be reused if it already has memory allocated to it.

## 6.18 OCI SecurityPurgeBlock

### Purpose

OCI SecurityPurgeBlock purges a buffer block of its memory.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

*Table 6–25 OCI SecurityPurgeBlock parameters*

Parameter Name	Description	Mode	Type
OCI Security	osshandle		
OCI Error	error_handle		
buffer_block	Buffer block	{IN/OUT}	nztBufferBlock

### Comments

This command affects only the memory used by the buffer. It does not affect the block itself.

# 6.19 OCISecuritySetBlock

**Purpose**

OCISecuritySetBlock sets a buffer block to a known state.

**Parameter Descriptions**

Following is a list of parameters, their descriptions, modes, and types.

**Table 6–26 OCISecuritySetBlock parameters**

Parameter Name	Description	Mode	Type
OCISecurity	osshandle		
OCLError	error_handle		
flags_to_set	Flags to set	{IN}	uword
buffer_length	Length of buffer	{IN}	size_t
used_buffer_length	Used length of buffer	{IN}	size_t
buffer_block	Buffer	{IN}	ub1

**Comments**

This function allocates memory and stores a pointer in the buffer block.

---

## PL/SQL Functions

This chapter describes the PL/SQL interface to the Oracle Cryptographic Toolkit. The PL/SQL procedures and functions are grouped into the following five functional categories:

Section 7.1, “General Purpose Procedures”

Section 7.2, “Digital Signature”

Section 7.3, “Hash”

Section 7.4, “Random Number Generation”

Each PL/SQL function description contains the following information:

***Table 7–1 PL/SQL Procedure and Function Descriptions***

Purpose	Describes what the procedure or function does
Parameter Descriptions	Lists each parameter name along with its mode and type

# 7.1 General Purpose Procedures

The following functions and procedures are available to applications. They are contained within the DBMS\_CRYPTO\_TOOLKIT package. Consult the file DBMS\_OCTK.SQL for a full listing of functions and procedures.

## Initialize

Initialize starts the Oracle Cryptographic Toolkit operation. No additional parameters are required.

## Terminate

Terminate ends the Oracle Cryptographic Toolkit operation. No additional parameters are required.

## OpenWallet

OpenWallet opens a wallet based on a given wallet resource locator (WRL). There are two versions of this procedure: one enables an application to use its own data structure for the wallet, and the other lets the application use the wallet data structure that comes with the Oracle Cryptographic Toolkit.

**Table 7–2    PROCEDURE OpenWallet**

Parameter Name	Mode	Type
password	IN	VARCHAR2
wallet	IN OUT	Wallet
persona_list	OUT	Persona_List
wallet_resource_locator	IN	VARCHAR2

### 7.1.1 Procedures Used by Applications That Use the Wallet

The following functions and procedures are used by applications which want to use the wallet kept by the Oracle Cryptographic Toolkit.

#### OpenWallet

OpenWallet opens a wallet based on a given wallet resource locator (optional). There are two versions of this procedure. This version opens the wallet that is kept internally by the package.

**Table 7–3 PROCEDURE OpenWallet**

Parameter Name	Mode	Type
password	IN	VARCHAR2
persona_list	OUT	Persona_List
wallet_resource_locator	IN	VARCHAR2

#### CloseWallet

CloseWallet closes a wallet. This version uses the wallet that is kept internally by the package. No parameters are needed for the function.

**Table 7–4 PROCEDURE CloseWallet**

Parameter Name	Mode	Type

#### DestroyWallet

DestroyWallet deletes a wallet bases on a given wallet resource locator. The wallet resource locator is optional.

**Table 7–5 PROCEDURE DestroyWallet**

Parameter Name	Mode	Type
password	IN	VARCHAR2
wallet_resource_locator	IN	VARCHAR2

**StorePersona**

StorePersona stores a given persona in the specified wallet.

**Table 7–6** *PROCEDURE StorePersona*

Parameter Name	Mode	Type
persona	IN	Persona

**OpenPersona**

OpenPersona opens a persona within a wallet.

**Table 7–7** *PROCEDURE OpenPersona*

Parameter Name	Mode	Type
persona	IN	Persona

**ClosePersona**

ClosePersona closes a persona within a wallet.

**Table 7–8** *PROCEDURE ClosePersona*

Parameter Name	Mode	Type
persona	IN	Persona

**RemovePersona**

RemovePersona removes a persona from a wallet.

**Table 7–9** *PROCEDURE RemovePersona*

Parameter Name	Mode	Type
persona	IN	Persona

**CreatePersona**

CreatePersona creates a persona.

**Table 7–10** *PROCEDURE CreatePersona*

Parameter Name	Mode	Type
cipher_type	IN	Cipher
private_information	IN OUT	Private_Persona_Information



**Table 7–10 PROCEDURE CreatePersona**

Parameter Name	Mode	Type
prl	IN OUT	VARCHAR2
alias	IN	VARCHAR2
longer_description	IN	VARCHAR2
persona	OUT	Persona

**RemoveIdentity**

RemoveIdentity destroys an identity.

**Table 7–11 PROCEDURE RemoveIdentity**

Parameter Name	Mode	Type
identity	OUT	Identity

**CreateIdentity**

CreateIdentity creates an identity.

**Table 7–12 CreateIdentity**

Parameter Name	Mode	Type
identitytype	IN	Identity_Type
public_identity	IN	VARCHAR2
alias	IN	VARCHAR2
longer_description	IN	VARCHAR2
trust_qualifier	IN	VARCHAR2
identity	OUT	Identity

**AbortIdentity**

AbortIdentity aborts an identity.

**Table 7–13 AbortIdentity**

Parameter Name	Mode	Type
identity	IN OUT	Identity
persona	IN	

**StoreTrustedIdentity**

StoreTrustedIdentity stores an identity as a trustpoint within a wallet.

**Table 7–14** *StoreTrustedIdentity*

Parameter Name	Mode	Type
identity	IN OUT	Identity

**Validate**

Validate uses the trusted identities associated with a persona to validate an identity.

**Table 7–15** *Validate*

Parameter Name	Mode	Type
persona	IN	Persona
identity	IN	Identity
validated	OUT	BOOLEAN

## 7.2 Digital Signature

Use the following routines to create and verify digital signatures. There are two versions of each routine: one for raw data and another for strings. The routines are as follows:

Section 7.2.1, “Sign”

Section 7.2.2, “Verify”

Section 7.2.3, “SignDetached”

Section 7.2.4, “VerifyDetached”

## 7.2.1 Sign

### Purpose

The Sign routine creates an attached signature.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 7–16   Sign parameters for raw data**

Parameter Name	Mode	Type
persona	IN	Persona
input	IN	RAW
signature	OUT	RAW
signature_state	IN	Crypto_Engine_State

**Table 7–17   Sign parameters for string data**

Parameter Name	Mode	Type
persona	IN	Persona
input_string	IN	VARCHAR2
signature	OUT	RAW
signature_state	IN	Crypto_Engine_State

## 7.2.2 Verify

### Purpose

The Verify routine verifies an attached signature.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 7–18 Verify parameters for raw data**

Parameter Name	Mode	Type
persona	IN	Persona
signature	IN	RAW
extracted_message	OUT	RAW
verified	OUT	BOOLEAN
validated	OUT	BOOLEAN
signing_party_identity	OUT	Identity
signature_state	IN	Crypto_Engine_State

**Table 7–19 Verify parameters for string data**

Parameter Name	Mode	Type
persona	IN	Persona
signature	IN	RAW
extracted_message_string	OUT	VARCHAR2
verified	OUT	BOOLEAN
validated	OUT	BOOLEAN
signing_party_identity	OUT	Identity
signature_state	IN	Crypto_Engine_State

### 7.2.3 SignDetached

**Purpose**

The SignDetached routine generates a detached signature.

**Parameter Descriptions**

Following is a list of parameters, their descriptions, modes, and types.

**Table 7–20   SignDetached parameters for raw data**

Parameter Name	Mode	Type
persona	IN	Persona
input	IN	RAW
signature	OUT	RAW
signature_state	IN	Crypto_Engine_State

**Table 7–21   SignDetached parameters for string data**

Parameter Name	Mode	Type
persona	IN	Persona
input_string	IN	VARCHAR2
signature	OUT	RAW
signature_state	IN	Crypto_Engine_State

## 7.2.4 VerifyDetached

### Purpose

The VerifyDetached routine verifies a detached signature.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 7–22** *VerifyDetached parameters for raw data*

Parameter Name	Mode	Type
persona	IN	Persona
data	IN	RAW
signature	IN	RAW
verified	OUT	BOOLEAN
validated	OUT	BOOLEAN
signing_party_identity	OUT	Identity
signature_state	IN	Crypto_Engine_State

**Table 7–23** *VerifyDetached parameters for string data*

Parameter Name	Mode	Type
persona	IN	Persona
data_string	IN	VARCHAR2
signature	IN	RAW
verified	OUT	BOOLEAN
validated	OUT	BOOLEAN
signing_party_identity	OUT	Identity
signature_state	IN	Crypto_Engine_State

## 7.3 Hash

Use the following routines to generate checksums. There are two versions of each routine: one for raw data and another for strings. The routines are as follows:

Section 7.3.1, “KeyedHash”

Section 7.3.2, “Hash”



## 7.3.1 KeyedHash

### Purpose

The following KeyedHash routine generates a public key checksum.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 7–24** *KeyedHash parameters for raw data*

Parameter Name	Mode	Type
persona	IN	Persona
input	IN	RAW
keyed_hash	OUT	RAW
hash_state	IN	Crypto_Engine_State

**Table 7–25** *KeyedHash parameters for string data*

Parameter Name	Mode	Type
persona	IN	Persona
input_string	IN	VARCHAR2
keyed_hash	OUT	RAW
hash_state	IN	Crypto_Engine_State

## 7.3.2 Hash

### Purpose

The following Hash routine generates a checksum.

### Parameter Descriptions

Following is a list of parameters, their descriptions, modes, and types.

**Table 7–26 Hash parameters for raw data**

Parameter Name	Mode	Type
persona	IN	Persona
input	IN	RAW
hash	OUT	RAW
hash_state	IN	Crypto_Engine_State

**Table 7–27 Hash parameters for string data**

Parameter Name	Mode	Type
persona	IN	Persona
input_string	IN	VARCHAR2
hash	OUT	RAW
hash_state	IN	Crypto_Engine_State

## 7.4 Random Number Generation

Use the DBMS\_RANDOM package to generate random numbers. The routines contained within the package are as follows.

### SeedRandom

The following SeedRandom routine supplies a seed to the Oracle Cryptographic Toolkit's random number generator.

### Parameter Descriptions

Following is a list of parameter names, their modes, and types.

**Table 7–28** *SeedRandom parameters for numeric data*

Parameter Name	Mode	Type
seed	IN	BINARY_INTEGER

### Random

The Random routine generates a random number between -999999999 and 999999999. This function returns a BINARY\_INTEGER.



# Part III

---

## Appendices

Part III, Appendices, contains the following reference information:

- “Sample PL/SQL Code”
- “OCI - API Mappings”



# A

---

## Sample PL/SQL Code

This appendix contains a sample PL/SQL program written in C.

- “Sample PL/SQL Program”

## A.1 Sample PL/SQL Program

Following is a sample PL/SQL program for your reference. Segments of this code are numbered and contain narrative text explaining portions of the code.

```
declare
    wallet dbms_crypto_toolkit.Wallet;
    persona_list dbms_crypto_toolkit.Persona_List;
    persona dbms_crypto_toolkit.Persona;
    string_input VARCHAR2(6) := '123456';
    signature RAW(2048);
    signing_party dbms_crypto_toolkit.Identity;
    recipient dbms_crypto_toolkit.Identity;

    -- Flags to indicate the package state.
    initialized BOOLEAN := FALSE;
    wallet_opened BOOLEAN := FALSE;
    persona_opened BOOLEAN := FALSE;

    operation_unsupported EXCEPTION;
    PRAGMA EXCEPTION_INIT (operation_unsupported, -28841);
    ENCRYPTION_UNSUPPORTED_MESSAGE VARCHAR2(64) :=
        '**** ENCRYPTION UNSUPPORTED - IGNORING EXCEPTION ****';
    encrypted_string VARCHAR2 (2048);
    decrypted_string VARCHAR2 (2048);
    extracted_string VARCHAR2 (128);
    hash_string      VARCHAR2 (2048);
    string_verified  BOOLEAN := FALSE;
    string_validated BOOLEAN := FALSE;
    all_done         BOOLEAN := FALSE;
    done_exception   EXCEPTION;

BEGIN
1.  Start Oracle Cryptographic Toolkit operation.

    dbms_output.put_line('> Initialize');
    dbms_crypto_toolkit.Initialize;
    initialized := TRUE;
```



**2. Open a wallet at the default location.**

```

dbms_output.put_line('> OpenWallet');
dbms_crypto_toolkit.OpenWallet('server1', wallet, persona_list, 'default:');
wallet_opened := TRUE;

```

**3. Establish the identity associated with the first persona in the new wallet as the recipient.**

```

dbms_output.put_line('> Alias ' || persona_list(1).alias);
dbms_output.put_line('> Comment ' || persona_list(1).comment);
persona.persona := persona_list(1).persona;
recipient.Descriptor := persona_list(1).identity;

```

**4. Open the first persona.**

```

dbms_output.put_line('> OpenPersona');
dbms_crypto_toolkit.OpenPersona(persona);
persona_opened := TRUE;

```

**5. Create an attached signature associated with the current persona.**

```

dbms_output.put_line('> Sign');
dbms_crypto_toolkit.Sign(persona => persona, input => string_input,
                        signature => signature);

```

**6. Verify the attached signature.**

```

dbms_output.put_line('> Verify');
dbms_crypto_toolkit.Verify(persona => persona,
                        signature => signature,
                        extracted_message => extracted_string,
                        verified => string_verified,
                        validated => string_validated,
                        signing_pary_identity => signing_party);

```

```

IF string_validated THEN
    dbms_output.put_line('> Validated');
END IF;
IF string_verified THEN
    dbms_output.put_line('> Verified');
END IF;

```

**7. Create a detached signature associated with the current persona.**

```

dbms_output.put_line('> Sign detached');
dbms_crypt_toolkit.SignDetached(persona => persona,

```

```
input => string_input,
signature => signature);
```

#### 8. Verify the detached signature.

```
dbms_output.put_line('> Verify detached');
dbms_crypto_toolkit.VerifyDetached(persona => persona,
                                   data => string_input,
                                   signature => signature,
                                   verified => string_verified,
                                   validated => string_validated,
                                   signing_party_identity => signing_party);

IF string_validated THEN
    dbms_output.put_line('> Validated');
END IF;
IF string_verified THEN
    dbms_output.put_line('> Verified');
END IF;
```

#### 9. Generate a hash of the current message.

```
dbms_output.put_line('> Hash');
dbms_crypto_toolkit.Hash(persona => persona,
                          input => string_input,
                          hash => hash_string);

IF string_input = hash_string THEN
    dbms_output.put_line('> Hash Succeeded');
END IF;

all_done := TRUE
RAISE done_exception;

EXCEPTION

WHEN others THEN
```

#### 10. Close the current open persona.

```
IF persona_opened THEN
    dbms_output.put_line('>ClosePersona.ClosePersona');
    dbms_crypto_toolkit.ClosePersona(persona);
END IF;
```

```
BEGIN
```

**11. Close the current open persona.**

```
IF persona_opened THEN
    dbms_output.put_line('> ClosePersona');
    dbms_crypto_toolkit.ClosePersona(persona);
END IF;
```

**12. Close the open wallet.**

```
IF wallet_opened THEN
    dbms_output.put_line('> CloseWallet');
    dbms_crypto_toolkit.CloseWallet(wallet);
END IF;
```

**13. Stop the Oracle Cryptographic Toolkit operation.**

```
IF initialized THEN
    dbms_output.put_line('> Terminate');
    dbms_crypto_toolkit.TERMINATE;
END IF;
```

```
IF all_done = FALSE THEN
    RAISE;
```

```
END;
```



---

## OCI - API Mappings

This chapter lists each Oracle Call Interface (OCI) function that is directly mapped to an Application Programming Interface (API) function. Definitions for each function are also provided. The following topics are discussed:

- “Mappings”
- “OCI - API Mapping Exceptions”

## B.1 Mappings

### B.1.1 Overview

The Oracle Call Interface functions are direct mappings from the Oracle Security Server Toolkit Application Programming Interface to the Oracle Call Interface.

### B.1.2 OCI - API Mappings

Table B–1, “OCI Function Names and Descriptions”, below lists each Oracle Security Server OCI function along with its description.

**Table B–1 OCI Function Names and Descriptions**

OCI Name	Description
OCISecurityOpenWallet	Open a wallet based on a WRL
OCISecurityCloseWallet	Close a wallet
OCISecurityCreateWallet	Create a new wallet
OCISecurityDestroyWallet	Destroy an existing wallet
OCISecurityStorePersona	Store a persona in a wallet
OCISecurityOpenPersona	Open a persona
OCISecurityClosePersona	Close a persona
OCISecurityRemovePersona	Remove a persona from a wallet
OCISecurityCreatePersona	Create a persona
OCISecuritySetProtection	Modify the protection set in a persona
OCISecurityGetProtection	Get the protection set in a persona
OCISecurityRemoveIdentity	Remove an identity from a persona
OCISecurityCreateIdentity	Create an Identity
OCISecurityAbortIdentity	Discard an unstored identity
OCISecurityStoreTrusted Identity	Store an identity with an associated trust
OCISecuritySign	Generate an attached signature
OCISecuritySignExpansion	Determine the size of the attached signature buffer
OCISecurityVerify	Verify an attached signature

**Table B–1   OCI Function Names and Descriptions**

OCI Name	Description
OCISecurityValidate	Validate an identity
OCISecuritySignDetached	Generate a detached signature
OCISecuritySignDetExpansion	Determine the size of buffer needed
OCISecurityVerifyDetached	Verify a detached signature
OCISecurityKeyedHash	Generate a keyed hash
OCISecurityKeyedHashExpansion	Determine the space needed for a keyed hash
OCISecurityHash	Generate a hash
OCISecurityHashExpansion	Determine the size of the TDU for the hash
OCISecuritySeedRandom	supplies a seed to the Oracle Cryptographic Toolkit's random number generator
OCISecurityRandomBytes	Generate a series of random bytes
OCISecurityRandomNumber	Generate a random number
OCISecurityInitBlock	Initialize a buffer block
OCISecurityReuseBlock	Reuse a buffer block
OCISecurityPurgeBlock	Purge the memory used within a buffer block
OCISecuritySetBlock	Set the block to a known state

**B.2   OCI - API Mapping Exceptions**

There are no OCI - API mapping exceptions at this time.





---

# Glossary

## **API**

See Application Programming Interface.

## **Application Programming Interface**

A set of functions that allow applications written in C or C++ to communicate with an operating system and issue SQL statements to one or more Oracle servers.

## **Certificate**

A document that uses the signature of a trusted party to attest to the validity of its information.

## **Ciphertext**

The result of encrypting data into an apparently random and meaningless format. Ciphertext must be decrypted to be converted into a readable format.

## **Decrypt**

To restore an encrypted message to its original form, so the original message is readable.

## **Digital Signature**

A cryptographic checksum of data encrypted using an entity's private key. The result authenticates the signature as having been generated by an entity, and it protects the data from tampering, since the signature can be verified.

A digital signature is an example of a message. If the message is a PKCS#7 message, the message is considered to be in PKCS format.

**Encrypt**

The transformation of data into an apparently random and meaningless format (called ciphertext). The ciphertext is unreadable by anyone without the correct decryption key.

**Entity**

A person (physical, imaginary, or otherwise) or a process.

**Handle**

A pointer to a storage area allocated by the API library.

**Identity**

The binding of a public key and other information to an entity. It is possible to have more than one identity bound to an entity. Every identity has a type. Some better known identity types are X.509 certificates and PGP certificates.

**MD5**

A message-digest hashing algorithm that compresses a message of arbitrary length into a 128-bit digest.

**Message Format**

The message format describes the layout and the contents of a message such as a digital signature.

**OCI**

See Oracle Call Interface.

**Oracle Call Interface**

An application programming interface that allows applications written in C to interact with one or more Oracle servers. See *Programmer's Guide to the Oracle Call Interface*.

**Persona**

An instance of your electronic personality. Each instance contains one or more elements such as an identity, the private key associated with the identity, and other cipher keys. An entity may have more than one persona. A persona implies a set of actions that can be used and a set of message formats that can be generated.

**PL/SQL**

PL/SQL is Oracle Corporation's procedural language extension to Structured Query Language (SQL).

**RC4**

An encryption algorithm.

**Repository IO**

An abstraction from the various repositories (e.g., file, database, hardware) used by the wallet interface.

**RIO**

See Repository IO.

**Sign**

Data is signed using a persona from a wallet. The result may be formatted in a number of ways and may contain only the digital signature. The signed data may also contain the original data, possibly encrypted, along with information about the identity used for the signature.

**SQL**

See Structured Query Language.

**Structured Query Language**

A language used to query and manipulate databases.

**TDU**

See Toolkit Data Unit.

**Toolkit Data Unit**

An encoding of possibly formatted and/or cryptographically altered data that is created by an application via the Oracle Security Server Toolkit. The toolkit data unit is usually transferred to another application that uses the Oracle Security Server Toolkit to decode the toolkit data unit back into data.

A toolkit data unit is the message granularity of the Oracle Security Server Toolkit, and it is transport independent.

**Trustpoint**

One or more identities that are considered trustworthy and can be used to validate other identities.

**Verify**

A formatted message that results from signing is verified using the identity that signed the message. Verifying the signature does not mean that the data can be trusted. The identity associated with the message should be validated using a trust-point.

**Wallet**

A facility that acts as a container for credentials (identities, personas, and trust-points). Each entity has one or more wallets, and each wallet, while logically identical, may exist on a file system or on a hardware device. The wallet may be password protected.

A wallet may be shared (read only) across a network. In this case, the wallet should only contain public information (i.e., identities and trust points).

**Wallet Resource Locator**

Specifies the wallet location.

**WRL**

See Wallet Resource Locator.

---

# Index

## A

---

API Interfaces, 5  
API Layer, 5  
Attached sign/verify, 5

## C

---

Certificate Authority (CA), 2  
Certificate Management Services, 3  
Certificate Revocation List (CRL), 3  
Checksums  
    generating, 12  
Concepts  
    Cryptographic Engine, 4  
    Detached Signature, 4  
    Entity, 4  
    Enveloping, 4  
    Identity, 4  
    Persona, 4  
    Personal Resource Locator, 4  
    Protection Set, 5  
    Recipient Oriented Encryption, 5  
    security, 2  
    Signature, 5  
    Symmetric Encryption, 5  
    Toolkit Data Unit, 5  
    Trust Point, 6  
    Wallet, 6  
Cryptographic Engine functions, 5

## D

---

data structures, 5

Data type names, 2  
DBMS\_RANDOM, 15  
Definitions  
    Authentication, 2  
    Authorization, 2  
    Certificate, 2  
    Certificate Authority, 2  
    Confidentiality, 2  
    Cryptography, 2  
    Decryption, 2  
    Encryption, 3  
    Integrity, 3  
    Non-repudiation, 3  
    Oracle Cryptographic Toolkit, 4  
    Oracle Security Server, 2  
    Public/Private Key Pair, 3  
    Public-Key Encryption, 3  
    X.509, 3  
Detached sign/verify, 6  
Digital signatures  
    PL/SQL routines for, 7

## E

---

Examples  
    Generate a detached signature for an array of  
        bytes, 5  
    Random Number Generator, 2

## F

---

Features  
    Oracle Security Server, 2  
Functions

- Cryptographic Engine, 5
- OCI, 1
  - OCISecurityClosePersona, 7
  - OCISecurityCloseWallet, 5
  - OCISecurityHash, 15
  - OCISecurityInitBlock, 19
  - OCISecurityInitialize, 2
  - OCISecurityOpenPersona, 6
  - OCISecurityOpenWallet, 4
  - OCISecurityPurgeBlock, 21
  - OCISecurityRandomBytes, 17
  - OCISecurityRandomNumber, 18
  - OCISecurityReuseBlock, 20
  - OCISecuritySeedRandom, 16
  - OCISecuritySetBlock, 22
  - OCISecuritySign, 8
  - OCISecuritySignDetached, 12
  - OCISecurityTerminate, 3
  - OCISecurityValidate, 11
  - OCISecurityVerify, 9
  - OCISecurityVerifyDetached, 13
- Oracle Call Interface. See Functions
- OCI
- Persona/Identity, 6
- PL/SQL
  - Digital Signature, 7
  - General Purpose, 2
  - Hash, 12
  - Random Number Generation, 15
  - Use Oracle Wallet, 3
- Wallet, 6

## H

---

- Hash, 6

## I

---

- Identity
  - definition of, 7
- Interfaces
  - Oracle call interface, 10
  - PL/SQL, 10

## K

---

- Keyed hash, 6

## M

---

- Mapping
  - Exceptions, 3
  - Overview, 2

## O

---

- Oracle Call Interface, 10
- Oracle Enterprise Manager, 3
- Oracle Security Server Manager, 3

## P

---

- Persona
  - definition of, 8
- PL/SQL functions
  - AbortIdentity, 5
  - ClosePersona, 4
  - CloseWallet, 3
  - CreateIdentity, 5
  - CreatePersona, 4
  - DestroyWallet, 3
  - iInitialize, 2
  - OpenPersona, 4
  - OpenWallet, 2
  - RemoveIdentity, 5
  - RemovePersona, 4
  - StorePersona, 3
  - StoreTrustedIdentity, 6
  - Terminate, 2
  - Validate, 6
- PL/SQL interface, 10
- PL/SQL routines
  - Hash, 14
  - KeyedHash, 13
  - Random, 15
  - SeedRandom, 15
  - Sign, 8
  - SignDetached, 10
  - Verify, 9
  - VerifyDetached, 11

## Prefixes

data type names, 2

Program Flow, 2

## Programming Steps

Interface with the Oracle Security Server, 3

## R

---

Random Number Generator, 2

Example, 2

Functions, 2

## Relationship

between Oracle Cryptographic Toolkit and  
Oracle Security Server Services, 9

## S

---

## Sample

PL/SQL Program, 2

Security concepts, 2

## Signatures

DSS, 5

RSA, 5

## T

---

## Toolkit

Elements of, 7

## Trusted Identity

definition of, 8

## W

---

## Wallet

definition of, 9

## X

---

X.509 v1 Certificate, 2

