# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>xv</td>
</tr>
<tr>
<td>Intended Audience</td>
<td>xv</td>
</tr>
<tr>
<td>Documentation Accessibility</td>
<td>xv</td>
</tr>
<tr>
<td>Related Documents</td>
<td>xvi</td>
</tr>
<tr>
<td>Conventions</td>
<td>xvi</td>
</tr>
<tr>
<td><strong>1 Introduction to Oracle Security Developer Tools</strong></td>
<td>1-1</td>
</tr>
<tr>
<td>Cryptography</td>
<td>1-1</td>
</tr>
<tr>
<td>Types of Cryptographic Algorithms</td>
<td>1-2</td>
</tr>
<tr>
<td>Symmetric Cryptographic Algorithms</td>
<td>1-2</td>
</tr>
<tr>
<td>Asymmetric Cryptographic Algorithms</td>
<td>1-2</td>
</tr>
<tr>
<td>Hash Functions</td>
<td>1-3</td>
</tr>
<tr>
<td>Additional Cryptography Resources</td>
<td>1-3</td>
</tr>
<tr>
<td>Public Key Infrastructure (PKI)</td>
<td>1-3</td>
</tr>
<tr>
<td>Key Pairs</td>
<td>1-3</td>
</tr>
<tr>
<td>Certificate Authority</td>
<td>1-4</td>
</tr>
<tr>
<td>Digital Certificates</td>
<td>1-4</td>
</tr>
<tr>
<td>Related PKI Standards</td>
<td>1-4</td>
</tr>
<tr>
<td>Benefits of PKI</td>
<td>1-5</td>
</tr>
<tr>
<td>Web Services Security</td>
<td>1-6</td>
</tr>
<tr>
<td>SAML</td>
<td>1-6</td>
</tr>
<tr>
<td>SAML Assertions</td>
<td>1-7</td>
</tr>
<tr>
<td>SAML Requests and Responses</td>
<td>1-8</td>
</tr>
<tr>
<td>SAML Request and Response Cycle</td>
<td>1-8</td>
</tr>
<tr>
<td>SAML Protocol Bindings and Profiles</td>
<td>1-9</td>
</tr>
<tr>
<td>SAML and XML Security</td>
<td>1-9</td>
</tr>
<tr>
<td>Federation</td>
<td>1-9</td>
</tr>
<tr>
<td>Overview of Oracle Security Developer Tools</td>
<td>1-10</td>
</tr>
<tr>
<td>Oracle Crypto</td>
<td>1-11</td>
</tr>
<tr>
<td>Oracle Security Engine</td>
<td>1-11</td>
</tr>
<tr>
<td>Oracle CMS</td>
<td>1-12</td>
</tr>
<tr>
<td>Oracle S/MIME</td>
<td>1-12</td>
</tr>
<tr>
<td>Oracle PKI SDK</td>
<td>1-12</td>
</tr>
<tr>
<td>Oracle PKI SDK LDAP</td>
<td>1-12</td>
</tr>
<tr>
<td>Oracle PKI SDK TSP</td>
<td>1-12</td>
</tr>
</tbody>
</table>
# Oracle JCE Provider Features and Benefits

Oracle JCE Provider Features and Benefits

Oracle JCE Provider

Oracle XML Security

Oracle SAML

Oracle Web Services Security

Oracle Liberty SDK

Oracle XKMS

## Oracle Crypto

### Oracle Crypto Features and Benefits

Oracle Crypto Features and Benefits

Oracle Crypto Packages

### Setting Up Your Oracle Crypto Environment

Setting Up Your Oracle Crypto Environment

System Requirements for Oracle Crypto

Setting the CLASSPATH Environment Variable

Setting the CLASSPATH on Windows

Setting the CLASSPATH on UNIX

### Core Classes and Interfaces

Core Classes and Interfaces

Keys

The oracle.security.crypto.core.Key Interface

The oracle.security.crypto.core.PrivateKey Interface

The oracle.security.crypto.core.PublicKey Interface

The oracle.security.crypto.core.SymmetricKey Interface

Key Generation

The oracle.security.crypto.core.KeyPairGenerator Class

The oracle.security.crypto.core.SymmetricKeyGenerator Class

Ciphers

Symmetric Ciphers

The RSA Cipher

Password Based Encryption

Signatures

Message Digests

The oracle.security.crypto.core.MessageDigest Class

The oracle.security.crypto.core.MAC Class

Key Agreement

Pseudo-Random Number Generators

The oracle.security.crypto.core.RandomBitsSource class

The oracle.security.crypto.core.EntropySource class

The Oracle Crypto Java API Reference

Example Programs

## Oracle JCE Provider

### Oracle JCE Provider Features and Benefits

Oracle JCE Provider Features and Benefits

Using the Oracle JCE Provider

### Setting Up Your Oracle JCE Provider Environment

Setting Up Your Oracle JCE Provider Environment

System Requirements for Oracle JCE Provider

Installation Requirements
4 Oracle Security Engine

Oracle Security Engine Features and Benefits ................................................................. 4-1
Oracle Security Engine Packages .................................................................................. 4-1
Setting Up Your Oracle Security Engine Environment .................................................. 4-2
System Requirements for Oracle Security Engine .......................................................... 4-2
Setting the CLASSPATH Environment Variable .............................................................. 4-2
Setting the CLASSPATH on Windows .............................................................................. 4-2
Setting the CLASSPATH on UNIX .................................................................................. 4-2
Core Classes and Interfaces ............................................................................................. 4-3
The oracle.security.crypto.cert.X500RDN Class ............................................................... 4-3
The oracle.security.crypto.cert.X500Name Class ............................................................ 4-3
The oracle.security.crypto.cert.CertificateRequest Class .............................................. 4-4
The oracle.security.crypto.cert.X509 Class .................................................................... 4-5
The Oracle Security Engine Java API Reference ............................................................ 4-5
Example Programs .......................................................................................................... 4-5

5 Oracle CMS

Oracle CMS Features and Benefits ................................................................................ 5-1
Content Types ................................................................................................................ 5-1
Differences Between Oracle CMS and PKCS #7 Version 1.5 ........................................... 5-2
Setting Up Your Oracle CMS Environment ..................................................................... 5-2
System Requirements ...................................................................................................... 5-2
Setting the CLASSPATH Environment Variable .............................................................. 5-2
Setting the CLASSPATH on Windows .............................................................................. 5-3
Setting the CLASSPATH on UNIX .................................................................................. 5-3
Developing Applications with Oracle CMS ..................................................................... 5-3
CMS Object Types ........................................................................................................... 5-4
Constructing CMS Objects using the CMS***ContentInfo Classes ................................ 5-4
Abstract Base Class CMSContentInfo ............................................................................... 5-4
Constructing a CMS Object ............................................................................................ 5-5
Reading a CMS Object ................................................................................................. 5-5
The CMSDataContentInfo Class .................................................................................... 5-5
The ESSReceipt Class .................................................................................................... 5-6
The CMSDigestedDataContentInfo Class ....................................................................... 5-7
Constructing a CMS Digested-data Object ..................................................................... 5-8
Reading a CMS Digested-data Object ........................................................................... 5-8
Detached digested-data Objects .................................................................................... 5-8
The CMSSignedDataContentInfo Class .......................................................................... 5-9
Constructing a CMS Signed-data Object ....................................................................... 5-10
Reading a CMS Signed-data Object ............................................................................ 5-11
External Signatures (Detached Objects) .......................................................................... 5-12
Oracle S/MIME Features and Benefits

Setting Up Your Oracle S/MIME Environment

System Requirements for Oracle S/MIME

Setting the CLASSPATH Environment Variable

Setting the CLASSPATH on Windows

Setting the CLASSPATH on UNIX

Developing Applications with Oracle S/MIME

Core Classes and Interfaces

The oracle.security.crypto.smime.SmimeObject Interface

The oracle.security.crypto.smime.SmimeSignedObject Interface

The oracle.security.crypto.smime.SmimeSigned Class

The oracle.security.crypto.smime.SmimeEnveloped Class

The oracle.security.crypto.smime.SmimeMultipartSigned Class

Example Programs
The oracle.security.crypto.smime.SmimSignedReceipt Class ........................................ 6-6
The oracle.security.crypto.smime.SmimCompressed Class ........................................ 6-7
Supporting Classes and Interfaces .................................................................................. 6-8
The oracle.security.crypto.smime.Smim Interface .......................................................... 6-8
The oracle.security.crypto.smime.SmimUtils Class ......................................................... 6-8
The oracle.security.crypto.smime.MailTrustPolicy Class ................................................ 6-8
The oracle.security.crypto.smime.SmimCapabilities Class ............................................ 6-8
The oracle.security.crypto.smime.SmimDataContentHandler Class ............................. 6-9
The oracle.security.crypto.smime.ess Package ................................................................ 6-9
Using the Oracle S/MIME Classes .................................................................................. 6-9
Using the Abstract Class SmimObject ............................................................................. 6-10
Signing Messages .......................................................................................................... 6-10
Creating "Multipart/Signed" Entities ............................................................................. 6-11
Creating Digital Envelopes ............................................................................................ 6-11
Creating "Certificates-Only" Messages ......................................................................... 6-12
Reading Messages .......................................................................................................... 6-12
Authenticating Signed Messages .................................................................................... 6-12
Opening Digital Envelopes (Encrypted Messages) ......................................................... 6-13
Adding Enhanced Security Services (ESS) ................................................................. 6-14
Processing Enhanced Security Services (ESS) ............................................................. 6-14
The Oracle S/MIME Java API Reference ......................................................................... 6-14
Example Programs ........................................................................................................ 6-15

7 Oracle PKI SDK

Oracle PKI SDK CMP ....................................................................................................... 7-1
Oracle PKI SDK CMP Features and Benefits ................................................................. 7-1
Package Overview for Oracle PKI SDK CMP ............................................................... 7-2
Setting Up Your Oracle PKI SDK CMP Environment ................................................. 7-2
System Requirements for Oracle PKI SDK CMP ....................................................... 7-2
Setting the CLASSPATH Environment Variable ....................................................... 7-2
Setting the CLASSPATH on Windows .......................................................................... 7-2
Setting the CLASSPATH on UNIX ................................................................................. 7-3
The Oracle PKI SDK CMP Java API Reference ............................................................ 7-3
Example Programs ........................................................................................................ 7-3

Oracle PKI SDK OCSP .................................................................................................... 7-3
Oracle PKI SDK OCSP Features and Benefits ............................................................. 7-3
Setting Up Your Oracle PKI SDK OCSP Environment .............................................. 7-4
System Requirements for Oracle PKI SDK OCSP ..................................................... 7-4
Setting the CLASSPATH Environment Variable ....................................................... 7-4
Setting the CLASSPATH on Windows .......................................................................... 7-4
Setting the CLASSPATH on Unix .................................................................................. 7-4
The Oracle PKI SDK OCSP Java API Reference ............................................................ 7-5
Example Programs ........................................................................................................ 7-5

Oracle PKI SDK TSP ....................................................................................................... 7-5
Oracle PKI SDK TSP Features and Benefits ................................................................. 7-5
Class and Interface Overview for Oracle PKI SDK TSP ............................................ 7-5
 Oracle SAML Features and Benefits ................................................................. 9-1
 Oracle XML Security Features and Benefits ......................................................... 8-2

 8 Oracle XML Security

 Oracle XML Security Features and Benefits ................................................................. 8-2
 Setting Up Your Oracle XML Security Environment ................................................... 8-2
 System Requirements for Oracle XML Security ......................................................... 8-2
 Setting the CLASSPATH Environment Variable ....................................................... 8-3
 Setting the CLASSPATH on Windows ......................................................................... 8-3
 Setting the CLASSPATH on UNIX .............................................................................. 8-4
 Classes and Interfaces ............................................................................................... 8-4
 Core Classes ........................................................................................................... 8-4
 The oracle.security.xmlsec.dsig.XSSignature Class ............................................... 8-4
 The oracle.security.xmlsec.dsig.XSSignedInfo Class ............................................ 8-5
 The oracle.security.xmlsec.dsig.XSReference class ............................................. 8-5
 The oracle.security.xmlsec.dsig.XSKeyInfo class .................................................. 8-6
 The oracle.security.xmlsec.enc.XEEncryptedData class ......................................... 8-6
 The oracle.security.xmlsec.enc.XEEncryptedKey Class .......................................... 8-7
 The oracle.security.xmlsec.enc.XEEncryptionMethod Class .................................... 8-7
 The oracle.security.xmlsec.enc.XECipherData Class .............................................. 8-8
 Supporting Classes and Interfaces ........................................................................... 8-8
 The oracle.security.xmlsec.util.XMLURI Interface ............................................... 8-8
 The oracle.security.xmlsec.util.XMLeUtilis class ............................................... 8-9
 Common XML Security Questions ........................................................................... 8-9
 Common Questions about Keys and Certificates .................................................... 8-9
 Common Questions about XML Signatures ............................................................ 8-9
 Common Questions about XML Encryption ............................................................ 8-10
 The Oracle XML Security Java API Reference ....................................................... 8-10
 Example Programs .................................................................................................. 8-10

 9 Oracle SAML

 Oracle SAML Features and Benefits ......................................................................... 9-1
11 Oracle Liberty SDK

Oracle Liberty SDK Features and Benefits ................................................................. 11-1
Oracle Liberty 1.1 ....................................................................................................... 11-2
  Setting Up Your Oracle Liberty 1.1 Environment .................................................. 11-2
  Setting the CLASSPATH Environment Variable .................................................. 11-2
  Setting the CLASSPATH on Windows ................................................................. 11-2
  Setting the CLASSPATH on UNIX ...................................................................... 11-3
  Overview of Oracle Liberty 1.1 Classes and Interfaces ........................................ 11-3
  Core Classes and Interfaces ................................................................................. 11-3
    The oracle.security.xmlsec.liberty.v11.AuthnRequest Class ......................... 11-3
    The oracle.security.xmlsec.liberty.v11.AuthnResponse Class ....................... 11-4
    The oracle.security.xmlsec.liberty.v11.FederationTerminationNotification Class 11-4
    The oracle.security.xmlsec.liberty.v11.LogoutRequest Class ......................... 11-5
    The oracle.security.xmlsec.liberty.v11.LogoutResponse Class ....................... 11-6
    The oracle.security.xmlsec.liberty.v11.RegisterNameIdentifierRequest Class ... 11-6
    The oracle.security.xmlsec.liberty.v11.RegisterNameIdentifierResponse Class 11-7
  Supporting Classes and Interfaces ................................................................. 11-8
    The oracle.security.xmlsec.liberty.v11.LibertyInitializer class ..................... 11-8
    The oracle.security.xmlsec.liberty.v11.LibertyURI interface ......................... 11-8
    The oracle.security.xmlsec.liberty.v11.ac.AuthenticationContextURI interface ... 11-8
    The oracle.security.xmlsec.util.ac.AuthenticationContextStatement class ....... 11-9
    The oracle.security.xmlsec.saml.SAMLURI Interface ...................................... 11-9
    The oracle.security.xmlsec.saml.SAMLMessage class .................................... 11-9
  The Oracle Liberty SDK 1.1 API Reference ......................................................... 11-9
  Example Programs .............................................................................................. 11-9
Oracle Liberty 1.2 .................................................................................................. 11-10
  Setting Up Your Oracle Liberty 1.2 Environment .............................................. 11-10
  Setting the CLASSPATH Environment Variable ................................................ 11-10
  Setting the CLASSPATH on Windows ............................................................... 11-10
  Setting the CLASSPATH on Unix ................................................................. 11-10
  Overview of Oracle Liberty 1.2 Classes and Interfaces ...................................... 11-11
  Core Classes and Interfaces ............................................................................... 11-11
    The oracle.security.xmlsec.saml.Assertion class ........................................... 11-11
    The oracle.security.xmlsec.samlp.Request class .......................................... 11-12
    The oracle.security.xmlsec.samlp.Response class ....................................... 11-12
12 Oracle XKMS

Oracle XKMS Features and Benefits ................................................................. 12-1
Oracle XKMS Packages ...................................................................................... 12-1

Setting Up Your Oracle XKMS Environment .................................................... 12-2
System Requirements for Oracle XKMS ............................................................ 12-2
Setting the CLASSPATH Environment Variable ............................................... 12-2
Setting the CLASSPATH on Windows ............................................................... 12-2
Setting the CLASSPATH on UNIX ................................................................. 12-3

Core Classes and Interfaces ............................................................................ 12-3
oracle.security.xmlsec.xkms.xkiss.LocateRequest ........................................... 12-3
oracle.security.xmlsec.xkms.xkiss.LocateResult .............................................. 12-4
oracle.security.xmlsec.xkms.xkiss.ValidateRequest ........................................ 12-4
oracle.security.xmlsec.xkms.xkiss.ValidateResult .......................................... 12-5
oracle.security.xmlsec.xkms.xkrss.RecoverRequest ....................................... 12-6
oracle.security.xmlsec.xkms.xkrss.RecoverResult .......................................... 12-6

The Oracle XKMS Java API Reference ............................................................. 12-7
Example Programs .............................................................................................. 12-8

A References

Glossary

Index
List of Figures

1–1    SAML Request-Response Cycle................................................................................................ 1-8
1–2    The Oracle Security Developer Tools.................................................................................... 1-10
## List of Tables

<table>
<thead>
<tr>
<th>Page</th>
<th>Table Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Summary of Public and Private Key Usage</td>
</tr>
<tr>
<td>5-1</td>
<td>Content Types Supported by Oracle CMS</td>
</tr>
<tr>
<td>5-2</td>
<td>CMS***ContentInfo Classes</td>
</tr>
<tr>
<td>5-3</td>
<td>Useful Methods of CMSContentInfo</td>
</tr>
<tr>
<td>5-4</td>
<td>Useful Methods of ESSReceipt</td>
</tr>
<tr>
<td>5-5</td>
<td>Useful Methods of CMSDigestedDataContentInfo</td>
</tr>
<tr>
<td>5-6</td>
<td>Useful Methods of CMSSignedDataContentInfo</td>
</tr>
<tr>
<td>5-7</td>
<td>Useful Methods of CMSEncryptedDataContentInfo</td>
</tr>
<tr>
<td>5-8</td>
<td>Useful Methods of CMSEnvelopedDataContentInfo</td>
</tr>
<tr>
<td>5-9</td>
<td>Useful Methods of CMSAuthenticatedDataContentInfo</td>
</tr>
<tr>
<td>5-10</td>
<td>The CMS***Stream Classes</td>
</tr>
<tr>
<td>5-11</td>
<td>The CMS***Connector Classes</td>
</tr>
<tr>
<td>6-1</td>
<td>Classes in the oracle.security.crypto.smime.ess Package</td>
</tr>
<tr>
<td>7-1</td>
<td>Oracle PKI SDK TSP Classes and Interfaces</td>
</tr>
<tr>
<td>10-1</td>
<td>Packages in the Oracle Web Services Security Library</td>
</tr>
<tr>
<td>12-1</td>
<td>Packages in the Oracle XKMS Library</td>
</tr>
<tr>
<td>A-1</td>
<td>Security Standards and Protocols</td>
</tr>
</tbody>
</table>
The *Oracle Security Developer Tools Reference* provides reference information about the Oracle Security Developer Tools. This Preface contains the following topics:

- Intended Audience
- Documentation Accessibility
- Related Documents
- Conventions

**Intended Audience**

*Oracle Security Developer Tools Reference* is intended for Java developers responsible for developing secure applications. This documentation assumes programming proficiency using Java, and familiarity with security concepts such as cryptography, public key infrastructure, Web services security, and identity federation.

**Documentation Accessibility**

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


**Accessibility of Code Examples in Documentation**

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

**Accessibility of Links to External Web Sites in Documentation**

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

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

Related Documents

For more information, see the following documentation available in the Oracle Application Server 10g (10.1.4.0.1) documentation set:

- Oracle Identity Management User Reference

Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><code>monospace</code></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
Security tools are a critical component for today's application development projects. Commercial requirements and government regulations dictate that sensitive data be kept confidential and protected from tampering or alteration.

Oracle Security Developer Tools provide you with the cryptographic building blocks necessary for developing robust security applications, ranging from basic tasks like secure messaging to more complex projects such as securely implementing a service-oriented architecture. The tools build upon the core foundations of cryptography, public key infrastructure, web services security, and federated identity management.

This chapter takes a closer look at these underlying security technologies and introduces the components of the Oracle Security Developer Tools. It covers these topics:
- Cryptography
- Public Key Infrastructure (PKI)
- Web Services Security
- SAML
- Federation
- Overview of Oracle Security Developer Tools

**Cryptography**

As data travels across untrusted communication channels, cryptography protects the transmitted messages from being intercepted (a passive attack) or modified (an active attack) by an intruder. To protect the message, an originator uses a cryptographic tool to convert plain, readable messages or plaintext into encrypted ciphertext. While the original text is present, its appearance changes into a form that is unintelligible if intercepted. The message recipient likewise uses a cryptographic tool to decrypt the ciphertext into its original readable format.

Cryptography secures communications over a network such as the internet by providing:
- Authentication, which assures the receiver that the information is coming from a trusted source. Authentication is commonly achieved through the use of a Message Authentication Code (MAC), digital signature, and digital certificate.
- Confidentiality, which ensures that only the intended receiver can read a message. Confidentiality is commonly attained through encryption.

- Integrity, which ensures that the received message has not been altered from the original. Integrity is commonly ensured by using a cryptographic hash function.

- Non-repudiation, which is a way to prove that a given sender actually sent a particular message. Non-repudiation is typically achieved through the use of digital signatures.

### Types of Cryptographic Algorithms

The mathematical operations used to map between plaintext and ciphertext are identified by a **cryptographic algorithm** (also known as a **cipher**). Cryptographic algorithms require the text to be mapped, and, at a minimum, require some value which controls the mapping process. This value is called a **key**.

Essentially, there are three types of cryptographic algorithms which can be categorized by the number of keys used for encryption and decryption, and by their application and usage. The basic types of cryptographic algorithms are:

- **Symmetric Cryptographic Algorithms**
- **Asymmetric Cryptographic Algorithms**
- **Hash Functions**

Each type is optimized for certain applications. Hash functions are suited for ensuring data integrity. Symmetric cryptography is ideally suited for encrypting messages. Asymmetric cryptography is used for the secure exchange of keys, authentication, and non-repudiation. Asymmetric cryptography could also be used to encrypt messages, although this is rarely done. Symmetric cryptography operates about 1000 times faster, and is better suited for encryption than asymmetric cryptography.

#### Symmetric Cryptographic Algorithms

A **symmetric cryptography** algorithm (also known as **secret key cryptography**) uses a single key for both encryption and decryption. The sender uses the key to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key to decrypt the message and recover the plaintext. The key must be known to both the sender and receiver. The biggest problem with symmetric cryptography is the secure distribution of the key.

Symmetric cryptography schemes are generally categorized as being either a **block cipher** or **stream cipher**. A block cipher encrypts one fixed-size block of data (usually 64 bits) at a time using the same key on each block. Some common block ciphers used today include **Blowfish**, **AES**, **DES**, and **3DES**.

Stream ciphers operate on a single bit at a time and implement some form of feedback mechanism so that the key is constantly changing. **RC4** is an example of a stream cipher that is used for secure communications using the SSL protocol.

#### Asymmetric Cryptographic Algorithms

An **asymmetric cryptography** algorithm (also known as **public key cryptography**) uses one key to encrypt the plaintext and another key to decrypt the ciphertext. It does not matter which key is applied first, but both keys are required for the process to work.

In asymmetric cryptography, one of the keys is designated the **public key** and is made widely available. The other key is designated the **private key** and is never revealed to
another party. To send messages under this scheme, the sender encrypts some information using the receiver’s public key. The receiver then decrypts the ciphertext using her private key. This method can also be used to prove who sent a message (non-repudiation). The sender can encrypt some plaintext with her private key, and when the receiver decrypts the message with the sender’s public key, the receiver knows that the message was indeed sent by that sender.

Some of the common asymmetric algorithms in use today are RSA, DSA, Diffie-Hellman, and Elliptic Curve Cryptography (ECC).

Hash Functions
A hash function (also known as a message digest) is a one-way encryption algorithm that essentially uses no key. Instead, a fixed-length hash value is computed based upon the plaintext that makes it impossible for either the contents or length of the plaintext to be recovered. Hash algorithms are typically used to provide a digital fingerprint of a file’s contents, often used to ensure that the file has not been altered by an intruder or virus. Hash functions are also commonly employed by many operating systems to encrypt passwords. Hash functions help preserve the integrity of a file. Some common hash functions include MD2, MD4, MD5 and SHA.

Additional Cryptography Resources
For more information, refer to the cryptography resources listed in Appendix A.

Public Key Infrastructure (PKI)
A public key infrastructure (PKI) is designed to enable secure communications over public and private networks. Besides secure transmission and storage of data, PKI enables secure e-mail, digital signatures, and data integrity.

These facilities are delivered using public key cryptography, a mathematical technique that uses a pair of related cryptographic keys to verify the identity of the sender (digital signature), or to ensure the privacy of a message (encryption). PKI facilities support secure information exchange over insecure networks, such as the Internet.

Critical elements for achieving the goals of PKI include:

- Encryption algorithms and keys to secure communications
- Digital certificates that associate a public key with the identity of its owner
- Key distribution methods to permit widespread, secure use of encryption
- A trusted entity, known as a Certificate Authority (CA), to vouch for the relationship between a key and its legitimate owner
- A Registration Authority (RA) that is responsible for verifying the information supplied in requests for certificates made to the CA

Relying third parties use the certificates issued by the CA and the public keys contained therein to verify digital certificates and encrypt data.

Key Pairs
Encryption techniques often use a text or number called a key, known only to the sender and recipient.
When both use the same key, the encryption scheme is called symmetric. Difficulties with relying on a symmetric system include getting that key to both parties without allowing an eavesdropper to get it, too; and the fact that a separate key is needed for every two people, so that each individual must maintain many keys, one for each recipient.

Public key cryptography uses a key pair of mathematically related cryptographic keys - the public key and the private key. For an explanation of the use of key pairs, see "Asymmetric Cryptographic Algorithms".

Table 1–1 summarizes who uses public and private keys and when:

<table>
<thead>
<tr>
<th>Function</th>
<th>Key Type</th>
<th>Whose Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt data for a recipient</td>
<td>Public key</td>
<td>Receiver</td>
</tr>
<tr>
<td>Sign data</td>
<td>Private key</td>
<td>Sender</td>
</tr>
<tr>
<td>Decrypt data received</td>
<td>Private key</td>
<td>Receiver</td>
</tr>
<tr>
<td>Verify a signature</td>
<td>Public key</td>
<td>Sender</td>
</tr>
</tbody>
</table>

Certificate Authority

A Certificate Authority (CA) is a trusted third party that vouches for the public key owner's identity. Oracle Certificate Authority is one such entity. Others include Verisign and Thawte.

Digital Certificates

The certification authority validates the public key's link to a particular entity by creating a digital certificate. This digital certificate contains the public key and information about the key holder and the signing certification authority. Using a PKI certificate to authenticate one's identity is analogous to identifying oneself with a driver's license or passport.

Related PKI Standards

A number of standards and protocols support PKI certificate implementation.

Cryptographic Message Syntax

Cryptographic Message Syntax (CMS) is a general syntax for data protection developed by the Internet Engineering Task Force (IETF). It supports a wide variety of content types including signed data, enveloped data, digests, and encrypted data, among others. CMS allows multiple encapsulation so that, for example, previously signed data can be enveloped by a second party.

Values produced by CMS are encoded using X.509 Basic Encoding Rules (BER), meaning that the values are represented as octet strings.

Secure/Multipurpose Internet Mail Extension

Secure/Multipurpose Internet Mail Extension (S/MIME) is an Internet Engineering Task Force (IETF) standard for securing MIME data through the use of digital signatures and encryption.

S/MIME provides the following cryptographic security services for electronic messaging applications:
Public Key Infrastructure (PKI)

- Authentication
- Message integrity and non-repudiation of origin (using digital signatures)
- Privacy and data security (using encryption)

Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP) is the open standard for obtaining and posting information to commonly used directory servers. In a public key infrastructure (PKI) system, a user’s digital certificate is often stored in an LDAP directory and accessed as needed by requesting applications and services.

Time Stamp Protocol

In a Time Stamp Protocol (TSP) system, a trusted third-party Time Stamp Authority (TSA) issues time stamps for digital messages. Time stamping proves that a message was sent by a particular entity at a particular time, providing non-repudiation for online transactions.

The Time Stamp Protocol, as specified in RFC 3161, defines the participating entities, the message formats, and the transport protocol involved in time stamping a digital message.

To see how a time-stamping system can work, suppose Sally signs a document and wants it time stamped. She computes a message digest of the document using a secure hash function and then sends the message digest (but not the document itself) to the TSA, which sends her in return a digital time stamp consisting of the message digest, the date and time it was received at the TSA server, and the signature of the TSA. Since the message digest does not reveal any information about the content of the document, the TSA cannot eavesdrop on the documents it time stamps. Later, Sally can present the document and time stamp together to prove when the document was written. A verifier computes the message digest of the document, makes sure it matches the digest in the time stamp, and then verifies the signature of the TSA on the time stamp.

Online Certificate Status Protocol

Online Certificate Status Protocol (OCSP) is one of two common schemes for checking the validity of digital certificates. The other, older method, which OCSP has superseded in some scenarios, is known as the certificate revocation list (CRL).

OCSP overcomes the chief limitation of CRL: the fact that updates must be frequently downloaded to keep the list current at the client end. When a user attempts to access a server, OCSP sends a request for certificate status information. The server sends back a response of good, revoked, or unknown. The protocol specifies the syntax for communication between the server (which contains the certificate status) and the client application (which is informed of that status).

Certificate Management Protocol

The certificate management protocol (CMP) handles all relevant aspects of certificate creation and management. CMP supports interactions between public key infrastructure (PKI) components, such as Certificate Authorities (CAs), Registration Authorities (RAs), and end entities that are issued certificates.

Benefits of PKI

PKI provides users with the following benefits:

- Secure and reliable authentication of users
Reliable authentication relies on two factors. The first is proof of possession of the private key part of the public/private pair, which is verified by an automatic procedure that uses the public key. The second factor is validation by a certification authority that a public key belongs to a specific identity. A PKI-based digital certificate validates this identity connection based on the key pair.

- **Data integrity**

  Using the private key of a public/private key pair to sign digital transactions makes it difficult to alter the data in transit. This “digital signature” is a coded digest of the original message encrypted by the sender’s private key. Recipients can readily use the sender’s corresponding public key to verify who sent the message and the fact that it has not been altered. Any change to the message or the digest would have caused the attempted verification using the public key to fail, telling the recipient not to trust it.

- **Non-repudiation**

  PKI can also be used to prove who sent a message. The sender encrypts some plaintext with her private key to create a digital signature, and when the receiver decrypts the message with the sender’s public key, the receiver knows that the message was indeed sent by that sender, making it difficult for the message originator to disown the message; this capability is known as non-repudiation.

- **Prevention of unauthorized access to transmitted or stored information**

  The time and effort required to derive the private key from the public key makes it unlikely that the message would be decrypted by anyone other than the key pair owner.

---

**Web Services Security**

*Web services* provide a standard way for businesses and other organizations to integrate Web-based applications using open standards technologies such as *XML*, *SOAP*, and *WSDL*.

SOAP is a lightweight protocol for exchange of information in a service oriented environment. In such an environment, applications can expose selected functionality (business logic, for example) for use by other applications. SOAP provides the means by which applications supply and consume these services; it is an XML-based protocol for message transport in a distributed, decentralized Web Services application environment.

While the core SOAP specification solves many problems related to XML and Web Services, it does not provide a means to address message security requirements such as confidentiality, integrity, message authentication, and non-repudiation. The need for securing SOAP prompted *OASIS* to put forward the Web Services Security standard, which:

- Specifies enhancements to allow signing and encryption of SOAP messages
- Describes a general-purpose method for associating security tokens with messages
- Provides additional means for describing the characteristics of tokens that are included with a message

**SAML**

Security Assertions Markup Language (*SAML*) is an XML-based framework for exchanging security information over the Internet. SAML enables the exchange of
authentication and authorization information between various security services systems that otherwise would not be able to interoperate.

The SAML 1.0 and 2.0 specifications were adopted by the Organization for the Advancement of Structured Information Standards (OASIS) in 2002 and 2005 respectively. OASIS is a worldwide not-for-profit consortium that drives the development, convergence, and adoption of e-business standards.

SAML 2.0 marks the convergence of the Liberty ID-FF, Shibboleth, and SAML 1.0/1.1 federation protocols.

**SAML Assertions**

SAML associates an identity (such as an e-mail address or a directory listing) with a subject (such as a user or system) and defines the access rights within a specific domain. Every SAML document contains an **assertion** element. SAML defines four kinds of assertions, which are declarations of one or more facts about a subject:

- **Subject assertions**, which are used to identify a particular user or system.
- **Authentication assertions**, which state that the user has proven his identity by a particular method at a specific time.
- **Attribute assertions**, which contain specific details about the user such as an employee number or account number.
- **Authorization assertions**, which state the resources a user can access and under what conditions.

Assertions are coded statements generated about events that have already occurred. While SAML makes assertions about credentials, it does not actually authenticate or authorize users. **Example 1–1** shows a typical SAML authentication assertion wrapped in a SAML response message:

**Example 1–1  Sample SAMLP Response Containing a SAML 1.0 Authentication Assertion**

```
<samlp:Response
    MajorVersion="1" MinorVersion="0"
    ResponseID="128.14.234.20.90123456"
    InResponseTo="123.45.678.90.12345678"
    IssueInstant="2005-12-14T10:00:23Z"
    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
    xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
    <samlp:Status>
        <samlp:StatusCode Value="samlp:Success" />
    </samlp:Status>
    <saml:Assertion
        MajorVersion="1" MinorVersion="0"
        AssertionID="123.45.678.90.12345678"
        Issuer="IssuingAuthority.com"
        IssueInstant="2005-12-14T10:00:23Z" >
        <saml:Conditions
            NotBefore="2005-12-14T10:00:30Z"
            NotAfter="2005-12-14T10:15:00Z" />
        </saml:Conditions>
        <saml:AuthenticationStatement
            AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
            AuthenticationInstant="2005-12-14T10:00:20Z">
            <saml:Subject>
                <saml:NameIdentifier NameQualifier="RelyingParty.com">
                    john.smith
                </saml:NameIdentifier>
            </saml:Subject>
        </saml:AuthenticationStatement>
    </saml:Assertion>
</samlp:Response>
```
SAML Requests and Responses

The authority that issues assertions is known as the **issuing authority** or **identity provider**. An issuing authority can be a third-party service provider or an individual business that is serving as an issuing authority within a private federation of businesses. SAML-compliant applications and services, which trust the issuing authority or identity provider and make use of its services, are called **relying parties** or **service providers**.

**SAML Request and Response Cycle**

In a typical SAML cycle, the relying party (or service provider), which needs to authenticate a specific client request, sends a SAML request to its issuing authority or identity provider. The identity provider responds with a SAML assertion, which supplies the relying party or service provider with the requested security information. This cycle is illustrated in **Figure 1–1**.

![Figure 1–1 SAML Request-Response Cycle](image_url)

For example, when a user signs into a SAML-compliant service of a relying party or identity provider, the service sends a "request for authentication assertion" to the issuing authority (identity provider). The issuing authority returns an "authentication assertion" reference stating that the user was authenticated by a particular method at a specific time. The service can then pass this assertion reference to other relying party/identity provider sites to validate the user’s credentials. When the user accesses another SAML-compliant site that requires authentication, that site uses the reference to request the "authentication assertion" from the issuing authority or identity provider, which states that the user has already been authenticated.
At the issuing authority, an assertion layer handles request and response messages using the SAML protocol, which can bind to various communication and transport protocols (HTTP, SOAP, and so on). Note that while the client always consumes assertions, the issuing authority or identity provider can act as producer and consumer since it can both create and validate assertions.

**SAML Protocol Bindings and Profiles**

SAML defines a protocol for requesting and obtaining assertions (SAML). Bindings define the standard way that SAML request and response messages are transported across the issuing authorities (identity providers) and relying parties (identity providers) by providing mappings between SAML messages and standard communication protocols. For example, the defined transport mechanism for SAML requests and responses is Simple Object Access Protocol (SOAP) over HTTP. This enables the exchange of SAML information across several Web services in a standard manner.

A profile describes how SAML assertions are embedded into and extracted out of standard frameworks and protocol. Web browser profiles for single sign-on and SOAP profiles for securing SOAP payloads are some of the profiles defined.

**SAML and XML Security**

In addition, SAML was designed to integrate with XML Signature and XML Encryption, standards from the World Wide Web Consortium for embedding encrypted data or digital signatures within an XML document. This support for XML signatures allows SAML to handle not only authentication, but also message integrity and nonrepudiation of the sender. See Chapter 8 for more information about Oracle XML Security.

---

**Federation**

As global businesses strive for ever-closer relationships with suppliers and customers, they face challenges in creating more intimate, yet highly secure trading relationships.

Parties conducting a business transaction must be certain of the identity of the person or agent with whom they are dealing; they must also be assured that the other has the authority to act on behalf of the business with whom the transaction is being conducted.

Historically, in the course of doing business with partners, companies have resorted to acquiring names, responsibilities, and other pertinent information about all entities who might act on behalf of the partner company. With changing roles and responsibilities, and particularly in large enterprises, this can create significant logistical problems as the data quickly becomes very costly to maintain and manage.

Besides complexity, other challenges include cost control, enabling secure access to resources for employees and customers, and regulatory compliance, among others.

These requirements are driving the move toward Federated Identity Management, in which a federated relationship is established between parties when one party presents its credentials to the other using a process known as “assertions.” The receiving party recognizes credentials issued by a trusted trading partner and in an agreed-upon format.

Key federation terminology includes:

- **Principal** - the key actor in a federated environment, being an entity that performs an authorized business task
Overview of Oracle Security Developer Tools

This section provides an introduction to Oracle Security Developer Tools, which are pure Java tools that enable you to complete a wide range of security projects and tasks.

Figure 1–2  The Oracle Security Developer Tools

Figure 1–2 shows the components of the Oracle Security Developer Tools, arranged in layers with the fundamental building-blocks at the bottom layer; each additional layer utilizes and builds upon the previous layers to provide tools for specific security applications.

Oracle Crypto and Oracle Security Engine are the basic cryptographic tools of the set. The next layer consists of Oracle CMS for message syntax, Oracle XML Security for signature encryption, and Oracle PKI SDK, which is a suite of PKI tools consisting of Oracle PKI SDK LDAP, Oracle PKI SDK TSP, Oracle PKI SDK OCSP, and Oracle PKI SDK CMP. Oracle S/MIME exploits Oracle CMS to provide a toolset for secure e-mail. The next layer contains Oracle SAML and Oracle Liberty SDK, which provides structured assertion markup and federated identity management capabilities. Finally, Oracle Web Services Security provides web services security.

1-10  Oracle Security Developer Tools Reference
A description of each tool follows:

- **Oracle Crypto**
- **Oracle Security Engine**
- **Oracle CMS**
- **Oracle S/MIME**
- **Oracle PKI SDK**
- **Oracle JCE Provider**
- **Oracle XML Security**
- **Oracle SAML**
- **Oracle Web Services Security**
- **Oracle Liberty SDK**
- **Oracle XKMS**

### Oracle Crypto

The Oracle Crypto toolkit provides the following features:

- Public key cryptography algorithms such as **RSA**
- Digital signature algorithms such as Digital Signature Algorithm (DSA), **RSA**, and **Elliptic Curve Cryptography (ECC)**
- Key exchange algorithms such as **Diffie-Hellman** and **Elliptic Curve Cryptography (ECC)**
- Symmetric cryptography algorithms such as **Blowfish**, **AES**, **DES**, **3DES**, **RC2**, and **RC4**
- Message digest algorithms such as **MD2**, **MD4**, **MD5**, **SHA-1**, **SHA-256**, **SHA-384**, and **SHA-512**
- **MAC** algorithms such as **HMAC-MD5** and **HMAC-SHA-1**
- Methods for building and parsing **ASN.1** objects

### Oracle Security Engine

The Oracle Security Engine toolkit provides the following features:

- **X.509** Version 3 Certificates, as defined in RFC 3280
- Full **PKCS#12** support
- **PKCS#10** support for certificate requests
- **CRLs** as defined in RFC 3280
- Implementation of **Signed Public Key And Challenge (SPKAC)**
- Support for **X.500** Relative Distinguished Name
- **PKCS#7** support for wrapping X.509 certificates and CRLs
- Implementation of standard X.509 certificates and CRL extensions
Oracle CMS

Oracle CMS provides an extensive set of tools for reading and writing CMS objects, and supporting tools for developing secure message envelopes.

Oracle CMS implements the IETF Cryptographic Message Syntax specified in RFC-2630. Oracle CMS implements all the RFC-2630 content types.

Oracle S/MIME

Oracle S/MIME provides the following Secure/Multipurpose Internet Mail Extension (S/MIME) features:

- Full support for X.509 Version 3 certificates with extensions, including certificate parsing and verification
- Support for X.509 certificate chains in PKCS#7 and PKCS#12 formats
- Private key encryption using PKCS#5, PKCS#8, and PKCS#12
- An integrated ASN.1 library for input and output of data in ASN.1 DER/BER format

Oracle PKI SDK

Oracle PKI SDK contains a set of tools for working with digital certificates, including access to LDAP directories, date stamping of digital messages, certificate validation, and certificate management. It includes the following toolkits:

- Oracle PKI SDK LDAP
- Oracle PKI SDK TSP
- Oracle PKI SDK OCSP
- Oracle PKI SDK CMP

Oracle PKI SDK LDAP

Oracle PKI SDK LDAP provides facilities for accessing a digital certificate within an LDAP directory. Some of the tasks you can perform using the Oracle PKI SDK LDAP are:

- Validating a user’s certificate in an LDAP directory
- Adding a certificate to an LDAP directory
- Retrieving a certificate from an LDAP directory
- Deleting a certificate from an LDAP directory

Oracle PKI SDK TSP

The Oracle PKI SDK TSP provides the following features and functionality:

- Oracle PKI SDK TSP conforms to RFC 3161 and is compatible with other products that conform to this time stamp protocol (TSP) specification.
- Oracle PKI SDK TSP provides an example implementation of a TSA server to use for testing TSP request messages, or as a basis for developing your own time stamping service.

Oracle PKI SDK OCSP

The Oracle PKI SDK OCSP provides the following features and functionality:
The Oracle PKI SDK OCSP conforms to RFC 2560 and is compatible with other products that conform to this specification, such as Valicert's Validation Authority.

The Oracle PKI SDK OCSP API provides classes and methods for constructing OCSP request messages that can be sent through HTTP to any RFC 2560 compliant validation authority.

The Oracle PKI SDK OCSP API provides classes and methods for constructing responses to OCSP request messages, and an OCSP server implementation that you can use as a basis for developing your own OCSP server to check the validity of certificates you have issued.

Oracle PKI SDK CMP
The set of functions supported by certificate management protocol (CMP) messages are:

- Registration of an entity, which takes place prior to issuing a certificate
- Initialization, such as the generation of a key pair
- Certification (issuing certificates)
- Key pair recovery for reissuing lost keys
- Key pair updates when a certificate expires and a new key pair and certificate needs to be generated
- Revocation requests to the CA to include a certificate in a CRL
- Cross-certification between two CAs

The Oracle PKI SDK CMP conforms to RFC 2510 and is compatible with other products that conform to this certificate management protocol (CMP) specification. In addition, it conforms to RFC 2511 and is compatible with other products that conform to this certificate request message format (CRMF) specification.

Oracle JCE Provider
Java Cryptography Extension (JCE) from Sun Microsystems is a framework for implementing encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms.

The Oracle JCE Provider package supplies a concrete implementation of a subset of the cryptographic services defined in JCE.

Oracle XML Security
XML Security refers to common data security requirements of XML documents, such as confidentiality, integrity, message authentication, and non-repudiation.

Oracle XML Security fulfills these needs by providing the following features:

- Support for the W3C XML Signature standard
- Support for the XML Encryption proposed standard
- Support for the Decryption Transform proposed standard
- Support for the XML Canonicalization standard
- Support for the Exclusive XML Canonicalization standard
- Compatibility with a wide range of JAXP 1.1 compliant XML parsers and XSLT engines
Oracle SAML

The Oracle SAML API provides tools and documentation to assist developers of SAML-compliant Java security services. You can integrate Oracle SAML into existing Java solutions, including applets, applications, EJBs, servlets, and JSPs.

Oracle SAML provides the following features:

- Support for the SAML 1.0/1.1 and 2.0 specifications
- Support for SAML-based single sign-on (SSO), Attribute, Metadata, Enhanced Client Proxy, and federated identity profiles

Oracle Web Services Security

Oracle Web Services Security provides an authentication and authorization framework based on OASIS specifications. Oracle Web Services Security provides the following features:

- Support for the SOAP Message Security standard
- Support for the Username Token Profile standard
- Support for the X.509 Certificate Token Profile standard
- Support for the SAML Assertion Token proposed standard (Draft 15)

Oracle Liberty SDK

Oracle Liberty SDK allows Java developers to design and develop single sign-on (SSO) and federated identity solutions based on the Liberty Alliance specifications. Oracle Liberty SDK, available in versions 1.1 and 1.2, aims to unify, simplify, and extend all aspects of development and integration of systems conforming to the Liberty Alliance 1.1 and 1.2 specifications.

Oracle Liberty SDK provides the following features:

- Support for the Liberty Alliance Project version 1.1 and 1.2 specifications
- Support for Liberty-based Single Sign-on and Federated Identity

Note: For additional information about the standards and specifications mentioned in this chapter, see Appendix A, "References".

Oracle XKMS

Oracle XKMS (XML Key Management Specification) provides a convenient way to handle public key infrastructures by allowing developers to write XML transactions for digital signature processing. Oracle XKMS implements the W3C XKMS standard and avoids some of the cost and complexity involved with public key infrastructures.
This chapter provides information about using the Oracle Crypto Software Development Kit (SDK). Oracle Crypto allows Java developers to develop applications that ensure data security and integrity.

This chapter contains the following topics:

- Oracle Crypto Features and Benefits
- Setting Up Your Oracle Crypto Environment
- Core Classes and Interfaces
- The Oracle Crypto Java API Reference
- Example Programs

**Oracle Crypto Features and Benefits**

Oracle Crypto provides the following features:

- Public key cryptography algorithms such as **RSA**
- Digital signature algorithms such as **DSA, RSA**, and **Elliptic Curve Cryptography (ECC)**
- Key exchange algorithms such as **Diffie-Hellman** and **Elliptic Curve Cryptography (ECC)**
- Symmetric cryptography algorithms such as **Blowfish, AES, DES, 3DES, RC2, and RC4**
- Message digest algorithms such as **MD2, MD4, MD5, SHA-1, SHA-256, SHA-384, and SHA-512**
- **MAC** algorithms such as **HMAC-MD5** and **HMAC-SHA-1**
- Methods for building and parsing **ASN.1** objects

**Oracle Crypto Packages**

Oracle Crypto contains the following packages:

- **oracle.security.crypto.core** - Basic cryptographic primitives
- **oracle.security.crypto.core.math** - Utility classes for handling mathematical functions
- **oracle.security.crypto.util** - Various utility classes
Setting Up Your Oracle Crypto Environment

This section explains how to set up your environment to use Oracle Crypto. It contains the following topics:

- System Requirements for Oracle Crypto
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle Crypto

In order to use the Oracle Crypto SDK, your system must have the Java Development Kit (JDK) version 1.2.2 or higher.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to the required jar and class files. Make sure that the osdt_core.jar file is included in your CLASSPATH.

Setting the CLASSPATH on Windows

To set your CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all of the required jar and class files to the CLASSPATH.
   
   For example, your CLASSPATH might look like this:
   
   C:\ORACLE_HOME\jlib\osdt_core.jar

6. Click OK.

Setting the CLASSPATH on UNIX

On UNIX, set your CLASSPATH environment variable to include the full path and file name of all of the required jar and class files. For example:

setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar

Core Classes and Interfaces

This section provides information and code samples for using the core classes and interfaces of Oracle Crypto. The core classes and interfaces are divided into the following categories:

- Keys
- Key Generation
Core Classes and Interfaces

- Ciphers
- Signatures
- Message Digests
- Key Agreement
- Pseudo-Random Number Generators

Keys

Oracle Crypto provides the following classes and interfaces for working with keys:

- The oracle.security.crypto.core.Key Interface
- The oracle.security.crypto.core.PrivateKey Interface
- The oracle.security.crypto.core.PublicKey Interface
- The oracle.security.crypto.core.SymmetricKey Class

The oracle.security.crypto.core.Key Interface

This interface represents a key which may be used for encryption or decryption, for generating or verifying a digital signature, or for generating or verifying a MAC. A key may be a private key, a public key, or a symmetric key.

The oracle.security.crypto.core.PrivateKey Interface

This interface represents a private key which may be an RSAPrivateKey, a DSAPrivateKey, a DHPrivateKey, an ECPPrivateKey or a PrivateKeyPKCS8 instance that holds an encrypted private key.

The oracle.security.crypto.core.PublicKey Interface

This interface represents a public key which may be a RSAPublicKey, a DSAPublicKey, a DHPublicKey or a ECPublicKey instance.

The oracle.security.crypto.core.SymmetricKey Class

This class represents a symmetric key which may be used for encryption, decryption or for MAC operations.

Key Generation

Oracle Crypto provides the following classes for key generation:

- The oracle.security.crypto.core.KeyPairGenerator Class
- The oracle.security.crypto.core.SymmetricKeyGenerator Class

The oracle.security.crypto.core.KeyPairGenerator Class

This abstract class is used to generate key pairs such as RSA, DSA, Diffie-Hellman or ECDSA key pairs.

To get a new key pair generator, create a new instance of KeyPairGenerator by calling the static getInstance() method with an AlgorithmIdentifier object as a parameter. Example 2–1 shows how to create a new KeyPairGenerator instance:

**Example 2–1 Code Example for Creating a New KeyPairGenerator Instance**

```java
KeyPairGenerator kpg = KeyPairGenerator.getInstance(AlgID.rsaEncryption);
```
This creates a KeyPairGenerator object from one of the concrete classes: RSAKeyPairGenerator, DSAKeyPairGenerator, DHKeyPairGenerator, or ECKeyPairGenerator.

Initialize the key pair generator by using one of the initialize() methods. Generate the key pair with the generateKeyPair() method. Example 2–2 shows how to initialize the key pair generator and then generate a key pair:

**Example 2–2  Code Example for Initializing and Generating a Key Pair**

```java
kpg.initialize(1024, RandomBitsSource.getDefault());
KeyPair kp = kpg.generateKeyPair();
PrivateKey privKey = kp.getPrivate();
PublicKey pubKey = kp.getPublic();
```

Save the keys using the output() method, or in the case of the private key, encrypt it and save it using the PrivateKeyPKCS8 class. Example 2–3 shows how to save a key pair.

**Example 2–3  Code Example for Saving a Key Pair**

```java
FileOutputStream pubKeyFos = new FileOutputStream("my-pub-key.der");
pubKey.output(pubKeyFos);
pubKeyFos.close();

PrivateKeyPKCS8 privKeyPKCS8 =
    new PrivateKeyPKCS8(privKey, "myPassword");
FileOutputStream privKeyFos =
    new FileOutputStream("my-encrypted-priv-key.der");
privKeyPKCS8.output(privKeyFos);
privKeyFos.close();
```

The oracle.security.crypto.core.SymmetricKeyGenerator Class

This class generates symmetric key pairs such as Blowfish, DES, 3DES, RC4, RC2, AES, and HMAC keys.

To get a new symmetric key generator, create a new instance of SymmetricKeyGenerator by calling the static getInstance() method with an AlgorithmIdentifier object as a parameter. Example 2–4 shows how to create a new SymmetricKeyGenerator instance:

**Example 2–4  Code Example for Creating a New SymmetricKeyGenerator Instance**

```java
SymmetricKeyGenerator skg = SymmetricKeyGenerator.getInstance(AlgID.desCBC);
```

Generate the key pair with the generateKey() method. You can then save the key by using the getEncoded() method. Example 2–5 shows how to generate and save a symmetric key pair.

**Example 2–5  Code Example for Generating and Saving Symmetric Keys**

```java
SymmetricKey sk = skg.generateKey();

FileOutputStream symKeyFos =
    new FileOutputStream("my-sym-key.der");
symKeyFos.write(sk.getEncoded());
```
Ciphers

The Oracle Crypto Cipher classes and interfaces are divided into the following categories:

- Symmetric Ciphers
- The RSA Cipher
- Password Based Encryption

Symmetric Ciphers

The symmetric ciphers are made up of two categories: the block ciphers (such as Blowfish, DES, 3DES, RC2, and AES) and the stream ciphers (such as RC4).

A symmetric cipher can be used for four types of operations:

- Encryption of raw data. Use one of the `encrypt()` methods by passing data to be encrypted.
- Decryption of encrypted data. Use one of the `decrypt()` methods by passing encrypted data to be decrypted.
- Wrapping of private or symmetric keys. Use one of the `wrapKey()` methods by passing the private or symmetric key to be encrypted.
- Unwrapping of private or symmetric encrypted keys. Use either the `unwrapPrivateKey()` or the `unwrapSymmetricKey()` method by passing the encrypted private or symmetric key to be decrypted.

The concrete block cipher classes extend the abstract `oracle.security.crypto.core.BlockCipher` class, which extends the `oracle.security.crypto.core.Cipher` class. The stream cipher classes directly extend the `oracle.security.crypto.core.Cipher` class.

To create a new instance of Cipher, call the static `getInstance()` method with an `AlgorithmIdentifier` and a `Key` object as parameters.

Example 2–6 shows how to create a new Cipher instance. First an RC4 object is created and initialized with the specified key. Second a block cipher DES object is created and initialized with the specified key and padding. This creates a cipher and initializes it with the passed parameters. To re-initialize an existing cipher, call one of the `initialize()` methods.

Example 2–6 Code Example for Creating a Cipher Instance

```java
Cipher rc4 = Cipher.getInstance(AlgID.rc4, rc4SymKey);

Cipher desCipher = Cipher.getInstance(AlgID.desCBC, desSymKey, Padding.PKCS5);
```

When using CBC ciphers, the `AlgorithmIdentifier` object may hold cryptographic parameters such as the initialization vector (IV) or the effective key length for RC2 ciphers. To specify these parameters when creating or initializing block ciphers, build a `CBCAlgorithmIdentifier` object or `RC2AlgorithmIdentifier` object with the cryptographic parameters. Example 2–7 shows how to create and initialize a CBC cipher and a RC2 cipher.
Example 2–7  Code Example for Creating and Initializing CBC Ciphers

CBCAlgorithmIdentifier cbcAlgID =
    new CBCAlgorithmIdentifier(AlgID.desCBC, iv);
desCipher.initialize(cbcAlgID, desSymKey, Padding.PKCS5);
RC2AlgorithmIdentifier rc2AlgID =
    new RC2AlgorithmIdentifier(iv, 56);
BlockCipher rc2Cipher =
    (BlockCipher)Cipher.getInstance(rc2AlgID, rc2SymKey, Padding.PKCS5);

The RSA Cipher

The RSA cipher is an implementation of PKCS#1 v2.0 that supports the RSAES-OAEP and RSAES-PKCS1-v1_5 encryption schemes. According to the specification, RSAES-OAEP is recommended for new applications, and RSAES-PKCS1-v1_5 is included only for compatibility with existing applications and protocols.

The encryption schemes are used to combine RSA encryption and decryption primitives with an encoding method. Encryption and decryption can only be done through the methods encrypt(byte[]) and decrypt(byte[]).

You can use an RSA cipher for four types of operations:

- Encryption of raw data. Use one of the encrypt() methods by passing data to be encrypted.
- Decryption of encrypted data. Use one of the decrypt() methods by passing encrypted data to be decrypted.
- Wrapping of keys. Use the wrapKey() method by passing the key to be encrypted.
- Unwrapping of encrypted keys. Use the unwrapSymmetricKey() method by passing the encrypted key to be decrypted.

To create a new instance of Cipher, call the static getInstance() method with AlgorithmIdentifier and Key objects as parameters. Example 2–8 demonstrates how to create an RSApkcs1 object and initialize it with the specified key. The cipher can then be used to encrypt or decrypt data.

Example 2–8  Code Example for Creating and Initializing an RSA Cipher

Cipher rsaEnc = Cipher.getInstance(AlgID.rsaEncryption, pubKey);
byte[] encryptedData = rsaEnc.encrypt(data);
Cipher rsaDec = Cipher.getInstance(AlgID..rsaEncryption, privKey);
byte[] decryptedData = rsaDec.decrypt(encryptedData);

When using RSA ciphers, the AlgorithmIdentifier object may hold cryptographic parameters such as the mask generation function for RSAES-OAEP. To specify these parameters when creating or initializing RSA ciphers, build an OAEPAlgorithmIdentifier, or use the default one located in the oracle.security.crypto.core.AlgID interface.

Password Based Encryption

The abstract oracle.security.crypto.core.PBE class provides methods for Password Based Encryption (PBE) operations. The concrete classes extending the PBE are the PKCSSPBE and PKCS12PBE classes.

You can use a PBE object for four types of operations:

- Encryption of raw data. For example:
Core Classes and Interfaces

Example 2–9  Code Example for Creating a PBE Object

```java
PBEAlgorithmIdentifier pbeAlgID =
    new PBEAlgorithmIdentifier(PBEAlgorithmIdentifier.pbeWithMD5AndDES_CBC, salt, 1024);
pbeEnc.initialize(pbeAlgID);
PBE pbeDec = PBE.getInstance(pbeAlgID);
```

Signatures

The oracle.security.crypto.core.Signature abstract class provides methods to sign and verify signatures. The concrete classes extending the Signature class are the RSAMDSignature, DSA and the ECDSA classes.

The algorithms available for signature operations are:

- For RSA: AlgID.md2WithRSAEncryption, AlgID.md5WithRSAEncryption and AlgID.sha_1WithRSAEncryption
- For DSA: AlgID.dsaWithSHA1
- For ECDSA: AlgID.ecdsaWithSHA1

To create a new instance of Signature, call the static getInstance() method with an AlgorithmIdentifier and a PrivateKey or PublicKey objects as parameters. Example 2–10 shows how to create a new Signature object and initialize it with the specified algorithm.

Example 2–10  Code Example for Creating a New Signature Object

```java
Signature rsaSign = Signature.getInstance(AlgID.md5WithRSAEncryption);
Signature rsaVerif = Signature.getInstance(AlgID.md5WithRSAEncryption);
```
Example 2–11 shows how to set the keys for the Signature objects and set the document to be signed or verified.

**Example 2–11  Code Example for Setting Signature Keys and Documents**

```java
rsaSign.setPrivateKey(privKey);
rsaSign.setDocument(data);
rsaVerif.setPublicKey(pubKey);
rsaVerif.setDocument(data);
```

Example 2–12 shows how to compute the signature using the private key or to verify the signature using the public key and the signature bytes.

**Example 2–12  Code Example for Computing or Verifying a Signature**

```java
byte[] sigBytes = rsaSign.sign();
boolean verified = rsaVerif.verify(sigBytes);
```

### Message Digests

Oracle Crypto provides the following message digest classes:

- The `oracle.security.crypto.core.MessageDigest Class`
- The `oracle.security.crypto.core.MAC Class`

#### The `oracle.security.crypto.core.MessageDigest Class`

The `MessageDigest` abstract class provides methods to hash and digest data. The concrete classes extending the `MessageDigest` class are the MD2, MD4, MD5 and the SHA classes.

The available algorithms for message digest operations are: `AlgID.md2`, `AlgID.md4`, `AlgID.md5`, `AlgID.sha_1`, `AlgID.sha_256`, `AlgID.sha_384` and `AlgID.sha_512`.

The basic process for creating a message digest is as follows:

1. Create a new instance of `MessageDigest` by calling the static `getInstance()` method with an `AlgorithmIdentifier` object as a parameter.
2. Add the data to be digested.
3. Compute the hash value.

Example 2–13 shows how to create an MD5 message digest object.

**Example 2–13  Code Example for Creating a Message Digest**

```java
//Create a new MD5 MessageDigest object
MessageDigest md5 = Signature.getInstance(AlgID.md5);

//Add the data to be digested
md5.update(data1);
md5.update(data2);

//Compute the hash value
md5.computeCurrent();
byte[] digestBits = md5.getDigestBits();
```
The oracle.security.crypto.core.MAC Class

The MAC abstract class provides methods to compute and verify a Message Authentication Code (MAC). The concrete class extending the MAC is the HMAC class.

The available algorithms for MAC operations are: AlgID.hmacMD5 and AlgID.hmacSHA.

The basic process for creating a MAC is as follows:
1. Create a new instance of MAC by calling the static getInstance() method with an AlgorithmIdentifier and a SymmetricKey object as a parameter.
2. Add the data to be digested.
3. Compute the MAC value and verify it.

Example 2–14 shows how to create a new HMAC object with the HMAC-SHA1 algorithm.

Example 2–14  Code Example for Creating a MAC

```java
//Create an HMAC object with the HMAC-SHA1 algorithm
MAC hmacSha1Compute = MAC.getInstance(AlgID.hmacSHA, hmacSha1Key);

//Add the data to be digested
hmacSha1Compute.update(data);

//Compute the MAC value and verify
byte[] macValue = hmacSha1Compute.computeMAC();
boolean verified = hmacSha1Verify.verifyMAC(data, macValue);
```

Key Agreement

The oracle.security.crypto.core.KeyAgreement class abstract class provides methods for public key agreement schemes such as Diffie-Hellman. The concrete classes extending the KeyAgreement class are the DHKeyAgreement and the ECDHKeyAgreement classes.

The available algorithms for key agreement operations are: AlgID.dhKeyAgreement and ECDHKeyAgreement (Elliptic Curve Diffie-Hellman key agreement).

The basic process for key agreement is as follows:
1. Create a new instance of KeyAgreement by calling the static getInstance() method with an AlgorithmIdentifier object as a parameter.
2. Set the local private key and the other party’s public key.
3. Compute the shared secret value.

Example 2–15 shows how to perform key agreement.

Example 2–15  Code Example for Key Agreement

```java
//Create a DH key agreement object
KeyAgreement dh = KeyAgreement.getInstance(AlgID.dhKeyAgreement);

//Set the private key and public key
dh.setPrivateKey(privKey);
dh.setPublicKey(otherPubKey);

//Compute the shared secret
byte[] sharedSecret = dh.generateSecret();

Pseudo-Random Number Generators

In cryptography, random numbers are used to generate keys. Cryptographic systems need cryptographically strong (pseudo) random numbers that cannot be guessed by an attacker.

Oracle Crypto provides the following pseudo-random number generator (PRNG) classes:

- The oracle.security.crypto.core.RandomBitsSource class
- The oracle.security.crypto.core.EntropySource class

The oracle.security.crypto.core.RandomBitsSource class

RandomBitsSource is an abstract class representing secure PRNG implementations. Note that, by the very nature of PRNGs, the security of their output depends on the amount and quality of seeding entropy used. Implementing classes should provide guidance as to their proper initialization and use. The concrete classes extending the RandomBitsSource are the MD5RandomBitsSource, SHA1RandomBitsSource, and the DSARandomBitsSource classes.

Create a new instance of RandomBitsSource by calling the static getDefault() method to return the default PRNG:

RandomBitsSource rbs = RandomBitsSource.getDefault();

A RandomBitsSource object can also be created by instantiating one of the subclasses:

RandomBitsSource rbs = new SHA1RandomBitsSource();

By default, a newly created PRNG created from a subclass will be seeded. To seed a generic RandomBitsSource object, use one of the seed methods by using a byte array or an EntropySource object:

rbs.seed(myByteArray);

The object is then ready to generate random data:

rbs.randomBytes(myRandomByteArray);

The oracle.security.crypto.core.EntropySource class

The EntropySource class provides a source of seed material for the PRNGs. The concrete classes extending the EntropySource are the SpinnerEntropySource and SREntropySource classes.

Create a new instance of EntropySource by calling the static getDefault() method to return the default entropy source:

EntropySource es = EntropySource.getDefault();

You can also create an EntropySource object by instantiating one of the subclasses:

EntropySource rbs = new SpinnerEntropySource();

The entropy source is readied for use by using one of the generateByte methods:
es.generateBytes(mySeedingArray);

The Oracle Crypto Java API Reference

The Oracle Crypto Java API reference (Javadoc) is available at:

*Oracle Security Developer Tools Crypto Java API Reference*

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at

The Java Cryptography Extension (JCE) from Sun Microsystems is an optional package to the Java 2 platform. It is a framework for implementing encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms.

The Oracle JCE Provider package supplies a concrete implementation of a subset of the cryptographic services defined in JCE 1.2.1.

This chapter contains these topics:

- Oracle JCE Provider Features and Benefits
- Setting Up Your Oracle JCE Provider Environment
- Example Programs

### Oracle JCE Provider Features and Benefits

Oracle JCE Provider supports a number of cryptographic algorithms in the following application areas:

- **Ciphers** -
  - AES
  - Blowfish
  - DES
  - Triple DES
  - DSA
  - RC2
  - RSA
  - RC4
  - PBE with MD2/MD5/SHA1 and DES/RC2/Triple DES/RC4

- **KeyAgreement**: Diffie-Hellman with two or more parties

- **PKCS5Padding** and **NoPadding** support for:
  - RSA
  - AES
  - DES
  - Triple DES
Oracle JCE Provider Features and Benefits

- RC2
- Blowfish
- PKCS1Padding and OAEPPadding for RSA
- Support for standard key ranges
- KeyFactory:
  - RSA
  - DSA
  - Diffie-Hellman
- SecretKeyFactory:
  - AES
  - Blowfish
  - DES
  - Triple DES
  - DSA
  - RC2
  - RC4
  - HMAC-MD5
  - HMAC-SHA1
  - PBE with MD2/MD5/SHA1 and DES/RC2/Triple DES/RC4
- Support for X.509EncodedKeySpec and PKCS8EncodedKeySpec
- KeyPairGeneration:
  - RSA
  - DSA
  - Diffie-Hellman

**Note:** While the minimum acceptable strength is 512, Oracle JCE Provider supplies a default strength of 1024.

- KeyGeneration:
  - AES
  - Blowfish
  - DES
  - Triple DES
  - RC2
  - RC4
  - HMAC-MD5
  - HMAC-SHA1
- Standard default parameters for DSA (same as those provided by SunJCE)
Setting Up Your Oracle JCE Provider Environment

- Message Digests:
  - MD2
  - MD5
  - SHA-1
  - SHA-256
  - SHA-284
  - SHA-512

- Signatures:
  - SHA1withDSA
  - MD5withRSA
  - SHA1withRSA
  - MD2withRSA

- MAC:
  - HMAC-MD5
  - HMAC-SHA1

- Support for standard ASN1 encodings

- SecureRandom:
  - MD5PRNG
  - SHA1PRNG

- Pseudo-random number generators, using proprietary algorithms based on the Bruce Schneier/Applied Cryptography design pattern

- Support for X509 certificates

- Key Store:
  - PKCS#8
  - PKCS#12

Using the Oracle JCE Provider

For more information about the Java Cryptography Extension and how to use the Oracle JCE Provider, please refer to the Sun JCE documentation at:

http://java.sun.com/products/jce/

Setting Up Your Oracle JCE Provider Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME. This section explains how to set up your environment for Oracle JCE Provider. It contains these topics:

- System Requirements for Oracle JCE Provider
- Installation Requirements
- Setting the CLASSPATH Environment Variable
System Requirements for Oracle JCE Provider

Oracle JCE Provider is compatible with Java Cryptography Extension (JCE) version 1.2.1. In order to use Oracle JCE Provider, you must install JCE 1.2.1 on your system.

The Java Cryptography Extension is available from Sun Microsystems at:
http://java.sun.com/products/jce/

Installation Requirements

Add the following line to your `java.security` file, which is usually located in `$JAVA_HOME/jre/lib/security`:

```
security.provider.1=com.phaos.jce.provider.Phaos
```

When installing the distribution files, the location of the Oracle JCE Provider jar file depends on where the JCE 1.2.1 framework is installed:

**If the JCE 1.2.1 framework is an "installed" extension**

If the JCE 1.2.1 framework is an installed extension, the following files:

- `jce1_2_1.jar`
- `jce_provider_jdk1x.jar`
- `US_export_policy.jar`
- `local_policy.jar`

must appear in the standard location for jar files of an installed extension:

<table>
<thead>
<tr>
<th>File</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>$JAVA_HOME/lib/ext</td>
<td>Win32</td>
</tr>
<tr>
<td>$JAVA_HOME/lib/ext</td>
<td>Solaris</td>
</tr>
</tbody>
</table>

where $JAVA_HOME refers to the directory where the Java software is installed.

**If the JCE 1.2.1 framework is located on the classpath**

If the JCE 1.2.1 framework is not installed as an extension but instead is located on the class path, and a security manager is installed, you need to grant permissions to the JCE 1.2.1 framework and JCE providers when you run applets or applications using JCE.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to the required jar and class files. Make sure that the following files are included in your CLASSPATH:

- `osdt_core3.jar`
- `osdt_jce.jar`

Setting the CLASSPATH on Windows

To set your CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.

3. Click Environment Variables.

4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.

5. Add the full path and file names for all of the required jar and class files to the CLASSPATH.

   For example, your CLASSPATH might look like this:

   C:\ORACLE_HOME\jlib\osdt_core3.jar;
   C:\ORACLE_HOME\jlib\osdt_jce.jar

6. Click OK.

Setting the CLASSPATH on UNIX

On UNIX, set your CLASSPATH environment variable to include the full path and file name of all of the required jar and class files. For example:

   setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core3.jar:
   $ORACLE_HOME/jlib/osdt_jce.jar

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at
This chapter provides information about using the Oracle Security Engine Software Development Kit (SDK) certificate package. Oracle Security Engine is a superset of Oracle Crypto. It contains all of the libraries and tools provided with Oracle Crypto, plus additional packages and utilities for generating digital certificates.

Oracle Crypto allows Java developers to develop applications that ensure data security and integrity. For more information about the Oracle Crypto functionality, see "Oracle Crypto" in Chapter 2.

For an overview of public key infrastructure (PKI), see "Public Key Infrastructure (PKI)" in Chapter 1.

This chapter contains the following topics:

- Oracle Security Engine Features and Benefits
- Setting Up Your Oracle Security Engine Environment
- Core Classes and Interfaces
- The Oracle Security Engine Java API Reference
- Example Programs

**Oracle Security Engine Features and Benefits**

Oracle Security Engine provides the following features:

- X.509 Version 3 Certificates, as defined in RFC 3280
- Full PKCS#12 support
- PKCS#10 support for certificate requests
- certificate revocation list (CRL) functionality as defined in RFC 3280
- Implementation of Signed Public Key And Challenge (SPKAC)
- Support for X.500 Relative Distinguished Names
- PKCS#7 support for wrapping X.509 certificates and CRLs
- Implementation of standard X.509 certificates and CRL extensions

**Oracle Security Engine Packages**

The Oracle Security Engine toolkit contains the following packages:

- oracle.security.crypto.cert - Facilities for handling digital certificates, CRLs, and PKCS#12.
Setting Up Your Oracle Security Engine Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME. This section provides information for setting up your environment for Oracle Security Engine. It contains the following topics:

- System Requirements for Oracle Security Engine
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle Security Engine

In order to use Oracle Security Engine, your system must have the Java Development Kit (JDK) version 1.2.2.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar
- osdt_cert.jar

Setting the CLASSPATH on Windows

To set your CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all of the required jar and class files to the CLASSPATH.
   For example, your CLASSPATH might look like this:
   %CLASSPATH%;C:\ORACLE_HOME\jlib\osdt_core.jar;
   C:\ORACLE_HOME\jlib\osdt_cert.jar;
6. Click OK.

Setting the CLASSPATH on UNIX

To set your CLASSPATH on UNIX, set your CLASSPATH environment variable to include the full path and file name of all of the required jar and class files. For example:

setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
$ORACLE_HOME/jlib/osdt_cert.jar:
Core Classes and Interfaces

This section provides information and code samples for using the certificate facility classes of Oracle Security Engine. Oracle Security Engine also includes all of the classes provided with Oracle Crypto. See Chapter 2, "Oracle Crypto" for an overview of the core Oracle Crypto classes.

The core certificate facility classes are:

- The oracle.security.crypto.cert.X500RDN Class
- The oracle.security.crypto.cert.X500Name Class
- The oracle.security.crypto.cert.CertificateRequest Class
- The oracle.security.crypto.cert.X509 Class

The oracle.security.crypto.cert.X500RDN Class

This class represents an X.500 Relative Distinguished Name (RDN). This is the building block for X.500 names. A RDN consists of a set of attribute-value pairs. Typically, there is a single attribute-value pair in each RDN.

**Example 4–1 Code Example for Creating and Retrieving an X500RDN Object**

```java
// Create the X500RDN object
X500RDN rdn = new X500RDN(PKIX.id_at_commonName, "Joe Smith");

// Retrieve the value
X500Name n = new X500Name;
String name = n.getAttribute(PKIX.id_at_commonName).getValue().getValue();
```

The oracle.security.crypto.cert.X500Name Class

This class represents distinguished names as used in the X.500 series of specifications, defined in X.520. An X500Name object is made of X500RDN objects. An X500Name holds attributes defining an entity such as the common name, country, organization, and so on.

To create an X500Name object, use the standard constructor and then populate the object with attributes. Once created, the object can then be DER-encoded to make it available to other processes:

**Example 4–2 Code Example for Creating an X500Name Object**

```java
X500Name name = new X500Name;
name.addComponent(PKIX.id_at_commonName, "Joe Smith");
name.addComponent(PKIX.id_at_countryName, "USA");
name.addComponent(PKIX.id_at_stateOrProvinceName, "NY");
name.addComponent(PKIX.id_at_localityName, "New York");
name.addComponent(PKIX.id_at_organizationalUnitName, "Oracle");
name.addComponent(PKIX.id_at_organizationalUnitName, "Engineering");
name.addComponent(PKIX.emailAddress, "joe.smith@oracle.com");

// Make object DER-encoded so its available to other processes
byte[] encodedName = Utils.toBytes(name);
X500Name n = new X500Name(new ByteArrayInputStream(encodedName));
String name = n.getAttribute(PKIX.id_at_commonName).getValue().getValue();
String email = n.getAttribute(PKIX.emailAddress).getValue().getValue();
```
The oracle.security.crypto.cert.CertificateRequest Class

This class represents a PKCS#10 certificate request containing information about an entity and a signature of the content of the request. The certificate request is used to convey information and authentication data (the signature) that will be used by a Certificate Authority (CA) to generate a certificate for the corresponding entity.

Creating a new certificate request involves the following high-level steps:

1. Create a new instance of CertificateRequest by using the empty constructor and setting the keys and the subject name, or by using the constructor taking an X500Name and a KeyPair object.
2. Add X.509 extensions to the certificate request.
3. Sign the certificate request and save it to a file.
4. Send the certificate request you created to a Certificate Authority.

**Example 4–3 Code Example for Creating a Certificate Request**

```java
//Create CertificateRequest by setting the keys and subject name
CertificateRequest certReq = new CertificateRequest();
certReq.setPrivateKey(privKey);
certReq.setPublicKey(pubKey);
certReq.setSubject(subjectName);

//OR

// Create CertificateRequest by taking an X500Name and KeyPair object
CertificateRequest certReq = new CertificateRequest(subjectName, keyPair);

// Add X.509 certificate extensions in a extensionRequest attribute
X509ExtensionSet extSet = new X509ExtensionSet();

// Basic Constraints: non-CA, critical
extSet.addExtension(new BasicConstraintsExtension(false, true));

// Key Usage: signature, data encipherment, key agreement
// & non-repudiation flags, critical
extSet.addExtension(new KeyUsageExtension(new int[] {
    KeyUsageExtension.DIGITAL_SIGNATURE,
    KeyUsageExtension.DATA_ENCIPHERMENT,
    KeyUsageExtension.KEY_AGREEMENT,
    KeyUsageExtension.NON_REPUDIATION
}, true));

// Subject Alternative Name: email address, non-critical
if (email.length() > 0)
    extSet.addExtension(new SubjectAltNameExtension(new GeneralName(GeneralName.Type.RFC822_NAME, email), false));

// Subject Key Identifier: key ID bytes, non-critical
extSet.addExtension(new SubjectKeyIDExtension(CryptoUtils.generateKeyID(kp.getPublic())));

req.addAttribute(PKIX.extensionRequest, extSet);

// Sign the certificate request and save to file
req.sign();
req.output(reqOS);
reqOS.close();
```
// The certificate request can then be sent to a CA

**The oracle.security.crypto.cert.X509 Class**

This class represents an X.509 certificate. Oracle Security Engine supports the generation of new certificates as well as the parsing of existing certificates.

**The Oracle Security Engine Java API Reference**

The Oracle Security Engine Java API reference (Javadoc) is available at:

*Oracle Security Developer Tools Security Engine Java API Reference*

**Example Programs**

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at

This chapter describes key features and benefits of Oracle CMS and explains how to set up and use Oracle CMS.

This chapter contains these topics:

- Oracle CMS Features and Benefits
- Setting Up Your Oracle CMS Environment
- Developing Applications with Oracle CMS
- The Oracle CMS Java API Reference
- Example Programs

Oracle CMS Features and Benefits

The Oracle CMS SDK is a pure Java API with an extensive set of tools for reading and writing CMS objects, sample programs, and supporting tools for developing secure message envelopes.

Oracle CMS implements the IETF Cryptographic Message Syntax specified in RFC-2630. A link to this document is available in Appendix A, "References".

Content Types

Oracle CMS supports all the content types specified in RFC-2630, as shown in Table 5-1:

<table>
<thead>
<tr>
<th>Type</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>1.2.840.113549.1.7.1</td>
</tr>
<tr>
<td>signed-data</td>
<td>1.2.840.113549.1.7.2</td>
</tr>
<tr>
<td>enveloped-data</td>
<td>1.2.840.113549.1.7.3</td>
</tr>
<tr>
<td>digested-data</td>
<td>1.2.840.113549.1.7.5</td>
</tr>
<tr>
<td>encrypted-data</td>
<td>1.2.840.113549.1.7.6</td>
</tr>
<tr>
<td>authenticated-data</td>
<td>1.2.840.113549.1.9.16.1.2</td>
</tr>
</tbody>
</table>

Oracle CMS is a full implementation of RFC-2630 with the following exceptions:

- There is no support for Attribute Certificates
- There is no support for Key Agreement RecipientInfo
Oracle CMS supports the following Enhanced Security Services for S/MIME content type specified in RFC-2634:

<table>
<thead>
<tr>
<th>Type</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>receipt</td>
<td>1.2.840.113549.1.9.16.1.2</td>
</tr>
</tbody>
</table>

A link to this document is available in Appendix A, "References".

The following IETF PKIX TimeStamp Protocol content type corresponding to RFC 3161 is supported:

<table>
<thead>
<tr>
<th>Type</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSTInfo</td>
<td>1.2.840.113549.1.9.16.1.4</td>
</tr>
</tbody>
</table>

---

**Note:** Oracle CMS will not process a content type other than the ones specified earlier.

**Differences Between Oracle CMS and PKCS #7 Version 1.5**

Oracle CMS is based on PKCS #7 v 1.5 but differs in the following ways:

- The enveloped-data contains an optional OriginatorInfo
- The SignerIdentifier in the signed-data SignerInfo is a choice of IssuerAndSerialNo or SubjectKeyIdentifier

**Note:** You must keep these differences in mind if you require interoperability with PKCS#7 implementations.

**Setting Up Your Oracle CMS Environment**

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME. This section describes how to set up your environment for Oracle CMS. It contains the following:

- **System Requirements**
- **Setting the CLASSPATH Environment Variable**

**System Requirements**

In order to use Oracle CMS, your system must have the Java Development Kit (JDK) version 1.2.2 or higher.

**Setting the CLASSPATH Environment Variable**

Your CLASSPATH environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your CLASSPATH:

- the osdt_core.jar file
- the osdt_cert.jar file
- the osdt_cms.jar file
Setting the CLASSPATH on Windows

To set the CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all the required jar and class files to the CLASSPATH.
   For example, your CLASSPATH might look like this:
   
   %CLASSPATH%;C:\ORACLE_HOME\jlib\osdt_core.jar;
   C:\ORACLE_HOME\jlib\osdt_cert.jar;
   C:\ORACLE_HOME\jlib\osdt/cms.jar;

6. Click OK.

Setting the CLASSPATH on UNIX

To set your CLASSPATH on UNIX, set your CLASSPATH environment variable to include the full path and file name of all of the required jar and class files. For example:

   setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
   $ORACLE_HOME/jlib/osdt_cert.jar:
   $ORACLE_HOME/jlib/osdt/cms.jar

Developing Applications with Oracle CMS

There are two approaches to reading and writing CMS objects with the oracle.security.crypto.cms package:

- Using the CMSContentInfo classes, which are relatively easy to utilize
- Using one of the following classes:
  - CMSInputStream
  - CMSOutputStream
  - CMSInputConnector
  - CMSOutputConnector

These classes provide the ability to read and write CMS objects in a single pass, eliminating the need to accumulate the input data before writing any output.

The Oracle CMS API enables you to build nested (wrapped) CMS objects with no limit on the number of wrappings.

This section contains these topics:

- CMS Object Types
- Constructing CMS Objects using the CMS***ContentInfo Classes
- Constructing CMS Objects using the CMS***Stream and CMS***Connector Classes
CMS Object Types

Application developers should be aware of some specific CMS object types which are discussed in subsequent sections.

A **detached object** applies to data and receipt content types. For these types, a detached object is one where the protected content is absent.

A **degenerate object** is a certificate-only signed-data object and is defined only for the signed-data content type. It refers to the case where the signed-data object has no signers. It is normally used to store certificates and is associated with file extensions `.p7b` and `.p7c`.

An external signature is defined only for the signed-data content type. It is essentially a detached signed-data object; that is, the signed-data object has one or more signers but the content that was signed is not present in the signed-data object.

Constructing CMS Objects using the CMS***ContentInfo Classes

Table 5–2 lists the classes which make up the CMS***ContentInfo classes.

<table>
<thead>
<tr>
<th>Class</th>
<th>Content Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSDataContentInfo</td>
<td>CMS.id_data</td>
</tr>
<tr>
<td>ESSReceipt</td>
<td>CMS.id_ct_receipt (RFC-2634 receipt)</td>
</tr>
<tr>
<td>CMSDigestedDataContentInfo</td>
<td>CMS.id_digestedData</td>
</tr>
<tr>
<td>CMSUnsignedDataContentInfo</td>
<td>CMS.id_unsignedData</td>
</tr>
<tr>
<td>CMSEncryptedDataContentInfo</td>
<td>CMS.id_encryptedData</td>
</tr>
<tr>
<td>CMSEnvelopedDataContentInfo</td>
<td>CMS.id_envelopedData</td>
</tr>
<tr>
<td>CMSAuthenticateDataContentInfo</td>
<td>CMS.id_ct_authData</td>
</tr>
</tbody>
</table>

You can use these classes to:

- Read and write objects of the appropriate content type
- Construct and process detached objects
- Create nested objects

A detailed discussion of CMS***ContentInfo classes follows.

Abstract Base Class CMSContentInfo

CMSContentInfo is an abstract class representing a fundamental CMS object. Table 5–2 lists the subclasses of CMSContentInfo.

Some of the useful methods of this abstract class are described in Table 5–3.

Table 5–3 Useful Methods of CMSContentInfo

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contentTypeName (contentType)</td>
<td>Returns the content type of the object as a string.</td>
</tr>
<tr>
<td>getContentType()</td>
<td>Returns the content type of the object as an object identifier (OID).</td>
</tr>
</tbody>
</table>
Constructing a CMS Object

Perform the following steps to construct a CMS object:

1. Create the object of the specified content type.
2. Initialize the object.
3. Call the `output()` method to write the object encoding.

If you are reading in an existing `CMSContentInfo`, but you do not know the concrete type in advance, use `inputInstance()`. To create a new object, use one of the constructors of the concrete subclass with which you are working. To read in one of a known concrete type, use the no-args constructor and then invoke the `input()` method.

Reading a CMS Object

Perform the following steps to read an object:

1. Call `CMSContentInfo.inputInstance()` to read in the object.
2. Call `getContentType()` to determine its content type.
3. You can now invoke the content type-specific operations.

The CMSDataContentInfo Class

The class `CMSDataContentInfo` represents an object of type id-data as defined by the constant `CMS.id_data`, and is intended to refer to arbitrary octet strings whose interpretation is left up to the application.

A useful method of this class is:

```
byte[] getData()
```

which returns the data stored in the data object.

To create a CMS data object:

1. Create an instance of `CMSDataContentInfo` using the constructor that takes a byte array, `documentBytes`, that contains the information:

   ```java
   CMSDataContentInfo exdata =
   new CMSDataContentInfo(byte[] documentBytes)
   ```

2. Write the data object to a file, for example `data.p7m`:

   ```java
   exdata.output(new FileOutputStream("data.p7m"));
   ```

---

**Table 5–3 (Cont.) Useful Methods of CMSContentInfo**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>input(java.io.InputStream is)</code></td>
<td>Initializes this object by reading a BER encoding from the specified input stream.</td>
</tr>
<tr>
<td><code>inputInstance(java.io.InputStream is)</code></td>
<td>Creates a new CMSContentInfo object by reading a BER encoding from the specified input stream.</td>
</tr>
<tr>
<td><code>isDegenerate()</code></td>
<td>Indicates if the object is degenerate.</td>
</tr>
<tr>
<td><code>isDetached()</code></td>
<td>Indicates if the object is detached.</td>
</tr>
<tr>
<td><code>output(java.io.OutputStream os)</code></td>
<td>Writes the encoding of the object to the given output stream.</td>
</tr>
</tbody>
</table>
The steps you use when reading a CMS data object depend on whether you know the object's content type.

1. Open a connection to the file using `FileInputStream`.
   
   If you know that the object stored in the file `data.p7m` is of content type `id-data`:
   ```java
   CMSDataContentInfo exdata =
   new CMSDataContentInfo(new FileInputStream("data.p7m"));
   ```
   
   However, if you do not know the content type in advance, check the type prior to reading:
   ```java
   CMSContentInfo cmsdata =
   CMSContentInfo.inputInstance(new FileInputStream("data.p7m"));
   if (cmsdata instanceof CMSDataContentInfo)
   {
      CMSDataContentInfo exdata = (CMSDataContentInfo) cmsdata;
      // ....
   }
   ```

2. To access the information stored in the CMS data object:
   ```java
   byte[] docBytes = exdata.getData();
   ```

The `ESSReceipt` Class

Class `ESSReceipt` represents an object of type `id-ct-receipt` as defined by the constant `CMS.id_ct_receipt`, and refers to an RFC-2634 receipt.

Table 5–4 lists some useful methods of this class.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[] getOriginatorSignatureValue()</td>
<td>Returns the signature value of the message that triggered the generation of this receipt.</td>
</tr>
<tr>
<td>ASN1ObjectID getReceiptContentType()</td>
<td>Returns the content type of the message that triggered the generation of this receipt.</td>
</tr>
<tr>
<td>byte[] getReceiptData()</td>
<td>Returns the encoded receipt.</td>
</tr>
<tr>
<td>byte[] getSignedContentIdentifier()</td>
<td>Returns the signed content identifier of the message that triggered the generation of this receipt.</td>
</tr>
<tr>
<td>void inputContent(InputStream is)</td>
<td>Initialize this object by reading the BER encoding from the specified input stream.</td>
</tr>
</tbody>
</table>

Take the following steps to create a CMS receipt object.

1. Create an instance of `ESSReceipt` using the constructor that takes a content type identifier, a byte array containing the signed content identifier and a byte array containing the originator signature value:
   ```java
   ESSReceipt rcpt =
   new ESSReceipt(contentType, signedContentIdentifier,
   ```
Developing Applications with Oracle CMS

originatorSignatureValue);

2. Write the receipt object to a file, for example data.p7m:

rcpt.output(new FileOutputStream("data.p7m"));

**Note:** When you create an ESSReceipt object, do not leave any input parameters set to null.

To read a receipt object:

1. Open a connection to the file using FileInputStream.

   If you know that the object stored in the file data.p7m is of content type id-ct-receipt:

   ```java
   ESSReceipt rcptdata = new ESSReceipt(new FileInputStream("data.p7m"));
   ```

   Otherwise, if the content type is unknown:

   ```java
   CMSContentInfo cmsdata = CMSContentInfo.inputInstance(new FileInputStream("data.p7m"));
   if (cmsdata instanceof ESSReceipt)
       ESSReceipt rcptdata = (ESSReceipt) cmsdata;
       // ....
   }
   ```

2. Access the information stored in the receipt object:

   ```java
   ASN1ObjectID contentType = getReceiptContentType();
   byte[] sciBytes = rcptdata.getSignedContentIdentifier()
   byte[] osvBytes = rcptdata.getOriginatorSignatureValue();
   ```

**The CMSDigestedDataContentInfo Class**

The class CMSDigestedDataContentInfo represents an object of type id-digestedData as defined by the constant CMS.id_digestedData.

Table 5–5 lists some of the useful methods of this class.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[] getDigest()</td>
<td>Returns the message digest value.</td>
</tr>
<tr>
<td>AlgorithmIdentifier getDigestAlgID()</td>
<td>Returns the message digest algorithm ID.</td>
</tr>
<tr>
<td>CMSContentInfo getEnclosed()</td>
<td>Returns the digested content.</td>
</tr>
<tr>
<td>ASN1ObjectID getEnclosedContentType()</td>
<td>Returns the content type of the digested content.</td>
</tr>
<tr>
<td>ASN1Integer getVersion()</td>
<td>Returns the version number of this object.</td>
</tr>
<tr>
<td>isDetached()</td>
<td>Indicates if this object is detached.</td>
</tr>
<tr>
<td>void setEnclosed(CMSContentInfo content)</td>
<td>Sets the encapsulated content, that is, the object that was originally digested.</td>
</tr>
<tr>
<td>void writeDetached(boolean writeDetached)</td>
<td>Indicates if the object that is being digested should be omitted when creating the CMSDigestedDataContentInfo object.</td>
</tr>
</tbody>
</table>
Constructing a CMS Digested-data Object

Take the following steps to create a CMS digested-data object.

1. Create an instance of CMSDigestedDataContentInfo using the constructor that takes the object to be digested and the digest algorithm identifier. For example, if contentInfo is a CMSDataContentInfo object and MD5 is the digest algorithm:

   ```java
   CMSDigestedDataContentInfo dig =
   new CMSDigestedDataContentInfo(contentInfo, CMS.md5);
   ```

2. Write the CMS digested-data object to a file named `data.p7m`.

   ```java
   dig.output(new FileOutputStream("data.p7m"));
   ```

Reading a CMS Digested-data Object

The steps you need to read a CMS digested-data object depend on whether you know the object's content type.

1. Open a connection to the `data.p7m` file using FileInputStream.
   If you know that the object stored in the file is of content type id-digestedData:

   ```java
   CMSDigestedDataContentInfo digdata =
   new CMSDigestedDataContentInfo(new FileInputStream("data.p7m"));
   ```

   However, if you do not know the content type in advance:

   ```java
   CMSContentInfo cmsdata =
   CMSContentInfo.inputInstance(new FileInputStream("data.p7m"));
   if (cmsdata instanceof CMSDigestedDataContentInfo)
   {
      CMSDigestedDataContentInfo digdata =
      (CMSDigestedDataContentInfo) cmsdata;
      // ..... 
   }
   ```

2. To access the information stored in the CMS digested-data object:

   ```java
   int version = digdata.getVersion().intValue();
   AlgorithmIdentifier digestAlgID = digdata.getDigestAlgID();
   byte[] digestValue = digdata.getDigest();
   CMSContentInfo digContentInfo = digData.getEnclosed()
   if (digData.getEnclosedContentType().equals(CMS.id_data))
      CMSDataContentInfo contentInfo = (CMSDataContentInfo)digContentInfo;
   ```

3. To verify the integrity of the protected data, verify the digest:

   ```java
   digData.verify();
   ```

Detached digested-data Objects

When working with a detached object, the object that is digested is not a part of the resulting CMS digested-data structure. To generate a detached object, call the `writeDetached (true | false)` method. For example:

```java
dig.writeDetached(true);
```
While you can read in a detached CMS digested-data object as shown earlier, the digest verification will fail because the original object that was digested is not present. To resolve this, call the `setEnclosed(CMScontentInfo)` method to set the `digestedContent`:

```java
digdata.setEnclosed(CMScontentInfo object);
```

followed by digest verification:

```java
digdata.verify();
```

**The CMSSignedDataContentInfo Class**

The class `CMSSignedDataContentInfo` represents an object of type `id-signedData` as defined by the constant `CMS.id_signedData`.

Oracle CMS supports a *choice* of `IssuerAndSerialNo` or `SubjectKeyIdentifier` for use as the `SignerIdentifier`. For interoperability with PKCS #7 and S/MIME, however, the `IssuerAndSerialNo` must be used as the `SignerIdentifier`.

Table 5–6 lists some useful methods of this class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void addCertificate(X509 cert)</code></td>
<td>Appends the given certificate to the list of certificates which will be included with this signed data object.</td>
</tr>
<tr>
<td><code>void addCRL(CRL crl)</code></td>
<td>Appends the given CRL to the list of CRLs which will be included with this signed data object.</td>
</tr>
<tr>
<td><code>void addSignature(AttributeSet authenticatedAttributes, PrivateKey signerKey, X509 signerCert, AlgorithmIdentifier digestAlgID, AlgorithmIdentifier digestEncryptionAlgID, AttributeSet unauthenticatedAttributes)</code></td>
<td>Adds a signature using the IssuerAndSerialNumber as the SignerIdentifier, that is, a Version1 CMSSignerInfo.</td>
</tr>
<tr>
<td><code>void addSignature(AttributeSet authenticatedAttributes, PrivateKey signerKey, X509 signerCert, AlgorithmIdentifier digestAlgID, AlgorithmIdentifier digestEncryptionAlgID, AttributeSet unauthenticatedAttributes, boolean useSPKI64)</code></td>
<td>Adds a signature using the SubjectKeyIdentifier as the SignerIdentifier; that is, a Version3 CMSSignerInfo.</td>
</tr>
<tr>
<td><code>void addSignerInfo(X509 signerCert, CMSSignerInfo signerInfo)</code></td>
<td>Adds a CMSSignerInfo to the list of signers.</td>
</tr>
<tr>
<td><code>Vector getCertificates()</code></td>
<td>Returns the list of certificates included with this signed data object.</td>
</tr>
<tr>
<td><code>Vector getCRLs()</code></td>
<td>Returns the list of CRLs included with this signed data object.</td>
</tr>
<tr>
<td><code>CMSContentInfo getEnclosed()</code></td>
<td>Returns the signed document.</td>
</tr>
<tr>
<td><code>ASN1ObjectID getEnclosedContentType()</code></td>
<td>Returns the content type of the document which was signed.</td>
</tr>
<tr>
<td><code>CMSSignerInfo getSignerInfo(X509 signerCert)</code></td>
<td>Returns the CMSSignerInfo corresponding to the certificate.</td>
</tr>
<tr>
<td><code>ASN1Integer getVersion()</code></td>
<td>Returns the version number of this object.</td>
</tr>
</tbody>
</table>
Oracle CMS supports the RSA and DSA signature algorithms.

**Constructing a CMS Signed-data Object**

Follow these steps to create a CMS signed-data object:

1. Create an instance of `CMSSignedDataContentInfo`. For example, to create the `CMSSignedDataContentInfo` object, pass the `contentInfo` object (the data that is to be signed):

   ```java
   CMSSignedDataContentInfo sig =
       new CMSSignedDataContentInfo(contentInfo);
   ```

2. Add signatures:

   ```java
   X509 signerCert = new X509(new FileInputStream("name"));
   PrivateKey signerKey =
       CryptoUtils.inputPrivateKey(new FileInputStream("name"));
   ```

   a. To add a signature using the IssuerAndSerialNo as the SignerIdentifier, MD5 digests and RSA Signature Algorithm:
sig.addSignature(null, signerKey, signerCert, CMS.md5, CMS.rsaEncryption, null);

b. To add a signature using the 64 bit SubjectKeyIdentifier as the SignerIdentifier, SHA-1 digests and DSS Signature Algorithm:
sig.addSignature(null, signerKey, signerCert, CMS.sha_1, CMS.dsaWithSHA, null, true);

c. To add a signature using the 160 bit SubjectKeyIdentifier as the SignerIdentifier, SHA-1 digests and RSA Signature Algorithm:
sig.addSignature(null, signerKey, signerCert, CMS.sha_1, CMS.rsaEncryption, null, false);

3. Add any Certificates and CRLs:
sig.addCertificate (....)
sig.addCRL (....)

4. Write the CMS signed-data object to a file, for example data.p7m:
sig.output(new FileOutputStream("data.p7m"));

Reading a CMS Signed-data Object
The steps you need to read a CMS signed-data object depend on whether you know the object’s content type.

1. Open a connection to the data.p7m file using FileInputStream.
   If you know that the object stored in the file is of content type id-signedData:
   CMSSignedDataContentInfo sigdata =
   new CMSSignedDataContentInfo(new FileInputStream("data.p7m"));

   However, if you do not know the content type in advance:
   CMSContentInfo cmsdata =
   CMSContentInfo.inputInstance(new FileInputStream("data.p7m"));
   if (cmsdata instanceof CMSSignedDataContentInfo)
   {
      CMSSignedDataContentInfo sigdata =
      (CMSSignedDataContentInfo) cmsdata;
      // ......
   }

2. Access the information stored in the CMS signed-data object:
   int version = sigdata.getVersion().intValue();
   CMSContentInfo sigContentInfo = sigData.getEnclosed()
   Vector certs = sigdata.getCertificates();
   Vector crls = sigData.getCRLS();
   Enumeration e = sigData.signers();
   CMSContentInfo sigContentInfo = sigData.getEnclosed();
   if (sigData.getEnclosedContentType().equals(CMS.id_data))
   CMSDataContentInfo contentInfo = (CMSDataContentInfo)sigContentInfo;

3. Verify the signature using the signer’s public key certificate:
sigData.verify(signerCert);

4. To get more information about the signer:
CMSSignerInfo sigInfo = sigdata.getSignerInfo(signerCert);
byte[] signatureValue = sigInfo.getEncryptedDigest();
AlgorithmIdentifier digest = sigInfo.getDigestAlgID();
AlgorithmIdentifier signature = sigInfo.getDigestEncryptionAlgID();
AttributeSet signedAttributes = sigInfo.getAuthenticatedAttributes();
AttributeSet unsignedAttributes = sigInfo.getUnauthenticatedAttributes();

External Signatures (Detached Objects)

For a detached object, the signed object is not part of the resulting CMS signed-data structure. To generate a detached object, call the writeExternalSignature() method:

```
sig.writeExternalSignature(true);
```

While you can read in a detached CMS signed-data object as shown in "Reading a CMS Signed-data Object", the signature verification will fail because the original object that was signed is not present. Call the setEnclosed (...) method to set the signed content:

```
sigdata.setEnclosed(contentInfo);
```

followed by signature verification:

```
sigdata.verify(signerCert);
```

Certificates/CRL-Only Objects

These are essentially CMSSignedDataContentInfo objects with attached certificates, or CRLs, or both, but without any signatures. To generate a Certificate/CRL-only object:

```
CMSSignedDataContentInfo sigdata =
    new CMSSignedDataContentInfo(new CMSDataContentInfo(new byte[0]));
sigData.addCertificate (...);
sigData.addCRL( ...);
sigData.output(...);
```

You can read in a Certificate/CRL-only signed-data object as shown in "Reading a CMS Signed-data Object".

The CMSEncryptedDataContentInfo Class

The class CMSEncryptedDataContentInfo represents an object of type id-encryptedData as defined by the constant CMS.id_encryptedData. Table 5–7 lists some useful methods of this class.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlgorithmIdentifier getContentEncryptionAlgID()</td>
<td>Returns the content encryption algorithm</td>
</tr>
<tr>
<td>CMSContentInfo getEnclosed(SymmetricKey decryptionKey)</td>
<td>Returns the decrypted content</td>
</tr>
<tr>
<td>ASN1ObjectID getEnclosedContentType()</td>
<td>Returns the content type of the encrypted content</td>
</tr>
<tr>
<td>byte[] getEncryptedContent()</td>
<td>Returns the encrypted content</td>
</tr>
</tbody>
</table>
You can use any of the ciphers supported by the Oracle Security Engine to perform the encryption operation, including RC2, DES, Triple-DES, AES, and so on.

### Constructing a CMS Encrypted-data Object

To create an encrypted-data object:

1. Create an instance of `CMSEncryptedDataContentInfo`. For example, if `contentInfo` is a `CMSDataContentInfo` object and the cipher is Triple-DES in CBC mode:
   ```java
   SymmetricKey contentEncryptionKey = 
   SymmetricKeyGenerator.getInstance(CMS.des_ede3_cbc).generateKey();
   CMSEncryptedDataContentInfo enc = 
   new CMSEncryptedDataContentInfo(contentInfo, contentEncryptionKey, 
   CMS.des_ede3_cbc);
   
   2. Write the encrypted-data object to a file, say `data.p7m`:
   ```java
   enc.output(new FileOutputStream("data.p7m"));
   ```

### Reading a CMS Encrypted-data Object

The steps you need to read an encrypted-data object depend on whether you know the object's content type.

1. Open a connection to the `data.p7m` file using `FileInputStream`.

   If you know that the object stored in the file `data.p7m` is of content type `id-encryptedData`:
   ```java
   CMSEncryptedDataContentInfo encdata = 
   new CMSEncryptedDataContentInfo(new FileInputStream("data.p7m"));
   ```

   However, if you do not know the content type in advance:
   ```java
   CMSContentInfo cmsdata = 
   CMSContentInfo.inputInstance(new FileInputStream("data.p7m"));
   if (cmsdata instanceof CMSEncryptedDataContentInfo) 
   { 
     CMSEncryptedDataContentInfo encdata = 
     (CMSEncryptedDataContentInfo) cmsdata;
   ```

---

**Table 5-7 (Cont.) Useful Methods of CMSEncryptedDataContentInfo**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeSet getUnprotectedAttributes()</td>
<td>Returns the set of unprotected attributes</td>
</tr>
<tr>
<td>ASN1Integer getVersion()</td>
<td>Returns the version number</td>
</tr>
<tr>
<td>boolean isDetached()</td>
<td>Indicates if this is a detached CMS object</td>
</tr>
<tr>
<td>void setEncryptedContent(byte[] encryptedContent)</td>
<td>Sets the encrypted content</td>
</tr>
<tr>
<td>void setUnprotectedAttributes (oracle.security.crypto.cert.AttributeSet unprotectedAttributes)</td>
<td>Sets the unprotected attributes</td>
</tr>
<tr>
<td>void writeDetached (boolean writeDetachedObject)</td>
<td>Indicates if the encryptedContent will be a part of the EncryptedContentInfo structure in this object's outputInfo encoding</td>
</tr>
</tbody>
</table>

---
2. To access the information stored in the CMS encrypted-data object:

```java
int version = encdata.getVersion().intValue();
AlgorithmIdentifier encAlgID = encdata.getContentEncryptionAlgID();
byte[] encValue = encdata.getEncryptedContent();
CMSContentInfo encContentInfo =
    encdata.getEnclosed(ContentEncryptionKey);  //Decrypt the Content
if (encData.getEnclosedContentType().equals(CMS.id_data))
    CMSDataContentInfo contentInfo = (CMSDataContentInfo)encContentInfo;
```

### Detached encrypted-data CMS Objects

If it is a detached object, the encrypted object is not a part of the resulting CMS encrypted-data structure. To generate a detached object, call the `writeDetached` method:

```java
enc.writeDetached(true);
```

While you can read in a detached CMS encrypted-data object as shown in "Reading a CMS Encrypted-data Object", the content decryption will fail because the original object that was encrypted is not present. Call the `setEncryptedContent` method to set the `encryptedContent`:

```java
encData.setEncryptedContent(enc.getEncryptedContent());
```

followed by content decryption:

```java
encdata.getEnclosed(ContentEncryptionKey);
```

### The CMSEnvelopedDataContentInfo Class

The class `CMSEnvelopedDataContentInfo` represents an object of type `id-envelopedData` as defined by the constant `CMS.id_envelopedData`.

Table 5–8 lists some useful methods of this class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void addRecipient(AlgorithmIdentifier keyEncryptionAlgID, SymmetricKey keyEncryptionKey, byte[] keyIdentifier, Date keyDate, ASN1Sequence otherKeyAttribute)</code></td>
<td>Adds a recipient using the key encryption (wrap) key exchange mechanism.</td>
</tr>
<tr>
<td><code>void addRecipient(CMSRecipientInfoSpec ris) </code></td>
<td>Adds a recipient using the key exchange mechanism specification</td>
</tr>
<tr>
<td><code>void addRecipient(X509 recipientCert, AlgorithmIdentifier keyEncryptionAlgID)</code></td>
<td>Adds a recipient using the key transport (IssuerAndSerialNo) key exchange mechanism</td>
</tr>
<tr>
<td><code>void addRecipient(X509 recipientCert, AlgorithmIdentifier keyEncryptionAlgID, boolean useSPKI64)</code></td>
<td>Adds a recipient using the key transport (SubjectKeyIdentifier) key exchange mechanism</td>
</tr>
<tr>
<td><code>AlgorithmIdentifier getContentEncryptionAlgID()</code></td>
<td>Returns the content encryption algorithm</td>
</tr>
</tbody>
</table>
Constructing a CMS Enveloped-data Object

To create an enveloped-data object:

1. Create an instance of CMSEnvelopedDataContentInfo. For example, if contentInfo is a CMSDataContentInfo object and the cipher is Triple-DES in CBC mode:

   ```java
   CMSEnvelopedDataContentInfo env =
       new CMSEnvelopedDataContentInfo(contentInfo, CMS.des_ede3_cbc);
   ```

2. Add recipients, keeping in mind the recipient’s key management technique.
   - If the recipient uses the key encryption (wrap) key management mechanism:

     ```java
     env.addRecipient(keyEncryptionAlgID, keyEncryptionKey,
                     keyIdentifier, keyDate, otherKeyAttribute);
     ```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSContentInfo.getEnclosed(PrivateKey, X509 recipientCert)</td>
<td>Returns the enclosed content after decryption using Key Transport RecipientInfo</td>
</tr>
<tr>
<td>CMSContentInfo.getEnclosed(SymmetricKey symmetricKey, byte[] keyIdentifier)</td>
<td>Returns the enclosed content after decryption using Key Encryption RecipientInfo</td>
</tr>
<tr>
<td>CMSContentInfo.getEnclosed(SymmetricKey symmetricKey, byte[] keyIdentifier, Date keyDate)</td>
<td>Returns the enclosed content after decryption</td>
</tr>
<tr>
<td>ASN1ObjectID.getEnclosedContentType()</td>
<td>Returns the content type of the encrypted content</td>
</tr>
<tr>
<td>byte[].getEncryptedContent()</td>
<td>Returns the encrypted content which is encrypted</td>
</tr>
<tr>
<td>OriginatorInfo.getOriginatorInfo()</td>
<td>Returns the OriginatorInfo</td>
</tr>
<tr>
<td>AttributeSet.getUnprotectedAttribs()</td>
<td>Returns the unprotected attributes</td>
</tr>
<tr>
<td>ASN1Integer.getVersion()</td>
<td>Returns the version number</td>
</tr>
<tr>
<td>boolean.isDetached()</td>
<td>Indicates if the encrypted content is not present</td>
</tr>
<tr>
<td>Enumeration.recipients()</td>
<td>Returns the list of message recipients</td>
</tr>
<tr>
<td>void.setEnclosed(byte[] encryptedContent)</td>
<td>Sets the Encrypted Content</td>
</tr>
<tr>
<td>void.setOriginatorInfo(OriginatorInfo origInfo)</td>
<td>Sets the OriginatorInfo</td>
</tr>
<tr>
<td>void.setUnprotectedAttribs()</td>
<td>Sets the unprotected attributes</td>
</tr>
<tr>
<td>(oracle.security.crypto.cert.AttributeSet unprotectedAttributes)</td>
<td></td>
</tr>
<tr>
<td>void.setWriteDetached(boolean writeDetached)</td>
<td>Indicates if the encrypted content must be omitted from this object's output encoding</td>
</tr>
</tbody>
</table>

ASN1ObjectID getEnclosedContentType() Returns the content type of the encrypted content

byte[].getEncryptedContent() Returns the encrypted content which is encrypted

OriginatorInfo.getOriginatorInfo() Returns the OriginatorInfo

AttributeSet.getUnprotectedAttribs() Returns the unprotected attributes

ASN1Integer.getVersion() Returns the version number

boolean.isDetached() Indicates if the encrypted content is not present

Enumeration.recipients() Returns the list of message recipients

void.setEnclosed(byte[] encryptedContent) Sets the Encrypted Content

void.setOriginatorInfo(OriginatorInfo origInfo) Sets the OriginatorInfo

void.setUnprotectedAttribs
(oracle.security.crypto.cert.AttributeSet unprotectedAttributes) Sets the unprotected attributes

void.setWriteDetached(boolean writeDetached) Indicates if the encrypted content must be omitted from this object's output encoding
If the recipient key exchange mechanism was specified using a
CMSRecipientInfoSpec object:
   env.addRecipient(ris)

If the recipient uses the key transport (IssuerAndSerialNo recipient
identifier) key management mechanism:
   env.addRecipient(recipientCert, CMS.rsaEncryption);

If the recipient uses the key transport (64-bit SubjectKeyIdentifier
recipient identifier) key management mechanism:
   env.addRecipient(recipientCert, CMS.rsaEncryption, true)

If the recipient uses the key transport (160-bit SubjectKeyIdentifier
recipient identifier) key management mechanism:
   env.addRecipient(recipientCert, CMS.rsaEncryption, false)

3. Set any optional arguments:
   env.setOriginatorInfo(originatorInfo);
   env.setUnprotectedAttribs(unprotectedAttributes);

4. Write the CMS enveloped-data object to a file, say data.p7m:
   enc.output(new FileOutputStream("data.p7m"));

Reading a CMS Enveloped-data Object
The steps you need to read the object depend on whether you know the object’s
content type.

1. Open a connection to the data.p7m file using FileInputStream. If you know
that the object stored in the file is of content type id-envelopedData:
   CMSEnvelopedDataContentInfo envdata =
       new CMSEnvelopedDataContentInfo(new FileInputStream("data.p7m"));

   However, if you do not know the content type in advance:
   CMSContentInfo cmsdata =
       CMSContentInfo.inputInstance(new FileInputStream("data.p7m"));
   if (cmsdata instanceof CMSEnvelopedDataContentInfo)
   {
       CMSEnvelopedDataContentInfo envdata =
           (CMSEnvelopedDataContentInfo) cmsdata;
   // 
   ......
   }

2. To access the information stored in the enveloped-data object:
   int version = envdata.getVersion().intValue();
   AlgorithmIdentifier encAlgID = envdata.getContentEncryptionAlgID();
   ASN1ObjectID contentType = env.getEnclosedContentType();
   byte[] encryptedContent = env.getEncryptedContent();
   OriginatorInfo origInfo = env.getOriginatorInfo();
   AttributeSet unprotected = env.getUnprotectedAttribs();

3. Decrypt the content depending on the recipient information:
CMSContentInfo envContentInfo =
    env.getEnclosed(privateKey, recipientCert);

or

CMSContentInfo envContentInfo =
    env.getEnclosed(symmetricKey, keyIdentifier);

or

CMSContentInfo envContentInfo =
    env.getEnclosed(symmetricKey, keyIdentifier, keyDate)
if (envContentInfo instanceof CMSDataContentInfo)
{
    CMSDataContentInfo contentInfo = (CMSDataContentInfo) envContentInfo;
    // ...
}

Key Transport Key Exchange Mechanism
This mechanism supports the use of either IssuerAndSerialNo or SubjectKeyIdentifier as the recipient identifier.

Key Agreement Key Exchange Mechanism
This mechanism is not currently supported.

Key Encryption (Wrap) Key Exchange Mechanism
Oracle CMS supports CMS3DESWrap and CMSRC2Wrap algorithms. Mixed mode wrapping is not supported; for example, 3DES keys cannot be RC2-wrapped.

Note: Using the OtherKeyAttribute could cause interoperability problems.

Detached Enveloped-data CMS Object
If working with a detached object, note that the enveloped object is not part of the resulting CMS enveloped-data structure. Call the writeDetached (..) method to generate a detached object:

```java
env.writeDetached(true);
```

While you can read in a detached enveloped-data object as shown in "Reading a CMS Enveloped-data Object", the content decryption will fail because the original, enveloped object is not present. Call the setEnclosed (..) method to set the enveloped content:

```java
envdata.setEnclosed(env.getEncryptedContent());
```

followed by content decryption:

```java
envdata.getEnclosed(.............);
```

The CMSAuthenticatedDataContentInfo Class
The class CMSAuthenticatedDataContentInfo represents an object of type id-ct-authData as defined by the constant CMS.id_ct_authData.
Table 5–9 lists some useful methods of this class.

### Table 5–9 Useful Methods of CMSAuthenticatedDataContentInfo

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void addRecipient(AlgorithmIdentifier keyEncryptionAlgID, SymmetricKey keyEncryptionKey, byte[] keyIdentifier, java.util.Date keyDate, ASN1Sequence otherKeyAttribute)</td>
<td>Adds a recipient using the key wrap key exchange mechanism</td>
</tr>
<tr>
<td>void addRecipient(CMSRecipientInfoSpec ris)</td>
<td>Adds a recipient using the specified key exchange mechanism</td>
</tr>
<tr>
<td>void addRecipient(X509 recipientCert, AlgorithmIdentifier keyEncryptionAlgID)</td>
<td>Adds a recipient using the key transport key exchange mechanism using the IssuerAndSerialNo as the recipient identifier</td>
</tr>
<tr>
<td>void addRecipient(X509 recipientCert, AlgorithmIdentifier keyEncryptionAlgID, boolean useSPKI64)</td>
<td>Adds a recipient using the key transport key exchange mechanism using the SubjectKeyIdentifier as the recipient identifier</td>
</tr>
<tr>
<td>AttributeSet getAuthenticatedAttributes()</td>
<td>Returns the Authenticated Attributes</td>
</tr>
<tr>
<td>AlgorithmIdentifier getDigestAlgID()</td>
<td>Returns the digest algorithm</td>
</tr>
<tr>
<td>CMSContentInfo getEnclosed()</td>
<td>Returns the authenticated content</td>
</tr>
<tr>
<td>ASN1ObjectID getEnclosedContentType()</td>
<td>Returns the content type of the enclosed content</td>
</tr>
<tr>
<td>byte[] getMAC()</td>
<td>Returns the message authentication code</td>
</tr>
<tr>
<td>AlgorithmIdentifier getMACAlgID()</td>
<td>Returns the MAC algorithm used for authentication</td>
</tr>
<tr>
<td>OriginatorInfo getOriginatorInfo()</td>
<td>Returns the Originator information</td>
</tr>
<tr>
<td>AttributeSet getUnauthenticatedAttributes()</td>
<td>Returns the Unauthenticated Attributes</td>
</tr>
<tr>
<td>ASN1Integer getVersion()</td>
<td>Returns the version number</td>
</tr>
<tr>
<td>boolean isDetached()</td>
<td>Indicates if this object is detached</td>
</tr>
<tr>
<td>java.utilEnumeration recipients()</td>
<td>Returns the list of message recipients</td>
</tr>
<tr>
<td>void setAuthenticatedAttributes(AttributeSet authenticatedAttributes, AlgorithmIdentifier digestAlgorithm)</td>
<td>Sets the Authenticated attributes</td>
</tr>
<tr>
<td>void setEnclosed(CMSContentInfo content)</td>
<td>Sets the authenticated content</td>
</tr>
<tr>
<td>void setOriginatorInfo(OriginatorInfo originatorInfo)</td>
<td>Sets the OriginatorInfo</td>
</tr>
<tr>
<td>void setUnauthenticatedAttributes(AttributeSet unauthenticatedAttributes)</td>
<td>Sets the unauthenticated attributes</td>
</tr>
</tbody>
</table>
Constructing a CMS Authenticated-data Object

Take the following steps to create an authenticated-data object:

1. Create an instance of CMSAuthenticatedDataContentInfo. For example, if contentInfo is a CMSDataContentInfo object, Triple-DES HMAC key and HMAC with SHA-1 MAC algorithm:

   ```
   SymmetricKey contentEncryptionKey =
   SymmetricKeyGenerator.getInstance(CMS.des_ede3_cbc).generateKey();
   CMSAuthenticatedDataContentInfo auth =
   new CMSAuthenticatedDataContentInfo(contentInfo,
   contentEncryptionKey, CMS.hmac_SHA_1);
   ```

2. Add recipients, keeping in mind the recipient’s key management technique.

   - If the recipient uses the key encryption (wrap) key management mechanism:
     ```
     auth.addRecipient(keyEncryptionAlgID, keyEncryptionKey, keyIdentifier,
     keyDate, otherKeyAttribute);
     ```

   - If the recipient key exchange mechanism was specified using a CMSRecipientInfoSpec object:
     ```
     auth.addRecipient(ris)
     ```

   - If the recipient uses the key transport (IssuerAndSerialNo recipient identifier) key management mechanism:
     ```
     auth.addRecipient(recipientCert, CMS.rsaEncryption);
     ```

   - If the recipient uses the key transport (64-bit SubjectKeyIdentifier recipient identifier) key management mechanism:
     ```
     auth.addRecipient(recipientCert, CMS.rsaEncryption, true)
     ```

   - If the recipient uses the key transport (160-bit SubjectKeyIdentifier recipient identifier) key management mechanism:
     ```
     auth.addRecipient(recipientCert, CMS.rsaEncryption, false)
     ```

3. Set any optional arguments:

   ```
   auth.setAuthenticatedAttributes(authenticatedAttributes, CMS.md5);
   ```

### Table 5–9 (Cont.) Useful Methods of CMSAuthenticatedDataContentInfo

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void verifyMAC(PrivateKey privateKey, X509 recipientCert)</td>
<td>Returns the enclosed content after decryption</td>
</tr>
<tr>
<td>void verifyMAC(SymmetricKey symmetricKey, byte[] keyIdentifier)</td>
<td>Returns the enclosed content after decryption</td>
</tr>
<tr>
<td>void verifyMAC(SymmetricKey symmetricKey, byte[] keyIdentifier, Date keyDate)</td>
<td>Returns the enclosed content after decryption</td>
</tr>
<tr>
<td>void verifyMAC(SymmetricKey symmetricKey, byte[] keyIdentifier, Date keyDate, ASN1Sequence otherKeyAttribute)</td>
<td>Returns the enclosed content after decryption</td>
</tr>
<tr>
<td>void writeDetached(boolean writeDetachedObject)</td>
<td>Indicates if the authenticated content must be omitted from this object’s output encoding</td>
</tr>
</tbody>
</table>
Developing Applications with Oracle CMS

auth.setOriginatorInfo(originatorInfo);
auth.setUnauthenticatedAttributes(unauthenticatedAttributes);

4. Write the CMS authenticated-data object to a file, say data.p7m:
   auth.output(new FileOutputStream("data.p7m");

Reading a CMS Authenticated-data Object

The steps you need to read the object depend on whether you know the object’s content type:

1. Open a connection to the data.p7m file using FileInputStream. If you know that the object stored in the file is of content type id-ct-authData:
   CMSAuthenticatedDataContentInfo authdata =
   new CMSAuthenticatedDataContentInfo(new FileInputStream("data.p7m"));

   However, if you do not know the content type in advance:
   CMSContentInfo cmsdata =
   CMSContentInfo.inputInstance(new FileInputStream("data.p7m"));
   if (cmsdata instanceof CMSAuthenticatedDataContentInfo) {
      CMSAuthenticatedDataContentInfo authdata =
      (CMSAuthenticatedDataContentInfo) cmsdata;
      // ..... }

2. To access the information stored in the CMS authenticated-data object:
   int version = authdata.getVersion().intValue();
   AlgorithmIdentifier macAlgID = authdata.getMACAlgID();
   byte[] macValue = authdata.getMAC();
   CMSContentInfo authContentInfo = authdata.getEnclosed();
   if (authData.getEnclosedContentType().equals(CMS.id_data))
      CMSDataContentInfo contentInfo = (CMSDataContentInfo)authContentInfo;

3. Verify the MAC depending on the recipient information:
   authdata.verifyMAC(recipientPrivateKey, recipientCert);
   or
   authdata.verifyMAC(symmetricKey, keyIdentifier)
   or
   authdata.verifyMAC(symmetricKey, keyIdentifier, keyDate)
   or
   authdata.verifyMAC(symmetricKey, keyIdentifier, keyDate, otherKeyAttribute)

Detached Authenticated-data CMS Objects

While you can read in a detached authenticated-data object as shown earlier, the MAC verification will fail because the original object that was authenticated is not present. To resolve this, call the setEnclosed (...) method to set the authenticated content:
authdata.setEnclosed(contentInfo);
followed by MAC verification using the appropriate key exchange mechanism:
authdata.verifyMAC(...)

**Wrapped (Triple or more) CMSContentInfo Objects**

To wrap a CMSContentInfo object in another CMSContentInfo object, you simply pass an initialized CMSContentInfo object to the enclosing CMSContentInfo object through its constructor. Call the output (..) method of the enclosing outermost CMSContentInfo object to generate the nested object.

**Reading a Nested (Wrapped) CMS Object**

The approach to reading a nested object depends on whether you know the outermost content type in advance.

If you do not know the outermost content type in advance, call the static method:
CMSContentInfo.inputInstance( ... )

If you do know the outermost content type in advance, call the appropriate constructor:
new CMS***DataContentInfo( .... )

Then, recursively call the getEnclosed(..) method to extract the next inner object.

**Constructing CMS Objects using the CMS***Stream and CMS***Connector Classes**

The CMS***DataContentInfo classes provide the same functionality as the CMS***Stream classes. The primary advantage of the CMS***Stream classes over the CMS***DataContentInfo classes is that CMS objects can be created or read in one pass without having to accumulate all the necessary information.

**Table 5–11** lists the content types of the CMS***Stream classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Content Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSDigestedDataInputStream,</td>
<td>CMS.id_digestedData</td>
</tr>
<tr>
<td>CMSDigestedDataOutputStream</td>
<td></td>
</tr>
<tr>
<td>CMSSignedDataInputStream,</td>
<td>CMS.id_signedData</td>
</tr>
<tr>
<td>CMSSignedDataOutputStream</td>
<td></td>
</tr>
<tr>
<td>CMSEncryptedDataInputStream,</td>
<td>CMS.id_encryptedData</td>
</tr>
<tr>
<td>CMSEncryptedDataOutputStream</td>
<td></td>
</tr>
<tr>
<td>CMSEnvelopedDataInputStream,</td>
<td>CMS.id_envelopedData</td>
</tr>
<tr>
<td>CMSEnvelopedDataOutputStream</td>
<td></td>
</tr>
<tr>
<td>CMSAuthenticatedDataInputStream,</td>
<td>CMS.id_ct_authData</td>
</tr>
<tr>
<td>CMSAuthenticatedDataOutputStream</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5–11** lists the content types of the CMS***Connector classes:
Limitations of the CMS***Stream and CMS***Connector Classes

There are some limitations to CMS***Stream and CMS***Connector classes when processing objects:

1. They cannot verify the digest of a detached CMS id-digestedData object.
2. They cannot verify the signature of a detached CMS id-signedData object.
3. They cannot verify the MAC of a detached CMS id-ct-authData object.

**Caution:** Always use the CMS**DataContentInfo classes when processing detached objects.

Difference between CMS***Stream and CMS***Connector Classes

The CMS***OutputStream class is an output stream filter which wraps the data written to it within a CMS (RFC-2630) ContentInfo structure, whose BER encoding is then written to the underlying output stream. The CMS***OutputConnector class is an output stream filter which likewise wraps the data written to it within a CMS (RFC-2630) ContentInfo structure, except that only the values octets of the Content field of the ContentInfo structure (minus the explicit [0] tag) are written to the underlying output stream.

The CMS***InputStream class is an input stream filter which reads in a BER encoding of a CMS (RFC-2630) ContentInfo structure from the underlying output stream. The CMS***InputConnector class is an input stream filter that expects the underlying input stream to be positioned at the start of the value octets of the Content field of the ContentInfo structure (after the explicit [0] tag).

CMS***Connectors are useful in creating and reading nested objects.

Using the CMS***OutputStream and CMS***InputStream Classes

To construct an object:

1. Create a CMS***OutputStream class of the appropriate content type. All the relevant parameters are passed through the constructor.
2. Write the data being protected to the CMS***OutputStream created in step 1.
3. After all the data is written, close the CMS***OutputStream created in step 1.

To read an object:

### Table 5–11 The CMS***Connector Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Content Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSDigestedDataInputConnector,</td>
<td>CMS.id_digestedData</td>
</tr>
<tr>
<td>CMSDigestedDataOutputConnector</td>
<td></td>
</tr>
<tr>
<td>CMSSignedDataInputConnector,</td>
<td>CMS.id_signedData</td>
</tr>
<tr>
<td>CMSSignedDataOutputConnector</td>
<td></td>
</tr>
<tr>
<td>CMSEncryptedDataInputConnector,</td>
<td>CMS.id_encryptedData</td>
</tr>
<tr>
<td>CMSEncryptedDataOutputConnector</td>
<td></td>
</tr>
<tr>
<td>CMSEnvelopedDataInputConnector,</td>
<td>CMS.id_envelopedData</td>
</tr>
<tr>
<td>CMSEnvelopedDataOutputConnector</td>
<td></td>
</tr>
<tr>
<td>CMSAuthenticatedDataInputConnector,</td>
<td>CMS.id_ct-authData</td>
</tr>
<tr>
<td>CMSAuthenticatedDataOutputConnector</td>
<td></td>
</tr>
</tbody>
</table>
1. Create a CMS***InputStream class of the appropriate content type by passing the underlying input stream through the constructor.

2. Read the protected data from the CMS***InputStream created in step 1 using the read() and read(byte[],...) methods.

3. Invoke terminate() after you have finished reading data from the CMS***InputStream created in step 1. This completes the reading of the object.

4. Invoke the appropriate methods to verify that the protected content is secure.

**CMS id-data Object**

The getData() method returns the data which can then be written to a CMS***OutputStream or CMS***OutputConnector.

**CMS id-ct-receipt Object**

The getReceiptData() method returns the encoded receipt which can then be written to a CMS***OutputStream or CMS***OutputConnector.

To read ESSReceipt data from the input stream:

```java
byte[] rcptData = in.read(...);
ESSReceipt er = new ESSReceipt();
er.inputContent(rcptData);
```

**CMS id-digestedData Object**

You will not be able to verify the digest of a detached digested-data object. Setting the boolean parameter writeEContentInfo in the CMSDigestedDataOutputStream constructor to false enables you to create a detached digested-data object.

**CMS id-signedData Object**

You will not be able to verify the signature of a detached signed-data object.

The CMSSignerInfoSpec class stores signer-specific information. For every signature you want to add, you will need to create a corresponding CMSSignerInfoSpec object which is then passed to the constructor.

Setting the boolean parameter createExternalSignatures in the CMSSignedDataOutputStream constructor to true enables you to create a detached signed-data object or external signatures.

To create a Certificate/CRL only object, do not pass any signer information to the CMSSignedDataOutputStream constructor.

**CMS id-encryptedData Objects**

Setting the boolean parameter writeEncryptedOutput in the CMSEncryptedDataOutputStream constructor to false enables you to create a detached encrypted-data object.

**CMS id-envelopedData Objects**

The CMSRecipientInfoSpec class stores recipient-specific information. For every recipient you want to add, you will need to create a corresponding CMSRecipientInfoSpec object which is then passed to the constructor.

Setting the boolean parameter writeContent in the CMSEnvelopedDataOutputStream constructor to false enables you to create a detached enveloped-data object.
Key Transport Key Exchange Mechanism
Use the CMSKeyTransRecipientInfoSpec class to store recipient information that uses the key transport key management mechanism.

Key Agreement Key Exchange Mechanism
This mechanism is not supported at this time.

Key Encryption (wrap) Key Exchange Mechanism
Use the CMSKEKRecipientInfoSpec class to store recipient information that uses the key wrap key management mechanism.

CMS id-ct-authData Objects
You will not be able to verify the MAC of a detached authenticated-data object.

Setting the boolean parameter detachEncapContent in the CMSAuthenticatedDataOutputStream constructor to true enables you to create a detached authenticated-data object.

Wrapping (Triple or more) CMSConnector Objects
You use CMSConnectorOutputConnectors to create nested objects.

Use the following code to create signed, enveloped, digested, and encrypted data and write it to the file nested.p7m:

```java
FileOutputStream fos = new FileOutputStream("nested.p7m");
CMSSignedDataOutputConnector conn1 =
new CMSKVerifyDataOutputConnector(fos, ....);
CMSEnvelopedDataOutputConnector conn2 =
new CMSAuthenticatedDataOutputConnector(conn1, ...);
CMSDigestedDataOutputConnector conn3 =
new CMSDigestedDataOutputConnector(conn2, ...);
CMSEncryptedDataOutputConnector conn4 =
new CMSEncryptedDataOutputConnector(conn3, ...);
OutputStream os = conn4.getOutputStream();
os.write(data);
os.close();
```

To read signed, enveloped, digested, and encrypted data stored in file nested.p7m:

```java
FileInputStream fis = new FileInputStream("nested.p7m");
CMSSignedDataInputConnector conn1 =
new CMSSignedDataInputConnector(fos, ....);
CMSEnvelopedDataInputConnector conn2 =
new CMSAuthenticatedDataInputConnector(conn1, ...);
CMSDigestedDataInputConnector conn3 =
new CMSDigestedDataInputConnector(conn2, ...);
CMSEncryptedDataInputConnector conn4 =
new CMSEncryptedDataInputConnector(conn3, ...);
```
Example Programs

```java
CMSEncryptedDataInputConnector conn4 =
    new CMSEncryptedDataInputConnector(conn3, ...);
InputStream is = conn4.getInputStream();
is.read(data);
```

The Oracle CMS Java API Reference

The Oracle CMS API Reference (Javadoc) is available at:

Oracle Security Developer Tools CMS Java API Reference

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at

This chapter provides an overview of Oracle S/MIME, describes key features and benefits, and explains how to set up and use Oracle S/MIME.

This chapter contains these topics:

- Oracle S/MIME Features and Benefits
- Setting Up Your Oracle S/MIME Environment
- Developing Applications with Oracle S/MIME
- The Oracle S/MIME Java API Reference
- Example Programs

Oracle S/MIME Features and Benefits

Oracle S/MIME is a pure Java solution which provides the following features:

- Full support for X.509 Version 3 certificates with extensions, including certificate parsing and verification
- Support for X.509 certificate chains in PKCS #7 and PKCS #12 formats
- Private key encryption using PKCS #5, PKCS #8, and PKCS #12
- An integrated ASN.1 library for input and output of data in ASN.1 DER/BER format

Setting Up Your Oracle S/MIME Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME. This section explains how to set up your environment for Oracle S/MIME. It contains these topics:

- System Requirements for Oracle S/MIME
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle S/MIME

In order to use Oracle S/MIME, your system must have the Java Development Kit (JDK) version 1.2.2 or higher. Oracle S/MIME also requires:

- An implementation of the JavaBeans Activation Framework (JAF). Sun's royalty-free implementation is available at:
  http://www.javasoft.com/beans/glasgow/jaf.html
Setting Up Your Oracle S/MIME Environment

- An implementation of the JavaMail API. Sun's royalty-free implementation is available at:
  

If you are using POP or IMAP, be sure to download Sun's POP3 (or IMAP) Provider, which is also available at the JavaMail page.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar file
- osdt_cert.jar file
- osdt_cms.jar file
- osdt_smime.jar file
- Your JAF (Java Activation Framework), JavaMail, and POP3 provider installations.

Any application using the Oracle S/MIME API must have all the necessary MIME types registered in its command map.

Some applications, specifically those reading S/MIME entries from a FileDataSource, will need to register the S/MIME file types.

Setting the CLASSPATH on Windows

To set the CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all the required jar and class files to the CLASSPATH. For example, your CLASSPATH might look like this:

   %CLASSPATH%;C:\ORACLE_HOME\jlib\osdt_core.jar;
   C:\ORACLE_HOME\jlib\osdt_cert.jar;
   C:\ORACLE_HOME\jlib\osdt_cms.jar;C:\ORACLE_HOME\jlib\osdt_smime.jar;
   C:\jaf-1.0.2\activation.jar;C:\javamail-1.3.1\mail.jar;

6. Click OK.

Setting the CLASSPATH on UNIX

On UNIX, set your CLASSPATH environment variable to include the full path and file names of all the required jar and class files. For example:

    setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar: 
    $ORACLE_HOME/jlib/osdt_cert.jar: 
    $ORACLE_HOME/jlib/osdt_cms.jar:$ORACLE_HOME/jlib/osdt_smime.jar: 
    /usr/lib/jaf-1.0.2/activation.jar:/usr/lib/javamail-1.3.1/mail.jar
Developing Applications with Oracle S/MIME

This section describes selected interfaces and classes in the Oracle S/MIME API and illustrates their use. It includes these topics:

- Core Classes and Interfaces
- Supporting Classes and Interfaces
- Using the Oracle S/MIME Classes

Selected methods are described as appropriate.

Core Classes and Interfaces

This section describes core classes and interfaces in the Oracle S/MIME API, and explains how to create and parse S/MIME objects.

The oracle.security.crypto.smime.SmimeObject Interface

The oracle.security.crypto.smime.SmimeObject interface represents an S/MIME object. Classes that implement this interface include:

- SmimeSigned
- SmimeEnveloped
- SmimeMultipartSigned
- SmimeSignedReceipt
- SmimeCompressed

Methods in this interface include:

```java
String generateContentType ()
Returns the content type string for this S/MIME object. For example:
"application/pkcs7-mime; smime-type=signed-data"
```

```java
String generateContentType (boolean useStandardContentTypes)
If the argument is true, returns the same as generateContentType(); if false, returns old-style (Netscape) content type string. For example:
"application/x-pkcs7-mime; smime-type=signed-data"
```

```java
void writeTo (java.io.OutputStream os, java.lang.String mimeType)
Outputs this object to the specified output stream.
```

The oracle.security.crypto.smime.SmimeSignedObject Interface

The oracle.security.crypto.smime.SmimeSignedObject interface extends SmimeObject, and specifies methods common to all S/MIME signed objects, including SmimeSigned and SmimeMultipartSigned.

Methods in this interface include:

```java
Vector getCertificates ()
Returns the list of certificates included in this S/MIME object's signed content.
```

```java
Vector getCRLs ()
Returns the list of certificate revocation lists in the S/MIME object's signed content.
```

```java
javax.mail.internet.MimeBodyPart getEnclosedBodyPart ()
Returns the document which was signed.
```

```java
oracle.security.crypto.smime.ess.EquivalentLabels getEquivalentLabels
```
Developing Applications with Oracle S/MIME

oracle.security.crypto.cert.X509 signerCert
Returns the EquivalentLabels if present or null.

oracle.security.crypto.smime.ess.ESSSecurityLabel getESSSecurityLabel
(oracle.security.crypto.cert.X509 signerCert)
Returns the ESSSecurityLabel if present or null.

oracle.security.crypto.smime.ess.MLExpansionHistory getMLExpansionHistory(oracle.security.crypto.cert.X509 signerCert)
Returns the MLExpansionHistory attribute if present or null.

oracle.security.crypto.smime.ess.ReceiptRequest getReceiptRequest(oracle.security.crypto.cert.X509 signerCert)
Returns the ReceiptRequest attribute if present or null.

oracle.security.crypto.smime.ess.SigningCertificate getSigningCertificate(oracle.security.crypto.cert.X509 signerCert)
Returns the SigningCertificate.

void verify (oracle.security.crypto.cert.CertificateTrustPolicy trustPolicy)
Returns normally if the signed contents include at least one valid signature according to the specified trust policy, otherwise throws an AuthenticationException.

void verifySignature (oracle.security.crypto.cert.X509 signerCert)
Returns normally if the signed contents contain a signature which can be validated by the given certificate, otherwise throws an AuthenticationException.

The oracle.security.crypto.smime.SmimeSigned Class

The oracle.security.crypto.smime.SmimeSigned class represents an S/MIME signed message (implements SmimeSignedObject). You may use this class to build a new message or parse an existing one.

Constructors and methods include:

SmimeSigned (javax.mail.internet.MimeBodyPart content)
Creates a new SmimeSigned object, using the specified MIME body part for the contents to be signed.

SmimeSigned ()
Creates a new empty SmimeSigned object, which is useful for building a "certificates-only" S/MIME message.

SmimeSigned (InputStream is)
Creates a new SmimeSigned object by reading its encoding from the specified input stream.

void addSignature (oracle.security.crypto.core.PrivateKey signerKey
oracle.security.crypto.cert.X509 signerCert,
oracle.security.crypto.core.AlgorithmIdentifier digestAlgID)
Adds a signature to the message, using the specified private key, certificate, and message digest algorithm.

void addSignature (oracle.security.crypto.core.PrivateKey signerKey,
oracle.security.crypto.cert.X509 signerCert,
oracle.security.crypto.core.AlgorithmIdentifier digestAlgID,
java.util.Date timeStamp)
Adds a signature to the message, including a time stamp.

void addSignature (oracle.security.crypto.core.PrivateKey signerKey,
oracle.security.crypto.cert.X509 signerCert,
oracle.security.crypto.core.AlgorithmIdentifier digestAlgID,
SmimeCapabilities smimeCaps)
Adds a signature to the message, including S/MIME capabilities.
javax.mail.internet.MimeBodyPart getEnclosedBodyPart ()
Returns the MIME body part that was signed.

To build a new message, use any of these three constructors:

// Create a new S/MIME Signed Message
SmimeSigned sig = new SmimeSigned();

// -OR-
// Create a new S/MIME Signed Message with a specified MIME body part
MimeBodyPart bp = new MimeBodyPart();
bp.setText("Hello from SendSignedMsg!");
SmimeSigned sig1 = new SmimeSigned(bp);

// -OR-
// Create a new S/MIME Signed Message with a specified MIME body part
// and a flag switching compression on or off
MimeBodyPart bp = new MimeBodyPart();
bp.setText("Hello from SendSignedMsg!");
boolean useCompression = true;
SmimeSigned sig2 = new SmimeSigned(bp, useCompression);

To parse a message, use the constructor that takes a java.io.InputStream:

InputStream is = InputStream containing message to be parsed
SmimeSigned sig = new SmimeSigned(is);

**The oracle.security.crypto.smime.SmimeEnveloped Class**

The oracle.security.crypto.smime.SmimeEnveloped class represents an S/MIME enveloped message (implements SmimeObject), and may be used to build a new message or parse an existing one.

Constructors and methods include:

SmimeEnveloped (javax.mail.internet.MimeBodyPart content, oracle.security.crypto.core.AlgorithmIdentifier contentEncryptionAlgID)
Creates a new SmimeEnveloped object from the specified MIME body part, using the specified content encryption algorithm.

SmimeEnveloped (InputStream is)
Creates a new SmimeEnveloped object by reading its encoding from the specified input stream.

void addRecipient (oracle.security.crypto.cert.X509 cert)
Encrypts the message for the recipient using the given public key certificate.

byte[] getEncryptedContent ()
Returns the contents without decrypting.

javax.mail.internet.MimeBodyPart getEnclosedBodyPart ( oracle.security.crypto.core.PrivateKey recipientKey, oracle.security.crypto.cert.X509 recipientCert)
Returns the MIME body part for the recipient specified by recipientCert, after decryption using the given recipient private key.

Use the following code to build a new message:

// Create a new S/MIME Enveloped Message with a specified MIME body part and a specified content
// encryption algorithm
MimeBodyPart bp = new MimeBodyPart();
bp.setText("Hello from SendSignedMsg!");
AlgorithmIdentifier algId = AlgID.aes256_CBC;
SmimeEnveloped env = new SmimeEnveloped(bp, algId);

To parse a message, use the constructor that takes a java.io.InputStream:

```
InputStream is = InputStream containing message to be parsed
SmimeEnveloped env = new SmimeEnveloped(is);
```

**The oracle.security.crypto.smime.SmimeMultipartSigned Class**

The oracle.security.crypto.smime.SmimeMultipartSigned class represents an S/MIME multi-part signed message. A multipart signed message is intended for email clients that are not MIME-aware. This class can be used to build a new message or parse an existing one.

Constructors and methods include:

- `SmimeMultipartSigned (javax.mail.internet.MimeBodyPart bodyPart, oracle.security.crypto.core.AlgorithmIdentifier digestAlgID)`
  Creates a new SmimeMultipartSigned message, with the specified MIME body part and message digest algorithm.

- `void addBodyPart (javax.mail.BodyPart part)`
  Inherited from javax.mail.Multipart, adds the specified body part to this SmimeMultipartSigned object. (See the javax.mail API documentation for more details.)

- `void addSignature (oracle.security.crypto.core.PrivateKey signerKey, oracle.security.crypto.cert.X509 signerCert)`
  Adds a signature to the message, using the specified private key and certificate.

- `void addSignature (oracle.security.crypto.core.PrivateKey signerKey, oracle.security.crypto.cert.X509 signerCert, java.util.Date timeStamp)`
  Adds a signature to the message, using the specified private key and certificate plus a time stamp.

- `void addSignature (oracle.security.crypto.core.PrivateKey signerKey, oracle.security.crypto.cert.X509 signerCert, java.util.Date timeStamp, SmimeCapabilities smimeCaps)`
  Adds a signature to the message, using the specified private key and certificate, plus S/MIME capabilities.

- `javax.mail.internet.MimeBodyPart getEnclosedBodyPart ()`
  Returns the MIME body part that was signed.

Use the following code to build a new message:

```
// Create a new S/MIME Multipart Signed Message with a specified
// MIME body part and a specified digest algorithm
MimeBodyPart bp = new MimeBodyPart();
bp.setText("Hello from SendSignedMsg!");
AlgorithmIdentifier algId = AlgID.sha1;
SmimeMultipartSigned sig = new SmimeMultipartSigned(bp, algId);
```

To parse a message, use the constructor that takes a javax.activation.DataSource:

```
DataSource ds = Data source containing message to be parsed
SmimeMultipartSigned sig = new SmimeMultipartSigned(ds);
```

**The oracle.security.crypto.smime.SmimeSignedReceipt Class**

The oracle.security.crypto.smime.SmimeSignedReceipt class represents an S/MIME wrapped and signed receipt. You may use this class to build a new message or parse an existing one.
To build a new message, use any of these four constructors:

```java
// Create a new S/MIME wrapped and signed receipt with the specified receipt,
// the specified digest of the message's signed attributes
// and the addresses of the receipt recipients
ESSReceipt receipt = ESS receipt to include in message
byte[] msgSigDigest = Digest of signed attributes to be included in message
Address[] addresses = Addresses of receipt recipients
SmimeSignedReceipt sig = new SmimeSigned(receipt, msgSigDigest, addresses);

// -OR-
// Create a new S/MIME wrapped and signed receipt
// with a specified S/MIME Signed Message containing the receipt
SmimeSignedObject sso = S/MIME signed message containing receipt
SmimeSignedReceipt sig1 = new SmimeSignedReceipt(sso);

// -OR-
// Create a new S/MIME wrapped and signed receipt with a
// specified S/MIME Signed Message containing the receipt,
// the signer's certificate and the addresses of the receipt recipients
SmimeSignedObject sso1 = S/MIME signed message containing receipt
X509 signerCert = The message signer's certificate
Address[] addresses1 = Addresses of receipt recipients
SmimeSignedReceipt sig2 = new SmimeSignedReceipt(sso1, signerCert, addresses1);

// -OR-
// Create a new S/MIME wrapped and signed receipt with a
// specified S/MIME Signed Message containing the receipt,
// the signer's certificate, the addresses of the receipt recipients and
// a specified MLExpansionHistory attribute.
SmimeSignedObject sso1 = S/MIME signed message containing receipt
X509 signerCert = The message signer's certificate
Address[] addresses1 = Addresses of receipt recipients
MLExpansionHistory mlExpansionHistory = The MLExpansionHistory attribute
SmimeSignedReceipt sig2 =
    new SmimeSignedReceipt(sso1, signerCert, addresses1, mlExpansionHistory);
```

To parse a message, use the constructor that takes a `java.io.InputStream`:

```java
InputStream is = Input stream containing message to be parsed
SmimeSignedReceipt sig = new SmimeSignedReceipt(is);
```

**The oracle.security.crypto.smime.SmimeCompressed Class**

The `oracle.security.crypto.smime.SmimeCompressed` class represents an S/MIME compressed message as defined in RFC 3274. You can use this class to build a new message or parse an existing one.

**Note:** A link to RFC 3274 is available in Appendix A, "References".

Use the following code to build a new message:

```java
// Create a new S/MIME Compressed Message with a specified MIME body part
MimeBodyPart bp = new MimeBodyPart();
bp.setText("Hello from SendSignedMsg!");
SmimeCompressed comp = new SmimeCompressed(bp);

// -OR-
// Create a new S/MIME Compressed Message with a specified MIME body part
```
// and a specified compression algorithm
MimeBodyPart bp = new MimeBodyPart();
bp.setText("Hello from SendSignedMsg!");
AlgorithmIdentifier algId = Smime.id_alg_zlibCompress;
SmimeCompressed comp = new SmimeCompressed(bp, algId);

To parse a message, use the constructor that takes a java.io.InputStream:
InputStream is = Input stream containing message to be parsed
SmimeCompressed compl = new SmimeCompressed(is);

Supporting Classes and Interfaces
This section describes Oracle S/MIME supporting classes and interfaces.

The oracle.security.crypto.smime.Smime Interface
The oracle.security.crypto.smime.Smime interface defines constants such as
algorithm identifiers, content type identifiers, and attribute identifiers.

The oracle.security.crypto.smime.SmimeUtils Class
The oracle.security.crypto.smime.SmimeUtils class contains static utility
methods.
Methods of this class include:
public static FileDataSource createFileDataSource (File file,
String contentTypeHeader)
public static FileDataSource createFileDataSource (String name,
String contentTypeHeader)

For transparent handling of multipart or multipart/signed S/MIME types, use these
methods instead of directly instantiating a javax.activation.FileDataSource.

Note: The default javax.activation.FileDataSource
included with JAF 1.0.1 does not handle multipart MIME boundaries
when used with Javamail 1.1.x.

The oracle.security.crypto.smime.MailTrustPolicy Class
The oracle.security.crypto.smime.MailTrustPolicy class implements a
certificate trust policy
(oracle.security.crypto.cert.CertificateTrustPolicy) used to verify
signatures on signed S/MIME objects.

The oracle.security.crypto.smime.SmimCapabilities Class
The oracle.security.crypto.smime.SmimCapabilities class encapsulates
a set of capabilities for an S/MIME object including, for example, the supported
encryption algorithms.
A useful method of this class is:
void addCapability(oracle.security.crypto.asn1.ASN1ObjectID capabilityID)
which adds the capability with the specified object ID to this set of S/MIME
capabilities.
The oracle.security.crypto.smime.SmimeDataContentHandler Class
The oracle.security.crypto.smime.SmimeDataContentHandler class provides the DataContentHandler for S/MIME content types. It implements javax.activation.DataContentHandler.

The oracle.security.crypto.smime.ess Package
The oracle.security.crypto.smime.ess package contains the following classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentHints</td>
<td>Content hints</td>
</tr>
<tr>
<td>ContentReference</td>
<td>Content reference</td>
</tr>
<tr>
<td>EquivalentLabels</td>
<td>ESS EquivalentLabels</td>
</tr>
<tr>
<td>ESSCertID</td>
<td>Represents the ESSCertID of a certificate which is used in the Signing Certificate Attribute</td>
</tr>
<tr>
<td>ESSSecurityLabel</td>
<td>An ESS security label</td>
</tr>
<tr>
<td>GeneralNames</td>
<td>The GeneralNames type, which is a SEQUENCE of the GeneralName type defined in RFC 2459 (a link to RFC 2459 is available in Appendix A, &quot;References&quot;)</td>
</tr>
<tr>
<td>MLDData</td>
<td>Represents the MLDData element which is used in the MExpansionHistory attribute</td>
</tr>
<tr>
<td>MExpansionHistory</td>
<td>Mailing list expansion history</td>
</tr>
<tr>
<td>ReceiptRequest</td>
<td>An ESS Receipt Request</td>
</tr>
<tr>
<td>ReceiptRequest.AllOrFirstTier</td>
<td>A 'AllOrFirstTier' is a part of the 'ReceiptsFrom' field of a ReceiptRequest</td>
</tr>
<tr>
<td>SigningCertificate</td>
<td>An ESS Signing Certificate</td>
</tr>
</tbody>
</table>

Using the Oracle S/MIME Classes
This section describes how to use the Oracle S/MIME SDK to work with multi-part signed messages, create and open digital envelopes, and implement Enhanced Security Services (ESS). It covers these topics:

- Using the Abstract Class SmimeObject
- Signing Messages
- Creating "Multipart/Signed" Entities
- Creating Digital Envelopes
- Creating "Certificates-Only" Messages
- Reading Messages
- Authenticating Signed Messages
- Opening Digital Envelopes (Encrypted Messages)
- Adding Enhanced Security Services (ESS)
Using the Abstract Class SmimeObject

SmimeObject is an abstract class representing a fundamental S/MIME message content entity. Subclasses of SmimeObject include SmimeSigned, SmimeEnveloped, and SmimeMultipartSigned.

One of the characteristics of SmimeObject implementations is that they "know their own MIME type" -- that is, they implement the generateContentType method. Thus, to place such an object inside a MIME message or body part, follow the same outline that was used in the SmimeSigned example:

1. Create the object.
2. Invoke generateContentType on the object to obtain a MIME type.
3. Pass the object, together with the generated content type, to the setContent method of a MimeMessage or MimeBodyPart object.

The SmimeObject class provides another version of the generateContentType method, which takes a boolean parameter. When given true as a parameter, generateContentType behaves exactly as in the case of no argument. When given false as a parameter, generateContentType returns the older MIME types required by certain mail clients, including Netscape Communicator 4.0.4. Specifically:

- "application/pkcs7-mime" becomes "application/x-pkcs7-mime"
- "application/pkcs7-signature" becomes "application/x-pkcs7-signature"

Signing Messages

Create a signed message, or signed MIME body part, using these steps:

1. Prepare an instance of MimeBodyPart which contains the content you wish to sign. This body part may have any content-type desired. In the following example we create a "text/plain" body part:

   MimeBodyPart doc = new MimeBodyPart();
   doc.setText("Example signed message.");

2. Create an instance of SmimeSigned using the constructor which takes the MimeBodyPart created earlier as argument.

   SmimeSigned sig = new SmimeSigned (doc);

3. Add all desired signatures. For each signature, you need to specify a private key, a certificate for the matching public key, and a message digest algorithm. For example:

   sig.addSignature (signatureKey, signatureCert, AlgID.sha1);

   In this example we specified the SHA-1 message digest algorithm. Alternatively, we could have specified the MD5 algorithm by passing AlgID.md5 as the argument.

4. Place your SmimeSignedObject into a MimeMessage or MimeBodyPart, as appropriate. For example:

   MimeMessage m = new MimeMessage();
   m.setContent (sig, sig.generateContentType());

   or

   MimeBodyPart bp = new MimeBodyPart();
   bp.setContent (sig, sig.generateContentType());
The `generateContentType` method used in these examples returns a string identifying the appropriate MIME type for the object, which in this case is:

```
application/pkcs7-mime; smime-type=signed-data
```

With these simple steps, you can now transport the MIME message, place the body part containing S/MIME content into a MIME multipart object, or perform any other operation appropriate for these objects. See the JavaMail API for details.

**Creating "Multipart/Signed" Entities**

The `SmimeMultipartSigned` class provides an alternative way to create signed messages. These messages use the "multipart/signed" mime type instead of "application/pkcs7-mime". The advantage is that the content of the resulting message is readable with non-MIME enabled mail clients, although such clients will not, of course, be able to verify the signature.

Creating a multi-part/signed message is slightly different from creating a signed message. For example, to send a multi-part/signed text message:

```java
// create the content text as a MIME body part
MimeBodyPart bp = new MimeBodyPart();
bp.setText("Example multipart/signed message.");
// the constructor takes the signature algorithm
SmimeMultipartSigned sig = new SmimeMultipartSigned(bp, AlgID.sha1);
// sign the content
sig.addSignature(signerKey, signerCert);
// place the content in a MIME message
MimeMessage msg = new MimeMessage();
msg.setContent(sig, sig.generateContentType());
```

The reason for identifying the message digest in the `SmimeMultipartSigned` constructor is that, unlike the case of application/pkcs7-mime signed data objects, multipart/signed messages require that all signatures use the same message digest algorithm.

The `generateContentType` method returns the following string:

```
multipart/signed; protocol="application/pkcs7-signature"
```

**Creating Digital Envelopes**

An S/MIME digital envelope (encrypted message) is represented by the `SmimeEnveloped` class. This is a MIME entity which is formed by encrypting a MIME body part with some symmetric encryption algorithm (eg, Triple-Des or RC2) and a randomly generated session key, then encrypting the session key with the RSA public key for each intended message recipient.

In the following example, `doc` is an instance of `MimeBodyPart`, which is to be wrapped in an instance of `SmimeEnveloped`, and `recipientCert` is the recipient's certificate.

```java
SmimeEnveloped env = new SmimeEnveloped(doc, Smime.dES_EDE3_CBC);
env.addRecipient(recipientCert);
```

Any number of envelope recipients may be added by making repeated calls to `addRecipient`.

---

Oracle S/MIME 6-11
Creating "Certificates-Only" Messages

It is possible to create an S/MIME signed-data object that contains neither content nor signatures; rather, it contains just certificates, or CRLs, or both. Such entities can be used as a certificate transport mechanism. They have the special content type:

application/pkcs7-mime; smime-type=certs-only

Here is an example:

X509 cert1, cert2;
SmimeSigned certBag = new SmimeSigned();
certBag.addCertificate(cert1);
certBag.addCertificate(cert2);

Now you can pass certBag to an appropriate setContent method. When generateContentType is invoked on certBag, it will automatically return a content type with the correct "certs-only" value for the smime-type parameter.

Reading Messages

The basic JavaMail API technique for extracting Java objects from MIME entities is to invoke the getContent() method on an instance of MimePart, an interface which models MIME entities and is implemented by the MimeMessage and MimeBodyPart classes.

The getContent method consults the currently installed default command map - which is part of the JavaBeans Activities Framework - to find a data content handler for the given MIME type, which is responsible for converting the content of the MIME entity into a Java object of the appropriate class.

The mailcap file provided with your distribution can be used to install the SmimeDataContentHandler class, which serves as a data content handler for the following types:

<table>
<thead>
<tr>
<th>Content Type</th>
<th>Returns Instance Of</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/pkcs7-mime</td>
<td>SmimeSigned or Smime Enveloped</td>
</tr>
<tr>
<td>application/pkcs7-signature</td>
<td>SmimeSigned</td>
</tr>
<tr>
<td>application/pkcs10</td>
<td>oracle.security.crypto.cert.CertificateRequest</td>
</tr>
<tr>
<td>multipart/signed</td>
<td>SmimeMultipartSigned</td>
</tr>
</tbody>
</table>

Authenticating Signed Messages

Once you obtain an instance of SmimeSigned or SmimeMultipartSigned from getContent(), you will naturally want to verify the attached signatures. To explain the available options for signature verification, it is necessary to discuss the structure of an S/MIME signed message.

The content of a signed S/MIME message is a CMS object of type SignedData. Such an object itself has a content - the document to which the signatures are applied - which is the text encoding of a MIME entity. It also contains from zero to any number of signatures, and, optionally, a set of certificates, CRLs, or both, which the receiving party may use to validate the signatures.

The SmimeSigned and SmimeMultipartSigned classes encapsulate all of this information. They provide two authentication methods: verifyingSignature and verify.
To verify a particular signature with a certificate already in possession, ignoring any certificate and CRLs attached by the signer, use verifySignature. For example:

```java
SmimeSignedObject sig =
    (SmimeSignedObject)msg.getContent(); // msg is a Message
    sig.verifySignature(cert, msg.getFrom()); // cert is an X509 object
```

If verification fails, the verifySignature method throws either an UnknownSignerException or an AuthenticationException; otherwise, it returns normally.

Use verify to verify that the content contains at least one valid signature; that is, there exists a valid certificate chain, starting from a trusted root CA, and terminating in a certificate for the private key which generated the signature. This method makes use of the attached certificate and CRLs in order to follow certificate chains.

For example, given a trusted certificate authority (CA) certificate already in hand:

```java
TrustedCAPolicy trusts = new TrustedCAPolicy();
    // if true, need CRL for each cert in chain
trusting.setRequireCRLs(false);
    // caCert is an X509 object with CA cert
    trusting.addTrustedCA(caCert);
SmimeSignedObject sig = (SmimeSignedObject)msg.getContent();
    sig.verify(trusts, msg.getFrom());
```

Like verifySignature, verify throws an AuthenticationException if the signature cannot be verified; otherwise it returns normally. In either case you can recover the document that was signed, which is itself a MIME entity, by invoking getEnclosedBodyPart():

```java
MimeBodyPart doc = sig.getEnclosedBodyPart();
```

### Opening Digital Envelopes (Encrypted Messages)

An S/MIME digital envelope consists of:

- A protected MIME body part, which has been encrypted with a symmetric key algorithm (for example, DES or RC2)
- A randomly generated content encryption key
- Information that allows one or more intended recipients to decrypt the content

For each recipient, this information consists of the content encryption key, itself encrypted with the recipient’s public key.

To obtain the encrypted content from an SmimeEnveloped object, you need the recipient’s private key and the corresponding certificate; the certificate is used as an index into the recipient information table contained in the envelope’s data structure.

For example:

```java
SmimeEnveloped env = (SmimeEnveloped)msg.getContent();
MimeBodyPart mbp = env.getEnclosedBodyPart(privKey, cert)
    // privKey is a PrivateKey object
    // cert is an X509 object
```

Passing the private key and the certificate to the getEnclosedBodyPart method returns the decrypted content as an instance of MimeBodyPart.

The getContent method can now be invoked on the MimeBodyPart object to retrieve the (now decrypted) content. This content may be a String (in the case of an encrypted text message), or any other object such as an SmimeSigned.
Adding Enhanced Security Services (ESS)
You can add the ESS services ReceiptRequests, SecurityLabels, and SigningCertificates to an S/MIME signed message by adding them to the signedAttributes of a signature.

```java
// Create a Signed Message
SmimeSigned sig = new SmimeSigned();
    AttributeSet signedAttributes = new AttributeSet();
```

**Receipt Request**  (oracle.security.crypto.smime.ess.ReceiptRequest)
To request a signed receipt from the recipient of a message, add a receiptRequest attribute to the signedAttributes field while adding a signature:

```java
ReceiptRequest rr = new ReceiptRequest();
        .......
    signedAttributes.addAttribute(Smime.id_aa_receiptRequest, rr);
```

**Security Label**  (oracle.security.crypto.smime.ess.ESSSecurityLabel)
To attach a security label to a message, add an ESSSecurityLabel attribute to the signedAttributes field while adding a signature:

```java
ESSSecurityLabel sl = new ESSSecurityLabel();
        .......
    signedAttributes.addAttribute(Smime.id_aa_securityLabel, sl);
```

**Signing Certificate**  (oracle.security.crypto.smime.ess.SigningCertificate)
To attach a signing certificate to a message, add a SigningCertificate attribute to the signedAttributes field while adding a signature:

```java
SigningCertificate sc = new SigningCertificate();
        .......
    signedAttributes.addAttribute(Smime.id_aa_signingCertificate, sc);
```

Use the signedAttributes while adding a signature:

```java
    sig.addSignature(signerKey, signerCert, digestAlgID, signedAttributes);
```

The ESS signed receipts are generated using the SmimeSignedReceipt class in the oracle.security.crypto.smime package, in a manner similar to using a SmimeSigned class, except that the content that is signed is an oracle.security.crypto.cms.ESSReceipt object.

Processing Enhanced Security Services (ESS)
An S/MIME signed receipt must have correctly set content type parameters for the data content handlers to recognize it. If the content type parameters are missing, the signed receipt is treated as a signed message.
Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at
A public key infrastructure (PKI) is a security architecture that provides an increased level of confidence for exchanging information over the Internet.

This chapter provides information about using the packages in Oracle PKI SDK, which is a set of software development kits (SDKs) for developing PKI-aware applications.

This chapter contains the following topics:

- Oracle PKI SDK CMP
- Oracle PKI SDK OCSP
- Oracle PKI SDK TSP
- Oracle PKI SDK LDAP

### Oracle PKI SDK CMP

This section provides information about using the Oracle public key infrastructure (PKI) Software Development Kit (SDK) for certificate management protocol (CMP). Oracle PKI SDK CMP allows Java developers to quickly implement certificate management functionality such as issuing and renewing certificates, creating and publishing CRLs, and providing key recovery capabilities.

This chapter contains the following topics:

- Oracle PKI SDK CMP Features and Benefits
- Setting Up Your Oracle PKI SDK CMP Environment
- The Oracle PKI SDK CMP Java API Reference
- Example Programs

### Oracle PKI SDK CMP Features and Benefits

The Oracle PKI SDK CMP provides the following features and functionality:

- Oracle PKI SDK CMP conforms to RFC 2510, and is compatible with other products that conform to this certificate management protocol (CMP) specification. RFC 2510 defines protocol messages for all aspects of certificate creation and management.

- Oracle PKI SDK CMP conforms to RFC 2511, and is compatible with other products that conform to this certificate request message format (CRMF) specification. RFC 2511 describes the Certificate Request Message Format (CRMF), which is used to convey X.509 certificate requests to a Certification Authority (CA).
Package Overview for Oracle PKI SDK CMP

The Oracle PKI SDK CMP toolkit contains the following packages:

- The oracle.security.crypto.cmp package provides classes that implement certificate management protocol (CMP) as described in RFC 2510, and certificate request message format (CRMF) as described in RFC 2511.
- The oracle.security.crypto.cmp.attribute package provides attribute classes for registration controls, registration information, and general information. This package includes the following classes and their subclasses:
  - RegistrationControl
  - RegistrationInfo
  - InfoTypeAndValue (which extends oracle.security.crypto.cert.AttributeTypeAndValue)
- The oracle.security.crypto.cmp.transport package provides classes for CMP and CRMF transport protocols. It includes the TCPMessage class and its specific message-type subclasses.

Setting Up Your Oracle PKI SDK CMP Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME. This section provides information for setting up your environment for Oracle PKI SDK CMP. It contains the following topics:

- System Requirements for Oracle PKI SDK CMP
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle PKI SDK CMP

In order to use Oracle PKI SDK CMP, your system must have the Java Development Kit (JDK) version 1.2.2 or higher.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar
- osdt_cert.jar
- osdt_cms.jar
- osdt_cmp.jar

Setting the CLASSPATH on Windows

To set your CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all of the required jar and class files to the CLASSPATH. For example:

C:\ORACLE_HOME\jlib\osdt_core.jar;C:\ORACLE_HOME\jlib\osdt_cert.jar;
C:\ORACLE_HOME\jlib\osdt_cms.jar;C:\ORACLE_HOME\jlib\osdt_cmp.jar

6. Click OK.

Setting the CLASSPATH on UNIX

On UNIX, set your CLASSPATH environment variable to include the full path and file names of all the required jar and class files. For example:

setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
$ORACLE_HOME/jlib/osdt_cert.jar:$ORACLE_HOME/jlib/osdt_cms.jar:
$ORACLE_HOME/jlib/osdt_cmp.jar

The Oracle PKI SDK CMP Java API Reference

The Oracle PKI SDK CMP Java API reference (Javadoc) is available at:

Oracle Security Developer Tools PKI SDK CMP Java API Reference

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at http://www.oracle.com/technology/index.html.

Oracle PKI SDK OCSP

This section provides information about using the Oracle Online Certificate Status Protocol (OCSP) Software Development Kit (SDK). Oracle PKI SDK OCSP allows Java developers to quickly develop OCSP-enabled client applications and OCSP responders that conform to RFC 2560 specifications.

This section contains the following topics:

- Oracle PKI SDK OCSP Features and Benefits
- Setting Up Your Oracle PKI SDK OCSP Environment
- The Oracle PKI SDK OCSP Java API Reference
- Example Programs

Oracle PKI SDK OCSP Features and Benefits

Oracle PKI SDK OCSP provides the following features and functionality:

- Oracle PKI SDK OCSP conforms to RFC 2560 and is compatible with other products that conform to this specification, such as Valicert’s Validation Authority. RFC 2560 specifies a protocol useful in determining the current status of a digital certificate without requiring CRLs.

- The Oracle PKI SDK OCSP API provides classes and methods for constructing OCSP request messages that can be sent through HTTP to any RFC 2560 compliant validation authority.

- The Oracle PKI SDK OCSP API provides classes and methods for constructing responses to OCSP request messages, and an OCSP server implementation that
you can use as a basis for developing your own OCSP server to check the validity of certificates you have issued.

Setting Up Your Oracle PKI SDK OCSP Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME. This section provides information for setting up your environment for Oracle PKI SDK OCSP. It contains the following topics:

- System Requirements for Oracle PKI SDK OCSP
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle PKI SDK OCSP

In order to use Oracle PKI SDK OCSP, your system must have the Java Development Kit (JDK) version 1.2.2 or higher. Also, make sure that your PATH environment variable includes the Java bin directory.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar
- osdt_cert.jar
- osdt_ocsp.jar

Setting the CLASSPATH on Windows

To set your CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all of the required jar and class files to the CLASSPATH. For example:
   
   C:\ORACLE_HOME\jlib\osdt_core.jar;C:\ORACLE_HOME\jlib\osdt_cert.jar;
   C:\ORACLE_HOME\jlib\osdt_ocsp.jar

6. Click OK.

Setting the CLASSPATH on Unix

On Unix, set your CLASSPATH environment variable to include the full path and file name of all the required jar and class files. For example:

```
setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
$ORACLE_HOME/jlib/osdt_cert.jar:
$ORACLE_HOME/jlib/osdt_ocsp.jar
```
The Oracle PKI SDK OCSP Java API Reference

The Oracle PKI SDK OCSP Java API reference (Javadoc) is available at:

Oracle Security Developer Tools PKI SDK OCSP Java API Reference

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at http://www.oracle.com/technology/index.html.

Oracle PKI SDK TSP

This section provides information about using the Oracle PKI SDK TSP, which allows Java developers to quickly implement time-stamping functionality within a public key infrastructure (PKI) framework.

This section contains the following topics:

- Oracle PKI SDK TSP Features and Benefits
- Setting Up Your Oracle PKI SDK TSP Environment
- The Oracle PKI SDK TSP Java API Reference
- Example Programs

Oracle PKI SDK TSP Features and Benefits

Oracle PKI SDK TSP provides the following features and functionality:

- Oracle PKI SDK TSP conforms to RFC 3161 and is compatible with other products that conform to this time stamp protocol (TSP) specification.
- Oracle PKI SDK TSP provides an example implementation of a TSA server to use for testing TSP request messages, or as a basis for developing your own time stamping service.

Class and Interface Overview for Oracle PKI SDK TSP

Oracle PKI SDK TSP contains the following classes and interfaces:

Table 7–1 Oracle PKI SDK TSP Classes and Interfaces

<table>
<thead>
<tr>
<th>Class or Interface Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSP Interface</td>
<td>Defines various constants associated with the Time Stamp Protocol (TSP).</td>
</tr>
<tr>
<td>HttpTSPRequest Class</td>
<td>Implementation of a TSP request message over HTTP.</td>
</tr>
<tr>
<td>HttpTSPResponse Class</td>
<td>Implementation of a TSP response message over HTTP.</td>
</tr>
<tr>
<td>MessageImprint Class</td>
<td>This class represents a MessageImprint object as defined in RFC 3161.</td>
</tr>
<tr>
<td>TSAPolicyID Class</td>
<td>This class represents a TSAPolicyID object as defined in RFC 3161.</td>
</tr>
<tr>
<td>TSPContentHandlerFactory Class</td>
<td>A content handler for TSP over HTTP.</td>
</tr>
<tr>
<td>TSPMessage Class</td>
<td>A TSP message.</td>
</tr>
</tbody>
</table>
Setting Up Your Oracle PKI SDK TSP Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in `ORACLE_HOME`. This section provides information for setting up your environment for Oracle PKI SDK TSP. It contains the following topics:

- System Requirements for Oracle PKI SDK TSP
- Setting the CLASSPATH Environment Variable

### System Requirements for Oracle PKI SDK TSP

In order to use Oracle PKI SDK TSP, your system must have the Java Development Kit (JDK) version 1.2.2 or higher. Also, make sure that your `PATH` environment variable includes the Java bin directory.

### Setting the CLASSPATH Environment Variable

Your `CLASSPATH` environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your `CLASSPATH`:

- `osdt_core.jar`
- `osdt_cert.jar`
- `osdt_cms.jar`
- `osdt_cmp.jar`
- `osdt_tsp.jar`

#### Setting the CLASSPATH on Windows

To set your `CLASSPATH` on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a `CLASSPATH` environment variable for your user profile. If a `CLASSPATH` environment variable already exists, select it and click Edit.
5. Add the full path and file names for all the required jar and class files to the `CLASSPATH`. For example:

```
%C\osdt_core.jar;
C:\osdt_cert.jar;
C:\osdt_cms.jar;C:\osdt_cmp.jar;
C:\osdt_tsp.jar
```
6. Click OK.
Setting the **CLASSPATH** on Unix

On Unix, set your **CLASSPATH** environment variable to include the full path and file name of all the required jar and class files. For example:

```bash
setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
$ORACLE_HOME/jlib/osdt_cert.jar:$ORACLE_HOME/jlib/osdt_cms.jar:
$ORACLE_HOME/jlib/osdt_cmp.jar:$ORACLE_HOME/jlib/osdt_tsp.jar
```

The Oracle PKI SDK TSP Java API Reference

The Oracle PKI SDK TSP Java API reference (Javadoc) is available at:

*Oracle PKI SDK TSP Java API Reference*

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at [http://www.oracle.com/technology/index.html](http://www.oracle.com/technology/index.html)

Oracle PKI SDK LDAP

This section provides information about using Oracle PKI SDK LDAP, which allows Java developers to quickly implement operations that involve publishing and retrieving digital certificates from a directory server.

This section contains the following topics:

- **Oracle PKI SDK LDAP Features and Benefits**
- **Setting Up Your Oracle PKI SDK LDAP Environment**
- **The Oracle PKI SDK LDAP Java API Reference**
- **Example Programs**

Oracle PKI SDK LDAP Features and Benefits

Oracle PKI SDK LDAP provides facilities for accessing a digital certificate within an LDAP directory. Some of the tasks you can perform with Oracle PKI SDK LDAP are:

- Validating a user’s certificate in an LDAP directory
- Adding a certificate to an LDAP directory
- Retrieving a certificate from an LDAP directory
- Deleting a certificate from an LDAP directory

Class Overview for Oracle PKI SDK LDAP

The `oracle.security.crypto.LDAP` package contains two classes:

- `LDAPCertificateValidator`, which validates a user certificate by checking whether it exists in its subject’s LDAP directory entry
- `LDAPUtils`, which is a collection of methods to add, retrieve, and remove certificates from a subject’s LDAP directory entry
Setting Up Your Oracle PKI SDK LDAP Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME. This section provides information on setting up your environment for Oracle PKI SDK LDAP. It contains the following topics:

- System Requirements for Oracle PKI SDK LDAP
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle PKI SDK LDAP

To use Oracle PKI SDK LDAP, your system must have the following:

- Java Development Kit (JDK) version 1.2.2 or higher. Also, make sure that the Java bin directory is added to your PATH environment variable.
- Sun Microsystem's Java Naming and Directory Interface (JNDI) version 1.2.1 or higher. You must add all of the JNDI jar files to your CLASSPATH.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar
- osdt_cert.jar
- osdt_ldap.jar
- jndi.jar, ldapbp.jar, ldap.jar, jaas.jar, and providerutil.jar (Sun’s Java Naming and Directory Interface (JNDI))

Setting the CLASSPATH on Windows

To set your CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all of the required jar and class files to the CLASSPATH. For example:
   
   ```
   C:\ORACLE_HOME\jlib\osdt_core.jar;C:\ORACLE_HOME\jlib\osdt_cert.jar;
   C:\ORACLE_HOME\jlib\osdt_ldap.jar;
   ```
6. Click OK.

Setting the CLASSPATH on Unix

On Unix, set your CLASSPATH environment variable to include the full path and file name of all the required jar and class files. For example:

```bash
setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
$ORACLE_HOME/jlib/osdt_cert.jar:
$ORACLE_HOME/jlib/osdt_ldap.jar
```
The Oracle PKI SDK LDAP Java API Reference

The Oracle PKI SDK LDAP Java API reference (Javadoc) is available at:

*Oracle Security Developer Tools PKI SDK LDAP Java API Reference*

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at

Extensible Markup Language (XML) is an application of Standard Generalized Markup Language (SGML). XML is a meta-language that allows implementors to define their own self-describing markup. Implementors use XML to define their own set of custom tags. The tags are similar to those found in an HTML document; like XML, HTML is also an application of SGML.

For a document to be valid, it must conform to all the constraints imposed by a given Document Type Definition (DTD) or schema. A valid XML document is said to be semantically correct.

XML security refers to standard security requirements of XML documents such as confidentiality, integrity, message authentication, and non-repudiation. The need for digital signature and encryption standards for XML documents prompted the World Wide Web Consortium (W3C) to put forth an XML Signature standard and an XML Encryption standard. The XML Signature standard is the product of a joint working group that also includes the Internet Engineering Task Force (IETF). In addition, the W3C and IETF have also jointly proposed an XML Key Management Specification (XKMS) that defines protocols for distributing and registering public keys associated with XML signatures and XML encryption.

This chapter describes key features and benefits of Oracle XML Security, and explains how to set up your environment to use Oracle XML Security.

This chapter contains these topics:

- Oracle XML Security Features and Benefits
- Setting Up Your Oracle XML Security Environment
- Classes and Interfaces
- Common XML Security Questions
- The Oracle XML Security Java API Reference
- Example Programs
Oracle XML Security Features and Benefits

The Oracle XML Security SDK is a pure Java solution which provides the following features:

- Support for the XML Signature standard
- Support for the XML Encryption standard
- Support for the Decryption Transform standard
- Support for the XML Canonicalization standard
- Support for the Exclusive XML Canonicalization standard
- Compatibility with a wide range of JAXP 1.1 compliant XML parsers and XSLT engines

Links to these standards are available in Appendix A, "References".

Oracle XML Security Features and Benefits

Setting Up Your Oracle XML Security Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME.

This section explains how to set up your environment for Oracle XML Security. It contains these topics:

- System Requirements for Oracle XML Security
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle XML Security

In order to use Oracle XML Security, your system must have the following components installed:

- The Java Development Kit (JDK) version 1.2.2 or higher
- A JAXP-compatible XML parser and XSLT processor

Oracle XML Security has been tested with the following implementations:

- Apache Xalan-Java (with Xerces-J)
- Oracle XDK for Java

See Also: The following resources provide more information about XML and XML standards:

- W3C’s Recommendation for XML 1.0
- JavaSoft’s XML FAQ
- O’Reilly’s XML Web site
- The Internet Engineering Task Force Web Site
- W3C’s Recommendation for XML Signatures
- W3C’s Recommendation for XML Encryption
- The proposed XML Key Management specification

Links to these resources are available in Appendix A, "References".

8-2 Oracle Security Developer Tools Reference
Setting Up Your Oracle XML Security Environment

Apache Libraries
Sun JDK 1.4.x distributions contain an embedded version of the Apache Crimson parser and an older version of the Apache Xalan XSLT engine. Oracle does not recommend using these versions, as they contain a number of bugs and incompatibilities that can result in signature and encryption failures. If you are using JDK 1.4.x with an Apache XML parser, the XSLT engine, or both, put the Apache library JAR files in your JRE's /lib/endorsed directory to override the JRE's built-in version of Apache.

Setting the CLASSPATH Environment Variable
Your CLASSPATH environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar
- osdt_cert.jar
- osdt_xmlsec.jar
- jaxen.jar, which is included in the $ORACLE_HOME/jlib directory of the security tools distribution. Oracle XML Security relies on the Jaxen XPath engine for XPath processing.
- The appropriate XML parser and XSLT processor implementations, unless you have installed them in your JRE's /lib/ext or /lib/endorsed directory.

Note: The Jaxen library included in the Oracle XML Security distribution is a modified version of the Jaxen 1.0 FCS release. If you also have an earlier Jaxen release in your CLASSPATH, you must ensure that the version from this distribution appears first.

Setting the CLASSPATH on Windows
If you are installing Oracle XML Security on Windows, set your CLASSPATH as follows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all the required jar and class files to the CLASSPATH.

For example, your CLASSPATH might look like this:

%CLASSPATH%;C:\ORACLE_HOME\jlib\osdt_core.jar;
6. Click OK.

Setting the CLASSPATH on UNIX
On UNIX, set your CLASSPATH environment variable to include the full path and file name of all the required jar and class files. For example:

```bash
setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
$ORACLE_HOME/jlib/osdt_cert.jar:
$ORACLE_HOME/jlib/osdt_xmlsec.jar:
$ORACLE_HOME/jlib/jaxen.jar:
```

Classes and Interfaces
This section describes classes in the XML Security API. It includes:

- **Core Classes**
- **Supporting Classes and Interfaces**

Core Classes
This section describes core classes, illustrates how to create class instances, and uses code samples to illustrate the capabilities of each class.

**The oracle.security.xmlsec.dsig.XSSignature Class**
This class represents the top-level Signature element of the XML Signature schema. Creating an instance of this class is the first step in creating a new signature or in verifying an existing signature.

To create a new signature, you create a new instance of the XSSignature class by calling the static newInstance() method:

**Example 8–1 Creating a Signature with XSSignature**

```java
XSSignature sig = XSSignature.newInstance("MySignatureID");
```

To obtain Signature elements from an XML document to verify a signature, you first obtain an org.w3c.dom.NodeList object that contains all the Signature elements as instances of org.w3c.dom.Node. You can then iterate through the NodeList and convert each node to an instance of XSSignature, as the following example illustrates:

**Example 8–2 Verifying a Signature with XSSignature**

```java
Document doc = Instance of org.w3c.dom.Document;
NodeList sigList = doc.getElementsByTagNameNS(XMLURI.ns_xmldsig, "Signature");
if (sigList.getLength() == 0)
    System.err.println("No XML-DSIG Signature elements found.");
for (int s = 0, n = sigList.getLength(); s < n; ++s)
    // Convert each org.w3c.dom.Node object to a oracle.security.xmlsec.dsig.XSSignature
    // object and perform verification
```
The oracle.security.xmlsec.dsig.XSSignedInfo Class

This class represents the SignedInfo element of the XML Signature schema. As with XSSignature, you must use this class to both create and verify signatures. In signature creation, you create an instance of this class with the following code:

Example 8–3 Creating a Signature with XSSignedInfo

```java
XSignature sig = XSSignature.newInstance("MySignatureID");
XSSignedInfo si = sig.createSignedInfo("MySignedInfoID");
```

When performing verification, you first obtain an instance of XSSignature as shown in Example 8–2, then obtain the SignedInfo element from the top-level Signature with the following code:

Example 8–4 Verifying a Signature with XSSignedInfo

```java
XSignature sig;
//Instance of XSSignature is obtained (Example 8-2)
//Get SignedInfo
XSSignedInfo si = sig.getSignedInfo();
```

The oracle.security.xmlsec.dsig.XSReference class

This class represents the Reference element of the XML Signature schema. You must use this class when creating and verifying signatures. In signature creation, you create an instance of this class with the following code:

Example 8–5 Creating Signature Reference Elements with XSReference

```java
XSignature sig = XSSignature.newInstance("MySignatureID");
String uri = "the URI of the data object you want to reference";
String type = "the type of the data object you want to reference (optional)";
XSAlgorithmIdentifier digestAlg =
    the digest algorithm identifier (e.g., XMLURI.alg_sha1);
XSReference ref =
    sig.createReference("MyReferenceID", uri, type, digestAlg);
```

When performing verification, you first obtain an instance of XSSignature as shown in Example 8–2, then obtain the Reference elements from the top-level Signature with the following code:

Example 8–6 Obtaining Reference Elements of XSSignature

```java
XSignature sig;
//Instance of XSSignature is obtained (Example 8-2)
//Get Vector of reference objects
Vector refs = sig.References();
```
The oracle.security.xmlsec.dsig.XSKeyInfo class
This class represents the KeyInfo element of the XML Signature schema. You may use this class for signature creation as well as signature verification.

In signature creation, you create an instance of this class with the following code:

Example 8–7 Creating Key Information Elements with XSKeyInfo
XSignature sig = XSSignature.newInstance("MySignatureID");
XSKeyInfo si = sig.createKeyInfo("MyKeyInfoID");

A KeyInfo element can have various child elements that contain the actual key data. The classes that support these KeyInfo children are found in the oracle.security.xmlsec.keys package.

For example, to create an RSAKeyValue element containing a signer’s public key, you can use the following code:

Example 8–8 Creating an RSAKeyValue Element with the Signer’s Public Key
X509 cert = An instance of the oracle.security.crypto.cert.X509 class;
XSKeyInfo ki = An instance of the XSKeyInfo class;
RSAKeyValue rsaKeyValue = ki.createKeyValue(cert.getPublicKey());
ki.addKeyInfoData(rsaKeyValue);

When performing verification, you first obtain an instance of XSSignature as shown in Example 8–2, then obtain the KeyInfo element from the top-level Signature with the following code:

Example 8–9 Obtaining KeyInfo Elements of XSSignature
XSSignature sig;

//Instance of XSSignature is obtained (Example 8-2)

XSSignedInfo si = sig.getKeyInfo();

The oracle.security.xmlsec.enc.XEEncryptedData class
This class represents the EncryptedData element of the XML encryption schema. You must create an instance of this class when encrypting or decrypting arbitrary data or an entire XML document.

When encrypting, you create an instance of this class with the following code:

Example 8–10 Using XEEncryptedData for Encryption
Document doc = Instance of org.w3c.dom.Document;
String dataType = Either XMLURI.obj_content (content only) or XMLURI.obj_Element (entire element);
XEEncryptedData encData =
XEEncryptedData.newInstance(doc, "MyEncryptedDataID", dataType);

When decrypting, you can obtain the EncryptedData elements from an XML document with the following code:
Example 8–11 Using XEEncryptedData for Decryption

Document doc = Instance of org.w3c.dom.Document;

// Get list of all XML EncryptedData elements in the document.
NodeList encDataList =
    doc.getElementsByTagNameNS(XMLURI.ns_xmlenc, "EncryptedData");
if (encDataList.getLength() == 0)
    System.err.println("No XML-ENC EncryptedData elements found.");

// Convert each org.w3c.dom.Node object to a
// oracle.security.xmlsec.enc.XEEncryptedData
// object and perform decryption
for (int s = 0, n = encDataList.getLength(); s < n; ++s)
{
    XEEncryptedData = new XEEncryptedData((Element)encDataList.item(s));

    //TODO: Perform decryption of the encrypted data
    // contained in this element
}

The oracle.security.xmlsec.enc.XEEncryptedKey Class

This class represents the EncryptedKey element of the XML Encryption Schema. You can use an instance of this class to encrypt and decrypt cryptographic key material.

When encrypting a key, you create an instance of this class with the following code:

Example 8–12 Using XEEncryptedKey for Key Encryption

Document doc = Instance of org.w3c.dom.Document;
XEEncryptedKey encKey = XEEncryptedKey.newInstance(doc, "MyEncryptedKeyID");

When decrypting a key, you first obtain the XEEncryptedData from an XML document using the code in Example 8–11, then obtain the EncryptedKey elements with the following code:

Example 8–13 Using XEEncryptedKey for Key Decryption

XEEncryptedData encData;
//Instance of XEEncryptedData is obtained (See Example 8-11

//Get Vector of XEEncryptedKey objects
XEKeyInfo ki = encData.getKeyInfo();
Vector encKeys;
if (ki != null)
    Vector encKeys = encData.getEncryptedKeys();

The oracle.security.xmlsec.enc.XEEncryptionMethod Class

This class represents the EncryptionMethod element of the XML encryption schema. It contains the algorithm and parameters used in encrypting data or encrypting a key.

When encrypting, you create an instance of this class with the following code:

Example 8–14 Using XEEncryptionMethod for Encryption

String algURI = "String containing the URI of the encryption algorithm";
XEEncryptedObject encObj = Instance of XEEncryptedData or XEEncryptedKey;
XEEncryptionMethod em = encObj.createEncryptionMethod(algURI);
When decrypting, you first obtain an EncryptedData element using Example 8–11, or an EncryptedKey element using Example 8–13, then obtain an EncryptionMethod element with the following code:

**Example 8–15 Using XEEncryptionMethod for Decryption**
XEEncryptedObject encObj;

//Obtain instance of XEEncryptedData (see class example earlier) or //XEEncryptedKey (see class example earlier)
XEEncryptionMethod em = encObj.getEncryptionMethod();

**The oracle.security.xmlsec.enc.XECipherData Class**
This class represents the CipherData element that provides the encrypted data. It either stores the encrypted data in the CipherValue element or refers to a source containing the data through the CipherReference element. When performing encryption, you create an instance of XEEncryptedData or XEEncryptedKey, then create an instance of XECipherData with the following code:

**Example 8–16 Using XECipherData when Encrypting**
XEEncryptedObject encObj;

//Create an instance of XEEncryptedData (see class example earlier) //XEEncryptedKey (see example 8-12)
XECipherData cd = encObj.createCipherData();

When decrypting, you first obtain an EncryptedData element using Example 8–11, or an EncryptedKey element using Example 8–13, then obtain an instance of an XECipherData element with the following code:

**Example 8–17 Using XECipherData when Decrypting**
XEEncryptedObject encObj;

//Obtain an instance of XEEncryptedData (see example 8-11) or //XEEncryptedKey (see example 8-13)
XECipherData cd = encObj.getCipherData();

**Supporting Classes and Interfaces**
This section describes additional classes and interfaces in the Oracle XML Security SDK.

**The oracle.security.xmlsec.util.XMLURI Interface**
This interface defines URI string constants for algorithms, namespaces, and objects. It uses the following naming convention:

- Algorithm URIs begin with "alg_".
- Namespace URIs begin with "ns_".
- Object type URIs begin with "obj_".
The oracle.security.xmlsec.util.XMLUtils class
This class contains static utility methods for XML and XML-DSIG. Methods frequently used in applications include the createDocBuilder(), createDocument(), toBytesXML(), and toStringXML() methods.

Common XML Security Questions
This section answers frequently asked questions about XML security and about using Oracle XML Security. It addresses these areas:

- Common Questions about Keys and Certificates
- Common Questions about XML Signatures
- Common Questions about XML Encryption

Common Questions about Keys and Certificates
This section describes common issues related to keys and certificates.

What is the DER format? The PEM format? How are these formats used?
DER is an abbreviation for ASN.1 Distinguished Encoding Rules. DER is a binary format that is used to encode certificates and private keys. Oracle XML Security SDK uses DER as its native format, as do most commercial products that use certificates and private keys.

Many other formats used to encode certificates and private keys, including PEM, PKCS #7, and PKCS #12, are transformations of DER encoding. For example, PEM (Privacy Enhanced Mail) is a text format that is the Base 64 encoding of the DER binary format. The PEM format also specifies the use of text BEGIN and END lines that indicate the type of content that is being encoded.

I received a certificate in my email in a text format. It has several lines of text characters that don't seem to mean anything. How do I convert it into the format that Oracle XML Security uses?
If you received the certificate in your email, it is in PEM format. You need to convert the certificate from PEM (Privacy-Enhanced Mail) format to ASN.1 DER (Distinguished Encoding Rules) format.

How do I use a certificate that is exported from a browser?
If you have exported the certificate from a browser, it is most likely in PKCS #12 format (*.p12 or *.pfx). You must parse the PKCS #12 object into its component parts.

Common Questions about XML Signatures
This section describes common questions about keys and certificates.

What signature algorithms does Oracle XML Security support?
Oracle XML Security supports the following signature algorithms:

- DSA with SHA1
- RSA with SHA1

See Also: For more information about these algorithms, refer to the links for DSA-SHA and RSA-SHA in Appendix A, "References".
Common Questions about XML Encryption

This section describes common issues related to keys and certificates.

**What data encryption algorithms does Oracle XML Security support?**
Oracle XML Security supports the following signature algorithms:

- AES-128 in CBC mode
- AES-192 in CBC mode
- AES-256 in CBC mode
- DES EDE in CBC mode

Links to these standards are available in Appendix A, "References".

**What key wrapping algorithms does Oracle XML Security support?**
Oracle XML Security supports the following key wrapping algorithms:

- AES-128
- AES-192
- AES-256
- DES-EDE

Links to these standards are available in Appendix A, "References".

**What key transport algorithms does Oracle XML Security support?**
Oracle XML Security supports the following key transport algorithms:

- RSAES-OAEP-ENCRYPT with MGF1
- RSAES-PKCS1-v1_5

Links to these standards are available in Appendix A, "References".

**What key agreement algorithms does Oracle XML Security support?**
Oracle XML Security supports the Diffie-Hellman key agreement algorithm.

*See Also:* A link to this standard is available in Appendix A, "References".

The Oracle XML Security Java API Reference

The Oracle XML Security API (Javadoc) is available at:

*Oracle Security Developer Tools XML Security Java API Reference*

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at

This chapter provides information about using the Oracle Security Assertions Markup Language (SAML) Software Development Kit (SDK). Oracle SAML allows Java developers to develop cross-domain single sign-on and federated access control solutions that conform to the SAML 1.0/1.1 and SAML 2.0 specifications.

This chapter contains the following topics:

- Oracle SAML Features and Benefits
- Oracle SAML 1.0/1.1
- Oracle SAML 2.0

Oracle SAML Features and Benefits

The Oracle SAML SDK provides a Java API with supporting tools, documentation, and sample programs to assist developers of SAML-compliant Java security services. Oracle SAML can be integrated into existing Java solutions, including applets, applications, EJBs, servlets, and JSPs.

Oracle SAML provides the following features:

- Support for the SAML 1.0/1.1 and 2.0 specifications
- Support for SAML-based single sign-on (SSO), Attribute, Metadata, Enhanced Client Proxy, and federated identity profiles

See Also: For more information and links to these specifications and related documents, see Appendix A, "References”.

Oracle SAML 1.0/1.1

This section explains how to set up your environment for Oracle SAML 1.0/1.1, how to use Oracle SAML 1.0/1.1, and the classes and interfaces of the Oracle SAML 1.0/1.1 toolkit. It contains the following topics:

- Oracle SAML 1.0/1.1 Packages
- Setting Up Your Oracle SAML 1.0/1.1 Environment
- Classes and Interfaces
- The Oracle SAML 1.0/1.1 Java API Reference
- Example Programs
Oracle SAML 1.0/1.1 Packages

The Oracle SAML Java API contains the following packages for creating SAML 1.0/1.1-compliant Java applications:

**oracle.security.xmlsec.saml**
This package contains classes that support SAML assertions.

**oracle.security.xmlsec.samlp**
This package contains classes that support the SAML request and response protocol (SAMLP).

Setting Up Your Oracle SAML 1.0/1.1 Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME. This section explains how to set up your environment for Oracle SAML 1.0/1.1. It contains these topics:

- System Requirements for Oracle SAML 1.0/1.1
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle SAML 1.0/1.1

In order to use Oracle SAML, your system must have the Java Development Kit (JDK) version 1.2.2 or higher.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar
- osdt_cert.jar
- osdt_xmlsec.jar
- osdt_saml.jar
- The jaxen.jar file (Jaxen XPath engine, included with your Oracle XML Security distribution)
- The jar files for your chosen XML parser and XSLT processor (for example, xalan.jar and xercesImpl.jar if using Apache Xalan-Java)

Setting the CLASSPATH on Windows

To set the CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all the required jar files to the CLASSPATH.
For example, your CLASSPATH might look like this:

```
%CLASSPATH%;C:\ORACLE_HOME\jlib\osdt_core.jar;
C:\ORACLE_HOME\jlib\osdt_cert.jar;
C:\ORACLE_HOME\jlib\osdt_xmlsec.jar;
C:\ORACLE_HOME\jlib\osdt_saml.jar;
C:\ORACLE_HOME\jlib\jaxen.jar;
C:\xalan-j_2_6_0\bin\xalan.jar:C:\xalan-j_2_6_0\bin\xercesImpl.jar
```

6. Click OK.

**Setting the CLASSPATH on UNIX** On UNIX, set your CLASSPATH environment variable to include the full path and file name of all the required jar and class files. For example:

```
setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
$ORACLE_HOME/jlib/osdt_cert.jar:
$ORACLE_HOME/jlib/osdt_xmlsec.jar:
$ORACLE_HOME/jlib/osdt_saml.jar:
$ORACLE_HOME/jlib/jaxen.jar:
/usr/lib/xalan-j_2_6_0/bin/xalan.jar:/usr/lib/xalan-j_2_6_0/bin/xercesImpl.jar
```

### Classes and Interfaces

This section provides information and code samples for using the classes and interfaces of Oracle SAML 1.0/1.1. It contains these topics:

- **Core Classes**
- **Supporting Classes and Interfaces**

#### Core Classes

This section provides a brief overview of the core SAML and SAMLP 1.0/1.1 classes with some brief code examples.

The core classes are:

- **The oracle.security.xmlsec.saml.SAMLInitializer Class**
- **The oracle.security.xmlsec.saml.Assertion Class**
- **The oracle.security.xmlsec.samlp.Request Class**
- **The oracle.security.xmlsec.samlp.Response Class**

**The oracle.security.xmlsec.saml.SAMLInitializer Class** This class initializes the Oracle SAML toolkit. By default Oracle SAML is automatically initialized for SAML v1.0. You can also initialize Oracle SAML for a specific version of the SAML specification. When the initialize method is called for a specific version, previously initialized versions will remain initialized. Example 9–1 shows how to initialize the SAML toolkit for SAML v1.0 and SAML v1.1.

**Example 9–1 Initializing the Oracle SAML Toolkit**

```
// initializes for SAML v1.1
SAMLInitializer.initialize(1, 1);
// initializes for SAML v1.0, done by default
SAMLInitializer.initialize(1, 0);
```
The oracle.security.xmlsec.saml.Assertion Class  This class represents the Assertion element of the SAML Assertion schema.

Example 9–2 shows how to create a new Assertion element and append it to an existing XML document.

Example 9–2  Creating an Assertion Element and Appending to an XML Document

Document doc = Instance of org.w3c.dom.Document;
Assertion assertion = new Assertion(doc);
doc.getDocumentElement().appendChild(assertion);

Example 9–3 shows how to obtain Assertion elements from an XML document.

Example 9–3  Obtaining Assertion Elements From an XML Document

Document doc = Instance of org.w3c.dom.Document;

// Get a list of all Assertion elements in the document

NodeList assertList =
    doc.getElementsByTagNameNS(SAMLURI.ns_saml, "Assertion");
if (assertList.getLength() == 0)
    System.err.println("No Assertion elements found.");

// Convert each org.w3c.dom.Node object to a
// oracle.security.xmlsec.saml.Assertion object and process

for (int s = 0, n = assertList.getLength(); s < n; ++s)
{
    Assertion assertion = new Assertion((Element)assertList.item(s));

    // Process Assertion element

    ...
}

The oracle.security.xmlsec.samlp.Request Class  This class represents the Request element of the SAML Protocol schema.

Example 9–4 shows how to create a new Request element and append it to an existing XML document.

Example 9–4  Creating a Request Element and Appending to an XML Document

Document doc = Instance of org.w3c.dom.Document;
Request request = new Request(doc);
doc.getDocumentElement().appendChild(request);

Example 9–5 shows how to obtain Request elements from an existing XML document.

Example 9–5  Obtaining Request Elements From an XML Document

Document doc = Instance of org.w3c.dom.Document;

// Get a list of all Request elements in the document

NodeList reqList =
    doc.getElementsByTagNameNS(SAMLURI.ns_samlp, "Request");
if (reqList.getLength() == 0)
    System.err.println("No Request elements found.");
for (int s = 0, n = reqList.getLength(); s < n; ++s) {
    Request request = new Request((Element)reqList.item(s));
    // Process Request element
    ...
}

The oracle.security.xmlsec.samlp.Response Class This class represents the Response element of the SAML Protocol schema.

Example 9–6 shows how to create a Response element and append it to an existing XML document.

Example 9–6 Creating a Response Element and Appending to an XML Document

```
Document doc = Instance of org.w3c.dom.Document;
Response response = new Response(doc);
doc.getDocumentElement().appendChild(response);
```

Example 9–7 shows how to obtain Response elements from an existing XML document.

Example 9–7 Obtaining Response Elements From an XML Document

```
Document doc = Instance of org.w3c.dom.Document;

// Get a list of all Response elements in the document
NodeList respList =
    doc.getElementsByTagNameNS(SAMLURI.ns_samlp, "Response");
if (respList.getLength() == 0)
    System.err.println("No Response elements found.");

// Convert each org.w3c.dom.Node object to a
// oracle.security.xmlsec.samlp.Response object and process
for (int s = 0, n = respList.getLength(); s < n; ++s) {
    Response response = new Response((Element)respList.item(s));
    // Process Response element
    ...
}
```

Supporting Classes and Interfaces

This section provides an overview of the supporting classes and interfaces of Oracle SAML 1.0/1.1:

- The oracle.security.xmlsec.saml.SAMLURI Interface
- The oracle.security.xmlsec.saml.SAMLMessage Class

The oracle.security.xmlsec.saml.SAMLURI Interface This interface defines URI string constants for algorithms, namespaces, and objects. The following naming conventions are used:
- Action Namespace URIs defined in the SAML 1.0 specifications begin with `action_`
- Authentication Method Namespace URIs defined in the SAML 1.0 specifications begin with `authentication_method_`
- Confirmation Method Namespace URIs defined in the SAML 1.0 specifications begin with `confirmation_method_`
- Namespace URIs begin with `ns_`

**The oracle.security.xmlsec.saml.SAMLMessage Class** This is the base class for all the SAML and SAML extension messages that may be signed and contain an XML-DSIG (digital signature) structure.

**The Oracle SAML 1.0/1.1 Java API Reference**
The Oracle SAML 1.0/1.1 Java API reference (Javadoc) is available at: *Oracle Security Developer Tools SAML 1.0/1.1 Java API Reference*

**Example Programs**

**Oracle SAML 2.0**
This section explains how to set up your environment for Oracle SAML 2.0, how to use Oracle SAML 2.0, and the classes and interfaces of the Oracle SAML 2.0 toolkit. It contains the following topics:

- **Oracle SAML 2.0 Packages**
- **Setting Up Your Oracle SAML 2.0 Environment**
- **Classes and Interfaces**
- **The Oracle SAML 2.0 Java API Reference**
- **Example Programs**

**Oracle SAML 2.0 Packages**
The Oracle SAML Java API contains the following packages for creating SAML 2.0-compliant Java applications:

- **oracle.security.xmlsec.saml2.core**
  This package contains classes that support SAML assertions.

- **oracle.security.xmlsec.saml2.protocol**
  This package contains classes that support the SAML request and response protocol (SAMLP).

- **oracle.security.xmlsec.saml2.ac**
  This package contains classes that support the SAML authentication context basic types.
oracle.security.xmlsec.saml2.ac.classes
This package contains classes that support various SAML authentication context classes.

oracle.security.xmlsec.saml2.metadata
This package contains classes that support the SAML metadata.

oracle.security.xmlsec.saml2.profiles.attributes
This package contains classes that support various SAML attribute profiles.

oracle.security.xmlsec.saml2.profiles.sso.ecp
This package contains classes that support the SAML ECP SSO profile.

Setting Up Your Oracle SAML 2.0 Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME.

This section explains how to set up your environment for Oracle SAML 2.0. It contains these topics:

- System Requirements for Oracle SAML 2.0
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle SAML 2.0

In order to use Oracle SAML, your system must have the Java Development Kit (JDK) version 1.2.2 or higher.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names of all the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar
- osdt_cert.jar
- osdt_xmlsec.jar
- osdt_saml2.jar
- The jaxen.jar file (Jaxen XPath engine, included with your Oracle XML Security distribution)
- The jar files for your chosen XML parser and XSLT processor (for example, xalan.jar and xercesImpl.jar if using Apache Xalan-Java)

Setting the CLASSPATH on Windows

To set the CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all of the required jar files to the CLASSPATH.

For example, your CLASSPATH might look like this:

```
%CLASSPATH%;C:\ORACLE_HOME\jlib\osdt_core.jar;
C:\ORACLE_HOME\jlib\osdt_cert.jar;
C:\ORACLE_HOME\jlib\osdt_xmlsec.jar;
C:\ORACLE_HOME\jlib\osdt_saml2.jar;
C:\ORACLE_HOME\jlib\jaxen.jar;
C:\xalan-j_2_6_0\bin\xalan.jar;C:\xalan-j_2_6_0\bin\xercesImpl.jar
```

6. Click OK.

**Setting the CLASSPATH on UNIX**

On UNIX, set your CLASSPATH environment variable to include the full path and file name of all of the required jar and class files. For example:

```
setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
$ORACLE_HOME/jlib/osdt_cert.jar:
$ORACLE_HOME/jlib/osdt_xmlsec.jar:
$ORACLE_HOME/jlib/osdt_saml2.jar:
$ORACLE_HOME/jlib/jaxen.jar:
/usr/lib/xalan-j_2_6_0/bin/xalan.jar:/usr/lib/xalan-j_2_6_0/bin/xercesImpl.jar
```

### Classes and Interfaces

This section provides information and code samples for using the classes and interfaces of Oracle SAML 2.0. It contains these sections:

- Core Classes
- Supporting Classes and Interfaces

#### Core Classes

This section provides an overview of the core SAML and SAMLP classes with some brief code examples. The core classes are:

- The oracle.security.xmlsec.saml2.core.Assertion Class
- The oracle.security.xmlsec.saml2.protocol.AuthnRequest Class
- The oracle.security.xmlsec.saml2.protocol.StatusResponse Class

**The oracle.security.xmlsec.saml2.core.Assertion Class**

This class represents the Assertion element of the SAML Assertion schema.

**Example 9–8 Creating an Assertion Element and Appending it to an XML Document**

```java
Document doc = Instance of org.w3c.dom.Document;
Assertion assertion = new Assertion(doc);
doc.getDocumentElement().appendChild(assertion);
```

**Example 9–9 Obtaining Assertion Elements From an XML Document**

```java
// Get a list of all Assertion elements in the document
```
NodeList assertList =
    doc.getElementsByTagNameNS(SAML2URI.ns_saml, "Assertion");
if (assertList.getLength() == 0)
    System.err.println("No Assertion elements found.");

  // Convert each org.w3c.dom.Node object to a
  // oracle.security.xmlsec.saml2.core.Assertion object and process
  for (int s = 0, n = assertList.getLength(); s < n; ++s)
  {
    Assertion assertion = new Assertion((Element)assertList.item(s));
    // Process Assertion element
    ...
  }

The oracle.security.xmlsec.saml2.protocol.AuthnRequest Class  
This class represents the AuthnRequest element of the SAML Protocol schema. 
Example 9–10 shows how to create a new AuthnRequest element and append it to an existing XML document.

Example 9–10  Creating an AuthnRequest Element and Appending it to an XML Document

Document doc = Instance of org.w3c.dom.Document;
AuthnRequest request = new AuthnRequest(doc);
doc.getDocumentElement().appendChild(response);

Example 9–11 shows how to obtain AuthnRequest elements from an existing XML document.

Example 9–11  Obtaining AuthnRequest Elements From an XML Document

Document doc = Instance of org.w3c.dom.Document;

  // Get a list of all AuthnRequest elements in the document
  NodeList reqList =
    doc.getElementsByTagNameNS(SAML2URI.ns_samlp, "AuthnRequest");
if (reqList.getLength() == 0)
    System.err.println("No Request elements found.");

  // Convert each org.w3c.dom.Node object to a
  // oracle.security.xmlsec.saml2.protocol.AuthnRequest
  // object and process
  for (int s = 0, n = reqList.getLength(); s < n; ++s)
  {
    AuthnRequest request = new AuthnRequest((Element)reqList.item(s));
    // Process Request element
    ...
  }

The oracle.security.xmlsec.saml2.protocol.StatusResponse Class  
This class represents the StatusResponse element of the SAML Protocol schema. 
Example 9–12 shows how to create a StatusResponse element and append it to an existing XML document.
**Example 9–12  Creating a StatusResponse Element and Appending to an XML Document**

```java
Document doc = Instance of org.w3c.dom.Document;
StatusResponse response = new StatusResponse(doc);
doc.getDocumentElement().appendChild(response);
```

Example 9–13 shows how to obtain StatusResponse elements from an existing XML document.

**Example 9–13  Creating a StatusResponse Element and Appending it to an XML Document**

```java
Document doc = Instance of org.w3c.dom.Document;
// Get a list of all Response elements in the document
NodeList respList =
    doc.getElementsByTagNameNS(SAML2URI.ns_samlp, "StatusResponse");
if (respList.getLength() == 0)
    System.err.println("No Response elements found.");

// Convert each org.w3c.dom.Node object to a
// oracle.security.xmlsec.saml2.protocol.StatusResponse object and process
for (int s = 0, n = respList.getLength(); s < n; ++s)
{
    StatusResponse response = new StatusResponse((Element)respList.item(s));
    // Process StatusResponse element
    ...
}
```

**Supporting Classes and Interfaces**

This section provides an overview of the supporting classes and interfaces of Oracle SAML 2.0. It includes:

- The oracle.security.xmlsec.saml2.util.SAML2URI Interface

The oracle.security.xmlsec.saml2.util.SAML2URI Interface This interface defines URI string constants for algorithms, namespaces, and objects. The interface uses these naming conventions:

- Action namespace URIs defined in the SAML 1.0/1.1/2.0 specifications begin with action_
- Authentication method namespace URIs defined in the SAML 1.0/1.1/2.0 specifications begin with authentication_method_
- Confirmation method namespace URIs defined in the SAML 1.0/1.1/2.0 specifications begin with confirmation_method_
- Namespace URIs begin with ns_

**The Oracle SAML 2.0 Java API Reference**

The Oracle SAML Java API reference (Javadoc) is available at:

*Oracle Security Developer Tools SAML 1.0/1.1 Java API Reference*
Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at

Oracle Web Services Security

Oracle Web Services Security provides a framework of authorization and authentication for interacting with a web service using XML-based messages. This chapter provides information about key features and benefits of Oracle Web Services Security, and describes how to install and use the SDK.

This chapter contains these topics:

- Oracle Web Services Security Features and Benefits
- Setting Up Your Oracle Web Services Security Environment
- Classes and Interfaces
- The Oracle Web Services Security Java API Reference
- Example Programs

**Oracle Web Services Security Features and Benefits**

Oracle Web Services Security is a pure Java solution which provides the following features:

- Support for the SOAP Message Security standard
- Support for the Username Token Profile standard
- Support for the X.509 Certificate Token Profile standard
- Support for the SAML Assertion Token standard

**Oracle Web Services Security Packages**

The Oracle Web Services Security library contains the following packages:

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oracle.security.xmlsec.wss</td>
<td>Contains general-purpose Oracle Web Services Security classes, including interfaces for token and reference creation and validation</td>
</tr>
<tr>
<td>oracle.security.xmlsec.wss.encoding</td>
<td>Contains classes for encoding and decoding algorithms required to support Web Services processing</td>
</tr>
<tr>
<td>oracle.security.xmlsec.wss.saml</td>
<td>Contains core classes supporting SAML assertion tokens</td>
</tr>
<tr>
<td>oracle.security.xmlsec.wss.soap</td>
<td>Contains core classes supporting the creation and parsing of SOAP messages with WSS security headers</td>
</tr>
</tbody>
</table>
Setting Up Your Oracle Web Services Security Environment

This section explains how to set up your environment for Oracle Web Services Security. It contains these topics:

- System Requirements for Oracle Web Services Security
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle Web Services Security

In order to use Oracle Web Services Security, you must have the following components:

- Java Development Kit (JDK) version 1.2.2 or higher
- A JAXP-compatible XML parser and XSLT processor.

Oracle Web Services Security has been tested with the following implementations:

- Apache Xalan-Java (with Xerces-J)
- Oracle XDK for Java

For questions regarding compatibility with other parsers, visit http://www.oracle.com/technology/documentation.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar

Table 10–1  (Cont.) Packages in the Oracle Web Services Security Library

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oracle.security.xmlsec.wss.transforms</td>
<td>Contains classes implementing the transformation algorithms defined in Oracle Web Services Security</td>
</tr>
<tr>
<td>oracle.security.xmlsec.wss.username</td>
<td>Contains classes supporting the creation and parsing of username tokens</td>
</tr>
<tr>
<td>oracle.security.xmlsec.soap</td>
<td>Contains SOAP utility classes</td>
</tr>
<tr>
<td>oracle.security.xmlsec.wss.x509</td>
<td>Contains core classes supporting X.509 certificate tokens</td>
</tr>
<tr>
<td>oracle.security.xmlsec.wss.utils</td>
<td>Contains Oracle Web Services Security utility classes</td>
</tr>
</tbody>
</table>

Related Documentation

The following resources provide more information about Web Services Security:

- OASIS WSS SOAP Message Security Specification
- OASIS WSS Username Token Profile Specification
- OASIS WSS X.509 Certificate Token Profile Specification
- OASIS WSS SAML Assertion Token Profile Specification

See Also: Links to these documents are available in Appendix A, "References".
Setting Up Your Oracle Web Services Security Environment

- osdt_cert.jar
- osdt_xmlsec.jar
- osdt_saml.jar
- The jaxen.jar file (Jaxen XPath engine, included with your Oracle XML Security distribution)

Note: Oracle XML Security relies on the Jaxen XPath engine for XPath processing. Note that the Jaxen library included in this distribution is a modified version of the Jaxen 1.0 FCS release. If your CLASSPATH also includes an earlier Jaxen release, you must ensure that the Oracle XML Security version appears first.

- osdt_wss.jar
- The appropriate XML parser and XSLT processor implementations, unless you have installed them in your JRE's /lib/ext or /lib/endorsed directory

Setting the CLASSPATH on Windows

To set the CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all of the required jar files to the CLASSPATH. For example, your CLASSPATH might look like this:

   %CLASSPATH%;C:\ORACLE_HOME\jlib\osdt_core.jar;
   C:\ORACLE_HOME\jlib\osdt_cert.jar;
   C:\ORACLE_HOME\jlib\osdt_xmlsec.jar;
   C:\ORACLE_HOME\jlib\osdt_saml.jar;
   C:\ORACLE_HOME\jlib\jaxen.jar;
   C:\ORACLE_HOME\jlib\osdt_wss.jar;

6. Click OK.

Setting the CLASSPATH on UNIX

On UNIX, set your CLASSPATH environment variable to include the full path and file name of all of the required jar and class files. For example:

    setenv CLASSPATH $CLASSPATH:ORACLE_HOME/jlib/osdt_core.jar:
    ORACLE_HOME/jlib/osdt_cert.jar:
    ORACLE_HOME/jlib/osdt_xmlsec.jar:
    ORACLE_HOME/jlib/osdt_saml.jar:
    ORACLE_HOME/jlib/jaxen.jar:
    ORACLE_HOME/jlib/osdt_wss.jar:
Classes and Interfaces

This section describes classes and interfaces in the Oracle Web Services Security API. It contains these topics:

- Core Classes and Interfaces
- Supporting Classes and Interfaces

Core Classes and Interfaces

This section describes the core classes in the Oracle Web Services Security API and provides examples of their use.

The oracle.security.xmlsec.wss.WSSecurity Class

The oracle.security.xmlsec.wss.WSSecurity class represents the top-level security element of the WSS SOAP Message Security schema. Creating an instance of this class is the first step in creating a new security header or in validating an existing security header.

To create a new security header, you create a new instance of the WSSecurity class by calling the static newInstance() method:

```
WSSecurity sig = WSSecurity.newInstance("MySecurityHeaderID");
```

Example 10–1 shows how to obtain security elements from an XML document in order to perform security processing:

1. Obtain an `org.w3c.dom.NodeList` object that contains all the security elements as instances of `org.w3c.dom.Node`.
2. Iterate through the `NodeList` and convert each node to an instance of `WSSecurity`.

```
Example 10–1 Obtaining Security Elements from an XML Document

Document doc = Instance of org.w3c.dom.Document;

// Get list of all WSS Security elements in the document.
NodeList secList =
    doc.getElementsByTagNameNS(WSSURI.ns_wsse, "Security");
if (secList.getLength() == 0)
    System.err.println("No wsse:Security elements found.");

// Convert each org.w3c.dom.Node object to an
// oracle.security.xmlsec.wss.WSSecurity object and perform verification
for (int s = 0, n = secList.getLength(); s < n; ++s)
{
    WSSecurity sec = new WSSecurity((Element)secList.item(s));
    
    // Process the wsse:Security header
    ... 
}
```

The oracle.security.xmlsec.wss.soap.WSSOAPEnvelope Class

The oracle.security.xmlsec.wss.soap.WSSOAPEnvelope class represents the SOAP message. As with WSSecurity, you must use this class to create SOAP messages as well as for parsing and validation.
To create a SOAP message, you can create an instance of this class with the code shown in Example 10–2:

**Example 10–2  Creating a SOAP Envelope**

```java
WSSOAPEnvelope env =
    new WSSOAPEnvelope.newInstance(XMLUtils.createDocBuilder());
WSecurity mySecHdr ......
env.addSecurity(mySecHdr);
```

When processing the message, you can obtain the Security element from the top-level SOAP message with the code shown in Example 10–3:

**Example 10–3  Obtaining the Security Element for a SOAP Message**

```java
WSSOAPEnvelope env;

//Get List of Security headers
ArrayList l = (ArrayList)env.getSecurity(null, false);
WSecurity sec = (WSecurity)l.get(0);
//Get List of Encrypted Keys
ArrayList r = (ArrayList) sec.getEncryptedKeys();
XEEncryptedKey xk = (XEEncryptedKey) r.get(0);
//Decrypt and Replace message contents
PrivateKey pk .... // Decryption Key
sec.decrypt (xk, pk);
```

**The oracle.security.xmlsec.wss.WSSElement Class**

`oracle.security.xmlsec.wss.WSSElement` is the base class for WSS Security elements. It supports reference elements with `local Id` and `wsu:Id` attributes for referencing them. All WSS schema elements, including tokens, extend this element.

**Supporting Classes and Interfaces**

This section describes supporting classes and interfaces in the Oracle Web Services Security API.

**The oracle.security.xmlsec.wss.utils.WSSURI Interface**

The `oracle.security.xmlsec.wss.utils.WSSURI` interface defines URI string constants for algorithms, namespaces, and objects.

**The oracle.security.xmlsec.wss.utils.WSSTokenUtils Class**

The `oracle.security.xmlsec.wss.utils.WSSTokenUtils` class contains static utility methods for WSS security token. Some of the methods that may be frequently used in an application include:

- `createSecurityToken()`
- `createSecurityTokenReference()`
- `createUsernameToken()`
- `createBinarySecurityToken()`
- `createBinarySecurityEncoder()`
- `createTimestamp()`
The `oracle.security.xmlsec.wss.utils.WSSUtils` Class

The `oracle.security.xmlsec.wss.utils.WSSUtils` class contains static utility methods for WSS. Some methods that may be frequently used in applications include:

- `addWsuIdToElement()`
- `createTextFromChild()`
- `insertChildElementWithText()`
- `prependChild()`
- `encodeBinary()`
- `decodeBinary()`

The Oracle Web Services Security Java API Reference

The Oracle Web Services Security API Reference (Javadoc) is available at:

Oracle Security Developer Tools Web Services Security Java API Reference

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at

The Liberty Alliance is an open organization that was founded with the goal of allowing individuals and businesses to engage in virtually any transaction without compromising the privacy and security of vital identity information. Specifications issued by the Liberty Alliance are based on an open identity federation framework, allowing partner companies to form business relationships based on a cross-organizational, federated network identity model.

This chapter describes the features and benefits of the Oracle Liberty SDK, and explains how to set up your environment and use Oracle Liberty SDK.

This chapter contains these topics:

- Oracle Liberty SDK Features and Benefits
- Oracle Liberty 1.1
- Oracle Liberty 1.2

Oracle Liberty SDK Features and Benefits

Oracle Liberty SDK allows Java developers to design and develop single sign-on (SSO) and federated identity management (FIM) solutions. Oracle Liberty SDK aims to unify, simplify, and extend all aspects of development and integration of systems conforming to the Liberty Alliance ID-FF 1.1 and 1.2 specifications.

Oracle Liberty SDK 1.1 and 1.2 enable simplified software development through the use of an intuitive and straightforward Java API. The toolkits provide tools, information, and examples to help you develop solutions that conform to the Liberty Alliance specifications. The toolkits can also be seamlessly integrated into any existing Java solution, including applets, applications, EJBs, servlets, JSPs, and so on.

The Oracle Liberty SDK is a pure java solution which provides the following features:

- Support for the Liberty Alliance ID-FF version 1.1 and 1.2 specifications
- Support for Liberty-based Single Sign-on and Federated Identity protocols
- Support for the SAML 1.0/1.1 specifications

See Also: You can find the Liberty Alliance specifications at http://www.projectliberty.org/resources/specifications.php.
Oracle Liberty 1.1

This section explains how to set up your environment for and use Oracle Liberty 1.1, and describes the classes and interfaces of Oracle Liberty 1.1. It contains the following topics:

- Setting Up Your Oracle Liberty 1.1 Environment
- Overview of Oracle Liberty 1.1 Classes and Interfaces
- The Oracle Liberty SDK 1.1 API Reference
- Example Programs

Setting Up Your Oracle Liberty 1.1 Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME.

This section explains how to set up your environment for Oracle Liberty 1.1. It contains these topics:

- System Requirements for Oracle Liberty 1.1
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle Liberty 1.1

In order to use Oracle Liberty 1.1, your system must have the Java Development Kit (JDK) version 1.2.2 or higher.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar
- osdt_cert.jar
- osdt_xmlsec.jar
- osdt_saml.jar
- The jaxen.jar file (Jaxen XPath engine, included with your Oracle XML Security distribution)
- the osdt_lib_v11.jar file

Setting the CLASSPATH on Windows

To set the CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all of the required jar files to the CLASSPATH.
For example, your CLASSPATH might look like this:

```
%CLASSPATH%;C:\ORACLE_HOME\jlib\osdt_core.jar;
C:\ORACLE_HOME\jlib\osdt_cert.jar;
C:\ORACLE_HOME\jlib\osdt_xmlsec.jar;
C:\ORACLE_HOME\jlib\osdt_saml.jar;
C:\ORACLE_HOME\jlib\jaxen.jar;
C:\ORACLE_HOME\jlib\osdt_lib_v11.jar;
```

6. Click OK.

**Setting the CLASSPATH on UNIX** To set your CLASSPATH on UNIX, set your CLASSPATH environment variable to include the full path and file name of all of the required jar and class files. For example:

```
setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
$ORACLE_HOME/jlib/osdt_cert.jar:
$ORACLE_HOME/jlib/osdt_xmlsec.jar:
$ORACLE_HOME/jlib/osdt_saml.jar:
$ORACLE_HOME/jlib/jaxen.jar:
$ORACLE_HOME/jlib/osdt_lib_v11.jar
```

### Overview of Oracle Liberty 1.1 Classes and Interfaces

This section introduces some useful classes and interfaces of Oracle Liberty SDK v. 1.1. It contains these topics:

- Core Classes and Interfaces
- Supporting Classes and Interfaces

### Core Classes and Interfaces

This section describes core classes and interfaces of the Oracle Liberty SDK v. 1.1. The core classes are:

- The `oracle.security.xmlsec.liberty.v11.AuthnRequest` Class
- The `oracle.security.xmlsec.liberty.v11.AuthnResponse` Class
- The `oracle.security.xmlsec.liberty.v11.FederationTerminationNotification` Class
- The `oracle.security.xmlsec.liberty.v11.LogoutRequest` Class
- The `oracle.security.xmlsec.liberty.v11.LogoutResponse` Class
- The `oracle.security.xmlsec.liberty.v11.RegisterNameIdentifierRequest` Class
- The `oracle.security.xmlsec.liberty.v11.RegisterNameIdentifierResponse` Class

The `oracle.security.xmlsec.liberty.v11.AuthnRequest` Class

This class represents the AuthnRequest element of the Liberty protocol schema.

**Example 11–1** Creating an AuthnRequest Element and Appending it to a Document

```
Document doc = Instance of org.w3c.dom.Document;
AuthnRequest authnRequest = new AuthnRequest(doc);
doc.getDocumentElement().appendChild(authnRequest);
```

**Example 11–2** shows how to obtain AuthnRequest elements from an XML document.
Example 11–2  Obtaining AuthnRequest Elements from a Document

Document doc = Instance of org.w3c.dom.Document;

// Get list of all AuthnRequest elements in the document.
NodeList arList =
    doc.getElementsByTagNameNS(LibertyURI.ns_liberty, "AuthnRequest");
if (arList.getLength() == 0)
    System.err.println("No AuthnRequest elements found.");

// Convert each org.w3c.dom.Node object to an
// oracle.security.xmlsec.liberty.v11.AuthnRequest object and process
for (int s = 0, n = arList.getLength(); s < n; ++s)
{
    AuthnRequest authnRequest =
        new AuthnRequest((Element)arList.item(s));

    // Process AuthnRequest element
    ...
}

The oracle.security.xmlsec.liberty.v11.AuthnResponse Class

This class represents the AuthnResponse element of the Liberty protocol schema.
Example 11–3 shows how to create a new AuthnResponse element and append it to a document.

Example 11–3  Creating an AuthnResponse Element and Appending it to a Document

Document doc = Instance of org.w3c.dom.Document;
AuthnResponse authnResponse = new AuthnResponse(doc);
doc.getDocumentElement().appendChild(authnResponse);

Example 11–4 shows how to obtain AuthnResponse elements from an XML document.

Example 11–4  Obtaining AuthnResponse elements from a Document

Document doc = Instance of org.w3c.dom.Document;

// Get list of all AuthnResponse elements in the document.
NodeList arList =
    doc.getElementsByTagNameNS(LibertyURI.ns_liberty, "AuthnResponse");
if (arList.getLength() == 0)
    System.err.println("No AuthnResponse elements found.");

// Convert each org.w3c.dom.Node object to an
// oracle.security.xmlsec.liberty.v11.AuthnResponse object and process
for (int s = 0, n = arList.getLength(); s < n; ++s)
{
    AuthnResponse authnResponse =
        new AuthnResponse((Element)arList.item(s));

    // Process AuthnResponse element
    ...
}

The oracle.security.xmlsec.liberty.v11.FederationTerminationNotification Class

This class represents the FederationTerminationNotification element of the Liberty protocol schema.
Example 11–5 shows how to create a new federation termination notification element and append it to a document.

**Example 11–5  Creating a FederationTerminationNotification Element and Appending it to a Document**

```java
Document doc = Instance of org.w3c.dom.Document;
FederationTerminationNotification ftn =
    new FederationTerminationNotification(doc);
doc.getDocumentElement().appendChild(ftn);
```

Example 11–6 shows how to obtain federation termination notification elements from an XML document.

**Example 11–6  Obtaining FederationTerminationNotification Elements from a Document**

```java
Document doc = Instance of org.w3c.dom.Document;

// Get list of all FederationTerminationNotification elements in the document
NodeList ftnList = doc.getElementsByTagNameNS(LibertyURI.ns_liberty,
   "FederationTerminationNotification");
if (ftnList.getLength() == 0)
    System.err.println("No FederationTerminationNotification elements found.");

// Convert each org.w3c.dom.Node object to an
// oracle.security.xmlsec.liberty.v11.FederationTerminationNotification
// object and process
for (int s = 0, n = ftnList.getLength(); s < n; ++s)
{
    FederationTerminationNotification ftn =
        new FederationTerminationNotification((Element)ftnList.item(s));

    // Process FederationTerminationNotification element
    ...
}
```

The oracle.security.xmlsec.liberty.v11.LogoutRequest Class

This class represents the LogoutRequest element of the Liberty protocol schema.

Example 11–7 shows how to create a new LogoutRequest element and append it to a document.

**Example 11–7  Creating a LogoutRequest Element and Appending it to a Document**

```java
Document doc = Instance of org.w3c.dom.Document;
LogoutRequest lr = new LogoutRequest(doc);
doc.getDocumentElement().appendChild(lr);
```

Example 11–8 shows how to obtain LogoutRequest elements from an XML document.

**Example 11–8  Obtaining LogoutRequest Elements from an XML Document**

```java
Document doc = Instance of org.w3c.dom.Document;

// Get list of all LogoutRequest elements in the document.
NodeList lrList = doc.getElementsByTagNameNS(LibertyURI.ns_liberty,
   "LogoutRequest");
if (lrList.getLength() == 0)
System.err.println("No LogoutRequest elements found.");

// Convert each org.w3c.dom.Node object to an
// oracle.security.xmlsec.liberty.v11.LogoutRequest
// object and process
for (int s = 0, n = lrList.getLength(); s < n; ++s)
{
    LogoutRequest lr = new LogoutRequest((Element)lrList.item(s));

    // Process LogoutRequest element
    ...
}

The oracle.security.xmlsec.liberty.v11.LogoutResponse Class
This class represents the LogoutResponse element of the Liberty protocol schema.
Example 11–9 shows how to create a new LogoutResponse element and append it to
document.

Example 11–9 Creating a LogoutResponse Element and Appending it to a Document
Document doc = Instance of org.w3c.dom.Document;
LogoutResponse lr = new LogoutResponse(doc);
doc.getDocumentElement().appendChild(lr);

Example 11–10 shows how to obtain LogoutResponse elements from an XML
document.

Example 11–10 Obtaining LogoutResponse elements from a Document
Document doc = Instance of org.w3c.dom.Document;

// Get list of all LogoutResponse elements in the document.
NodeList lrList =
    doc.getElementsByTagNameNS(LibertyURI.ns_liberty, "LogoutResponse");
if (lrList.getLength() == 0)
    System.err.println("No LogoutResponse elements found.");

// Convert each org.w3c.dom.Node object to an
// oracle.security.xmlsec.liberty.v11.LogoutResponse
// object and process
for (int s = 0, n = lrList.getLength(); s < n; ++s)
{
    LogoutResponse lr = new LogoutResponse((Element)lrList.item(s));

    // Process LogoutResponse element
    ...
}

The oracle.security.xmlsec.liberty.v11.RegisterNameIdentifierRequest Class
This class represents the RegisterNameIdentifierRequest element of the Liberty
protocol schema.
Example 11–11 shows how to create a new RegisterNameIdentifierRequest
element and append it to a document.
Example 11–11  Creating a RegisterNameIdentifierRequest Element and Appending it to a Document

```java
Document doc = Instance of org.w3c.dom.Document;
RegisterNameIdentifierRequest rnir =
    new RegisterNameIdentifierRequest(doc);
doc.getDocumentElement().appendChild(rnir);
```

Example 11–12 shows how to obtain RegisterNameIdentifierRequest elements from an XML document.

Example 11–12  Obtaining RegisterNameIdentifierRequest Elements from an XML Document

```java
Document doc = Instance of org.w3c.dom.Document;

// Get list of all RegisterNameIdentifierRequest elements in the document
NodeList rnirList = doc.getElementsByTagNameNS(LibertyURI.ns_liberty,
    "RegisterNameIdentifierRequest");
if (rnirList.getLength() == 0)
    System.err.println("No RegisterNameIdentifierRequest elements found.");

// Convert each org.w3c.dom.Node object to an
//oracle.security.xmlsec.liberty.v11.RegisterNameIdentifierRequest
// object and process
for (int s = 0, n = rnirList.getLength(); s < n; ++s)
{
    RegisterNameIdentifierRequest rnir = new
        RegisterNameIdentifierRequest((Element)rnirList.item(s));

    // Process RegisterNameIdentifierRequest element
    ...
}
```

The oracle.security.xmlsec.liberty.v11.RegisterNameIdentifierResponse Class

This class represents the RegisterNameIdentifierResponse element of the Liberty protocol schema.

Example 11–13 shows how to create a new RegisterNameIdentifierResponse element and append it to a document.

Example 11–13  Creating a RegisterNameIdentifierResponse Element and Appending it to a Document

```java
Document doc = Instance of org.w3c.dom.Document;
RegisterNameIdentifierResponse rnir = new RegisterNameIdentifierResponse(doc);
doc.getDocumentElement().appendChild(rnir);
```

Example 11–14 shows how to obtain RegisterNameIdentifierResponse elements from an XML document.

Example 11–14  Obtaining RegisterNameIdentifierResponse Elements from an XML Document

```java
Document doc = Instance of org.w3c.dom.Document;

// Get list of all RegisterNameIdentifierResponse elements in the document
NodeList rnirList = doc.getElementsByTagNameNS(LibertyURI.ns_liberty,
    "RegisterNameIdentifierResponse");
if (rnirList.getLength() == 0)
System.err.println("No RegisterNameIdentifierResponse elements found.");

// Convert each org.w3c.dom.Node object to an
// oracle.security.xmlsec.liberty.v11.RegisterNameIdentifierResponse
// object and process
for (int s = 0, n = rnirList.getLength(); s < n; ++s)
{
    RegisterNameIdentifierResponse rnir = new
        RegisterNameIdentifierResponse((Element)rnirList.item(s));

    // Process RegisterNameIdentifierResponse element
    ...
}

### Supporting Classes and Interfaces

This section describes supporting classes and interfaces of Oracle Liberty SDK v. 1.1.

The supporting classes and interfaces are:

- The `oracle.security.xmlsec.liberty.v11.LibertyInitializer` class
- The `oracle.security.xmlsec.liberty.v11.LibertyURI` interface
- The `oracle.security.xmlsec.liberty.v11.ac.AuthenticationContextURI` interface
- The `oracle.security.xmlsec.util.ac.AuthenticationContextStatement` class
- The `oracle.security.xmlsec.saml.SAMLURI` Interface
- The `oracle.security.xmlsec.saml.SAMLMessage` class

#### The `oracle.security.xmlsec.liberty.v11.LibertyInitializer` class

The `oracle.security.xmlsec.liberty.v11.LibertyInitializer` class handles load-time initialization and configuration of the Oracle Liberty SDK library. You must call this class's static `initialize()` method before making any calls to the Oracle Liberty SDK API.

#### The `oracle.security.xmlsec.liberty.v11.LibertyURI` interface

The `oracle.security.xmlsec.liberty.v11.LibertyURI` interface defines URI string constants for algorithms, namespaces and objects. The following naming convention is used:

- Algorithm URIs begin with "alg_".
- Namespace URIs begin with "ns_".
- Object type URIs begin with "obj_".
- Liberty profile namespace URIs begin with "prof_".

#### The `oracle.security.xmlsec.liberty.v11.ac.AuthenticationContextURI` interface

The `oracle.security.xmlsec.liberty.v11.ac.AuthenticationContextURI` interface defines URI string constants for algorithms, namespaces and objects. The following naming convention is used:

- Algorithm URIs begin with "alg_".
- Namespace URIs begin with "ns_".

11-8 Oracle Security Developer Tools Reference
- Object type URIs begin with "obj_".

The oracle.security.xmlsec.util.ac.AuthenticationContextStatement class

The oracle.security.xmlsec.util.ac.AuthenticationContextStatement class is an abstract class representing the top-level AuthenticationContextStatement element of the Liberty authentication context schema. Each concrete implementation of this class represents a respective class defined in the Liberty Authentication Context Specification.

The oracle.security.xmlsec.saml.SAMLURI Interface

The oracle.security.xmlsec.saml.SAMLURI interface defines URI string constants for algorithms, namespaces and objects. The following naming convention is used:
- Action namespace URIs defined in the SAML 1.0 specifications begin with "action_"
- Authentication method namespace URIs defined in the SAML 1.0 specifications begin with "authentication_method_"
- Confirmation method namespace URIs defined in the SAML 1.0 specifications begin with "confirmation_method_"
- Namespace URIs begin with "ns_".

The oracle.security.xmlsec.saml.SAMLMessage class

The oracle.security.xmlsec.saml.SAMLMessage class is the base class for all the SAML and SAML extension messages that may be signed and contain an XML-DSIG structure.

The Oracle Liberty SDK 1.1 API Reference

The Oracle Liberty SDK version 1.1 API Reference is available at:

Oracle Security Developer Tools Liberty SDK 1.1 Java API Reference

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at http://www.oracle.com/technology/index.html.

Oracle Liberty 1.2

This section describes the classes and interfaces of Oracle Liberty 1.2, and explains how to set up your environment and use Oracle Liberty 1.2. It contains these sections:
- Setting Up Your Oracle Liberty 1.2 Environment
- Overview of Oracle Liberty 1.2 Classes and Interfaces
- The Oracle Liberty SDK 1.2 API Reference
- Example Programs
Setting Up Your Oracle Liberty 1.2 Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME.

This section explains how to set up your environment for Oracle Liberty 1.2. It contains these topics:

- System Requirements for Oracle Liberty 1.2
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle Liberty 1.2

In order to use Oracle Liberty 1.2, your system must have the Java Development Kit (JDK) version 1.2.2 or higher. Also, make sure that your PATH environment variable includes the Java bin directory.

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to all of the required jar and class files. Make sure the following items are included in your CLASSPATH:

- osdt_core.jar
- osdt_cert.jar
- osdt_xmlsec.jar
- osdt_saml.jar
- The jaxen.jar file (Jaxen XPath engine, included with your Oracle XML Security distribution)
- osdt_lib_v12.jar

Setting the CLASSPATH on Windows

To set the CLASSPATH on Windows:

1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all of the required jar files to the CLASSPATH. For example, your CLASSPATH might look like this:

   %CLASSPATH%;C:\ORACLE_HOME\jlib\osdt_core.jar;
   C:\ORACLE_HOME\jlib\osdt_cert.jar;
   C:\ORACLE_HOME\jlib\osdt_xmlsec.jar;
   C:\ORACLE_HOME\jlib\osdt_saml.jar;
   C:\ORACLE_HOME\jlib\jaxen\jaxen.jar;
   C:\ORACLE_HOME\jlib\osdt_lib_v12.jar;

6. Click OK.

Setting the CLASSPATH on Unix
On Unix, set your CLASSPATH environment variable to include the full path and file name of all of the required jar and class files. For example:

```
setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
$ORACLE_HOME/jlib/osdt_cert.jar:
$ORACLE_HOME/jlib/osdt_xmlsec.jar:
$ORACLE_HOME/jlib/osdt_saml.jar:
$ORACLE_HOME/jlib/jaxen/jaxen.jar:
$ORACLE_HOME/jlib/osdt_lib_v12.jar
```

---

### Overview of Oracle Liberty 1.2 Classes and Interfaces

This section introduces some useful classes and interfaces of Oracle Liberty SDK v. 1.2. It contains these topics:

- Core Classes and Interfaces
- Supporting Classes and Interfaces

#### Core Classes and Interfaces

This section describes core classes and interfaces of the Oracle Liberty SDK, v. 1.2.

The core classes are:

- The oracle.security.xmlsec.saml.Assertion class
- The oracle.security.xmlsec.samlp.Request class
- The oracle.security.xmlsec.samlp.Response class
- The oracle.security.xmlsec.liberty.v12.AuthnRequest class
- The oracle.security.xmlsec.liberty.v12.AuthnResponse class
- The oracle.security.xmlsec.liberty.v12.FederationTerminationNotification class
- The oracle.security.xmlsec.liberty.v12.LogoutRequest class
- The oracle.security.xmlsec.liberty.v12.LogoutResponse class
- The oracle.security.xmlsec.liberty.v12.RegisterNameIdentifierRequest class
- The oracle.security.xmlsec.liberty.v12.RegisterNameIdentifierResponse class

#### The oracle.security.xmlsec.saml.Assertion class

The oracle.security.xmlsec.saml.Assertion class represents the Assertion element of the SAML Assertion schema.

Example 11–15 shows how to create a new assertion element and append it to a document.

**Example 11–15 Creating an Assertion element and Appending it to a Document**

```
Document doc = Instance of org.w3c.dom.Document;
Assertion assertion = new Assertion(doc);
doc.getDocumentElement().appendChild(assertion);
```

Example 11–16 shows how to obtain assertion elements from an XML document.

**Example 11–16 Obtaining Assertion Elements from a Document**

```
Document doc = Instance of org.w3c.dom.Document;
```
// Get list of all Assertion elements in the document
NodeList assertList =
    doc.getElementsByTagNameNS(SAMLURI.ns_saml, "Assertion");
if (assertList.getLength() == 0)
    System.err.println("No Assertion elements found.");

// Convert each org.w3c.dom.Node object to
// an oracle.security.xmlsec.saml.Assertion
// object and process
for (int s = 0, n = assertList.getLength(); s < n; ++s)
{
    Assertion assertion = new Assertion((Element)assertList.item(s));

    // Process Assertion element
    ...
}

The oracle.security.xmlsec.samlp.Request class
The oracle.security.xmlsec.samlp.Request class represents the Request element of the SAML Protocol schema.

Example 11–17 shows how to create a new Request element and append it to a document.

Example 11–17 Creating a Request element and Appending it to a Document
Document doc = Instance of org.w3c.dom.Document;
Request request = new Request(doc);
doc.getDocumentElement().appendChild(request);

Example 11–18 shows how to obtain Request elements from an XML document.

Example 11–18 Obtaining Request Elements from a Document
Document doc = Instance of org.w3c.dom.Document;

// Get list of all Request elements in the document
NodeList reqList =
    doc.getElementsByTagNameNS(SAMLURI.ns_samlp, "Request");
if (reqList.getLength() == 0)
    System.err.println("No Request elements found.");

// Convert each org.w3c.dom.Node object to an
// oracle.security.xmlsec.samlp.Request
// object and process
for (int s = 0, n = reqList.getLength(); s < n; ++s)
{
    Request request = new Request((Element)reqList.item(s));

    // Process Request element
    ...
}

The oracle.security.xmlsec.samlp.Response class
The oracle.security.xmlsec.samlp.Response class represents the Response element of the SAML Protocol schema.
Example 11–19 shows how to create a new element and append it to a document.

**Example 11–19  Creating a Response Element and Appending it to a Document**

Document doc = Instance of org.w3c.dom.Document;
Response response = new Response(doc);
doc.getDocumentElement().appendChild(response);

Example 11–20 shows how to obtain Response elements from an XML document.

**Example 11–20  Obtaining Response Elements from a Document**

Document doc = Instance of org.w3c.dom.Document;

// Get list of all Response elements in the document
NodeList respList =
    doc.getElementsByTagNameNS(SAMLURI.ns_samlp, "Response");
if (respList.getLength() == 0)
    System.err.println("No Response elements found.");

// Convert each org.w3c.dom.Node object to an
// oracle.security.xmlsec.samlp.Response
// object and process
for (int s = 0, n = respList.getLength(); s < n; ++s)
{
    Response response = new Response((Element)respList.item(s));

    // Process Response element
    ...
}

The oracle.security.xmlsec.liberty.v12.AuthnRequest class

The oracle.security.xmlsec.liberty.v12.AuthnRequest class represents
the AuthnRequest element of the Liberty protocol schema.

Example 11–21 shows how to create a new authorization request element and append
it to a document.

**Example 11–21  Creating an AuthnRequest Element and Appending it to a Document**

Document doc = Instance of org.w3c.dom.Document;
AuthnRequest authnRequest = new AuthnRequest(doc);
doc.getDocumentElement().appendChild(authnRequest);

Example 11–22 shows how to obtain AuthnRequest elements from an XML
document.

**Example 11–22  Obtaining AuthnRequest Elements from a Document**

Document doc = Instance of org.w3c.dom.Document;

// Get list of all AuthnRequest elements in the document
NodeList arList = doc.getElementsByTagNameNS(LibertyURI.ns_liberty,
    "AuthnRequest");
if (arList.getLength() == 0)
    System.err.println("No AuthnRequest elements found.");

// Convert each org.w3c.dom.Node object to
The `oracle.security.xmlsec.liberty.v12.AuthnRequest` class


**Example 11–23 Creating an AuthnResponse Element and Appending it to a Document**

```java
Document doc = Instance of org.w3c.dom.Document;
AuthnResponse authnResponse = new AuthnResponse(doc);
doc.getDocumentElement().appendChild(authnResponse);
```

Example 11–24 shows how to obtain `AuthnResponse` elements from an XML document.

**Example 11–24 Obtaining AuthnResponse Elements from a Document**

```java
Document doc = Instance of org.w3c.dom.Document;
// Get list of all AuthnResponse elements in the document.
NodeList arList =
    doc.getElementsByTagNameNS(LibertyURI.ns_liberty, "AuthnResponse");
if (arList.getLength() == 0)
    System.err.println("No AuthnResponse elements found.");

// Convert each org.w3c.dom.Node object to
// an oracle.security.xmlsec.liberty.v12.AuthnResponse
// object and process
for (int s = 0, n = arList.getLength(); s < n; ++s)
{
    AuthnResponse authnResponse =
        new AuthnResponse((Element)arList.item(s));

    // Process AuthnResponse element
    ...
}
```

The `oracle.security.xmlsec.liberty.v12.FederationTerminationNotification` class

The `oracle.security.xmlsec.liberty.v12.FederationTerminationNotification` class represents the `FederationTerminationNotification` element of the Liberty protocol schema.

Example 11–25 shows how to create a new federation termination notification element and append it to a document.
Example 11–25  Creating a DocumentFederationTerminationNotification Element and Appending it to a Document

```java
Document doc = Instance of org.w3c.dom.Document;
FederationTerminationNotification ftn =
    new FederationTerminationNotification(doc);
doc.getDocumentElement().appendChild(ftn);
```

Example 11–26 shows how to obtain federation termination notification elements from an XML document.

Example 11–26  Obtaining FederationTerminationNotification Elements from a Document

```java
Document doc = Instance of org.w3c.dom.Document;

// Get list of all FederationTerminationNotification elements in the document
NodeList ftnList = doc.getElementsByTagNameNS(LibertyURI.ns_liberty, "FederationTerminationNotification");
if (ftnList.getLength() == 0)
    System.err.println("No FederationTerminationNotification elements found.");

// Convert each org.w3c.dom.Node object to an oracle.security.xmlsec.liberty.v12.FederationTerminationNotification object and process
for (int s = 0, n = ftnList.getLength(); s < n; ++s)
{
    FederationTerminationNotification ftn = new
        FederationTerminationNotification((Element)ftnList.item(s));

    // Process FederationTerminationNotification element
    ...
}
```

The oracle.security.xmlsec.liberty.v12.LogoutRequest class

The oracle.security.xmlsec.liberty.v12.LogoutRequest class represents the LogoutRequest element of the Liberty protocol schema.

Example 11–27 shows how to create a new element and append it to a document.

Example 11–27  Creating a new LogoutRequest Element and Appending it to a Document

```java
Document doc = Instance of org.w3c.dom.Document;
LogoutRequest lr = new LogoutRequest(doc);
doc.getDocumentElement().appendChild(lr);
```

Example 11–28 shows how to obtain logout request elements from an XML document.

Example 11–28  Obtaining LogoutRequest Elements from an XML Document

```java
Document doc = Instance of org.w3c.dom.Document;

// Get list of all LogoutRequest elements in the document
NodeList lrList =
    doc.getElementsByTagNameNS(LibertyURI.ns_liberty, "LogoutRequest");
if (lrList.getLength() == 0)
    System.err.println("No LogoutRequest elements found.");

// Convert each org.w3c.dom.Node object to
// an oracle.security.xmlsec.liberty.v12.LogoutRequest
// object and process
```
for (int s = 0, n = lrList.getLength(); s < n; ++s)
{
    LogoutRequest lr = new LogoutRequest((Element)lrList.item(s));
    // Process LogoutRequest element
    ...
}

The oracle.security.xmlsec.liberty.v12.LogoutResponse class

The oracle.security.xmlsec.liberty.v12.LogoutResponse class represents the LogoutResponse element of the Liberty protocol schema.

Example 11–29 shows how to create a new logout response element and append it to a document.

Example 11–29  Creating a new LogoutResponse Element and Appending it to a Document

Document doc = Instance of org.w3c.dom.Document;
LogoutResponse lr = new LogoutResponse(doc);
doc.getDocumentElement().appendChild(lr);

Example 11–30 shows how to obtain logout response elements from an XML document.

Example 11–30  Obtaining LogoutResponse Elements from an XML Document

Document doc = Instance of org.w3c.dom.Document;

    // Get list of all LogoutResponse elements in the document
    NodeList lrList =
        doc.getElementsByTagNameNS(LibertyURI.ns_liberty, "LogoutResponse");
    if (lrList.getLength() == 0)
        System.err.println("No LogoutResponse elements found.");

    // Convert each org.w3c.dom.Node object to
    // an oracle.security.xmlsec.liberty.v12.LogoutResponse
    // object and process
    for (int s = 0, n = lrList.getLength(); s < n; ++s)
    {
        LogoutResponse lr = new LogoutResponse((Element)lrList.item(s));
        // Process LogoutResponse element
        ...
    }

The oracle.security.xmlsec.liberty.v12.RegisterNameIdentifierRequest class

The oracle.security.xmlsec.liberty.v12.RegisterNameIdentifierRequest class represents the RegisterNameIdentifierRequest element of the Liberty protocol schema.

Example 11–31 shows how to create a new RegisterNameIdentifierRequest element and append it to a document.
**Example 11–31  Creating a new RegisterNameIdentifierRequest Element and Appending it to a Document**

```java
document doc = Instance of org.w3c.dom.Document;
RegisterNameIdentifierRequest rnir = new RegisterNameIdentifierRequest(doc);
doc.getDocumentElement().appendChild(rnir);
```

**Example 11–32 shows how to obtain RegisterNameIdentifierRequest elements from an XML document.**

**Example 11–32  Obtaining RegisterNameIdentifierRequest Elements from an XML Document**

```java
document doc = Instance of org.w3c.dom.Document;

// Get list of all
// RegisterNameIdentifierRequest elements
// in the document
NodeList rnirList =
    doc.getElementsByTagNameNS(LibertyURI.ns_liberty,
        "RegisterNameIdentifierRequest");
if (rnirList.getLength() == 0)
    System.err.println("No RegisterNameIdentifierRequest elements found.");

// Convert each org.w3c.dom.Node object to a
// oracle.security.xmlsec.liberty.v12.RegisterNameIdentifierRequest
// object and process
for (int s = 0, n = rnirList.getLength(); s < n; ++s)
{
    RegisterNameIdentifierRequest rnir =
        new RegisterNameIdentifierRequest((Element)rnirList.item(s));

    // Process RegisterNameIdentifierRequest element
    ...
}
```

**The oracle.security.xmlsec.liberty.v12.RegisterNameIdentifierResponse class**

The oracle.security.xmlsec.liberty.v12.RegisterNameIdentifierResponse class represents the RegisterNameIdentifierResponse element of the Liberty protocol schema.

**Example 11–33 shows how to create a new RegisterNameIdentifierResponse element and append it to a document.**

**Example 11–33  Creating a New RegisterNameIdentifierResponse Element and Appending it to a Document**

```java
document doc = Instance of org.w3c.dom.Document;
RegisterNameIdentifierResponse rnir =
    new RegisterNameIdentifierResponse(doc);
doc.getDocumentElement().appendChild(rnir);
```

**Example 11–34 shows how to obtain RegisterNameIdentifierResponse elements from an XML document.**

**Example 11–34  Obtaining RegisterNameIdentifierResponse Elements from a Document**

```java
document doc = Instance of org.w3c.dom.Document;
```
// Get list of all RegisterNameIdentifierResponse elements in the document
NodeList rnirList =
    doc.getElementsByTagNameNS(LibertyURI.ns_liberty,
    "RegisterNameIdentifierResponse");

if (rnirList.getLength() == 0)
    System.err.println("No RegisterNameIdentifierResponse elements found.");

// Convert each org.w3c.dom.Node object to an
// oracle.security.xmlsec.liberty.v12.RegisterNameIdentifierResponse
// object and process
for (int s = 0, n = rnirList.getLength(); s < n; ++s)
{
    RegisterNameIdentifierResponse rnir = new
        RegisterNameIdentifierResponse((Element)rnirList.item(s));

    // Process RegisterNameIdentifierResponse element
    ...
}

### Supporting Classes and Interfaces

This section describes supporting classes and interfaces of Oracle Liberty SDK v.1.2:

- The `oracle.security.xmlsec.liberty.v12.LibertyInitializer` class
- The `oracle.security.xmlsec.liberty.v12.LibertyURI` interface
- The `oracle.security.xmlsec.util.ac.AuthenticationContextStatement` class
- The `oracle.security.xmlsec.saml.SAMLInitializer` class
- The `oracle.security.xmlsec.saml.SAMLURI` interface

#### The `oracle.security.xmlsec.liberty.v12.LibertyInitializer` class

This class handles load-time initialization and configuration of the Oracle Liberty SDK 1.2 library. You must call this class's static `initialize()` method before making any calls to the Oracle Liberty SDK 1.2 API.

#### The `oracle.security.xmlsec.liberty.v12.LibertyURI` interface

This interface defines URI string constants for algorithms, namespaces, and objects.

#### The `oracle.security.xmlsec.util.ac.AuthenticationContextStatement` class

This is an abstract class representing the top-level `AuthenticationContextStatement` element of the Liberty authentication context schema. Each concrete implementation of this class represents the respective class defined in the Liberty Authentication Context Specification.

#### The `oracle.security.xmlsec.saml.SAMLInitializer` class

This class handles load-time initialization and configuration of the Oracle SAML library. You should call this class's static `initialize(int major, int minor)` method, for version 1.1, before making any calls to the Oracle SAML Toolkit API for SAML 1.1.
The oracle.security.xmlsec.saml.SAMLURI Interface

The oracle.security.xmlsec.saml.SAMLURI interface defines URI string constants for algorithms, namespaces, and objects. The following naming convention is used:

- Action Namespace URIs defined in the SAML 1.1 specifications begin with "action_
- Authentication Method Namespace URIs defined in the SAML 1.1 specifications begin with "authentication_method_
- Confirmation Method Namespace URIs defined in the SAML 1.1 specifications begin with "confirmation_method_
- Namespace URIs begin with "ns_

The oracle.security.xmlsec.saml.SAMLMessage Class

oracle.security.xmlsec.saml.SAMLMessage is the base class for all the SAML and SAML extension messages that may be signed and contain an XML-DSIG structure.

The Oracle Liberty SDK 1.2 API Reference

The Oracle Liberty SDK version 1.2 API Reference (Javadoc) is available at:

Oracle Security Developer Tools Liberty SDK 1.2 Java API Reference

Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at http://www.oracle.com/technology/index.html.
Oracle XKMS

XKMS (XML Key Management Specification) is a W3C specification for public key management. It provides a convenient way to handle public key infrastructures by enabling developers to write XML transactions for digital signature processing.

This chapter contains these topics:

- Oracle XKMS Features and Benefits
- Setting Up Your Oracle XKMS Environment
- Core Classes and Interfaces
- The Oracle XKMS Java API Reference
- Example Programs

Oracle XKMS Features and Benefits

Oracle XKMS is a pure Java solution which consists of a toolkit for locating keys and verifying user identities across businesses and applications. It supports the secure, trusted messaging required for web services, and provides a way to sidestep some of the costs and complexity associated with PKI.

Oracle XKMS provides the following features:

- Simplified access to PKI functionality - by implementing the W3C XKMS Standard, Oracle XKMS combines the simplicity of XML with the robustness of PKI. With this toolkit, developers can easily deploy robust application functionality by deploying secure, lightweight client software.
- Supports complete key/certificate life cycle - Oracle XKMS helps enterprise applications locate, retrieve, and validate signature and encryption keys using lightweight Web Services infrastructure.
- Secures XKMS messages using XML Signatures - requests and responses can be digitally signed using Oracle XML toolkit.
- 100% Java with no native methods
- Works with JAXP 1.1 compliant XML parsers

Oracle XKMS Packages

The Oracle XKMS library contains the following packages:
Setting Up Your Oracle XKMS Environment

The Oracle Security Developer Tools are installed with Oracle Application Server in ORACLE_HOME. This section explains how to set up your environment for Oracle XKMS. It contains these topics:

- System Requirements for Oracle XKMS
- Setting the CLASSPATH Environment Variable

System Requirements for Oracle XKMS

In order to use Oracle XKMS, your system must have the following components installed:

- The Java Development Kit (JDK) version 1.2.2 or higher
- A JAXP-compatible XML parser and XSLT processor
- the Oracle XML Security toolkit

Setting the CLASSPATH Environment Variable

Your CLASSPATH environment variable must contain the full path and file names to the required jar and class files. Make sure that the following files are included in your CLASSPATH:

- osdt_core.jar
- osdt_cert.jar
- osdt_xmlsec.jar
- jaxen.jar, which is included in the $ORACLE_HOME/jlib directory of the security tools distribution. Oracle XML Security relies on the Jaxen XPath engine for XPath processing.
- The appropriate XML parser and XSLT processor implementations, unless you have installed them in your JRE's /lib/ext or /lib/endorsed directory.

Note: The Jaxen library included in the Oracle XML Security distribution is a modified version of the Jaxen 1.0 FCS release. If you also have an earlier Jaxen release in your CLASSPATH, you must ensure that the version from this distribution appears first.

Setting the CLASSPATH on Windows

To set your CLASSPATH on Windows:
1. In your Windows Control Panel, select System.
2. In the System Properties dialog, select the Advanced tab.
3. Click Environment Variables.
4. In the User Variables section, click New to add a CLASSPATH environment variable for your user profile. If a CLASSPATH environment variable already exists, select it and click Edit.
5. Add the full path and file names for all of the required jar and class files to the CLASSPATH.
   For example, your CLASSPATH might look like this:

   ```
   C:\ORACLE_HOME\jlib\osdt_core.jar;
   C:\ORACLE_HOME\jlib\osdt_cert.jar
   C:\ORACLE_HOME\jlib\osdt_xmlsec.jar
   ```
6. Click OK.

**Setting the CLASSPATH on UNIX**

On UNIX, set your CLASSPATH environment variable to include the full path and file name of all of the required jar and class files. For example:

```shell
setenv CLASSPATH $CLASSPATH:$ORACLE_HOME/jlib/osdt_core.jar:
C:\ORACLE_HOME\jlib\osdt_cert.jar
C:\ORACLE_HOME\jlib\osdt_xmlsec.jar
```

**Core Classes and Interfaces**

This section provides information and code samples for using the key classes and interfaces of Oracle XKMS. The core classes are:

- `oracle.security.xmlsec.xkms.xkiss.LocateRequest`
- `oracle.security.xmlsec.xkms.xkiss.LocateResult`
- `oracle.security.xmlsec.xkms.xkiss.ValidateRequest`
- `oracle.security.xmlsec.xkms.xkiss.ValidateResult`
- `oracle.security.xmlsec.xkms.xkrss.RecoverRequest`
- `oracle.security.xmlsec.xkms.xkrss.RecoverResult`

**oracle.security.xmlsec.xkms.xkiss.LocateRequest**

This class represents the XKMS LocateRequest element.

**Example 12–1 Creating an Instance of LocateRequest**

```java
// Parse the XML document containing the dsig:Signature.
Document sigDoc = //Instance of org.w3c.dom.Document;

//Create Query Key Binding
QueryKeyBinding queryKeyBinding = new QueryKeyBinding(sigDoc);
queryKeyBinding.setTimeInstant(new Date());
```
// Create the xkms:LocateRequest.
LocateRequest loc = new LocateRequest(sigDoc, queryKeyBinding);

Client requests of type LocateRequest must include an xkms:RespondWith attribute.
Example 12–2 shows how RespondWith can be added to a LocateRequest:

**Example 12–2  Adding RespondWith to a LocateRequest**

//Add xkms:RespondWith as X.509 Certificate.
loc.addRespondWith(XKMSURI.respondWith_X509Cert);

**oracle.security.xmlsec.xkms.xkiss.LocateResult**

This class represents the xkms:LocateResult element.
Example 12–3 shows how to create an instance of LocateResult:

**Example 12–3  Creating an Instance of LocateResult**

//Parse the XML document containin the dsig:Signature

// Create the xkms:LocateResult
LocateResult locRes = new LocateResult(sigDoc);

//Set ResultMajor to Success.
llocRes.setResultCode(XKMSURI.result_major_success, null);

If the LocateRequest contained a RespondWith attribute of X509Certificate, use the following code to add an X509 Certificate to the LocateResult:

**Example 12–4  Adding an X509 Certificate to LocateResult**

//Creating a signature and adding X509 certificate to the KeyInfo element.
X509 userCert = //Instance of oracle.security.crypto.cert.X509
XSSignature Sig = XSSignature.newInstance(sigDoc, "MySignature");
XSKeyInfo xsInfo = sig.getKeyInfo();
X509Data xData = xsInfo.createX509Data(userCert);

//Add X509Data to the KeyInfo
xsInfo.addKeyInfoData(xData);

//Set Key Binding and add KeyInfo the the KeyBinding
UnverifiedKeyBinding keyBinding = new UnverifiedKeyBinding(sigDoc);
keyBinding.setKeyInfo(xsInfo);

//Add Key Binding to LocateResult
locRes.addKeyBinding(keyBinding);

**oracle.security.xmlsec.xkms.xkiss.ValidateRequest**

This class represents the XKMS xkms:ValidateRequest element.
Example 12–5 shows how to create an instance of xkms:ValidateRequest:
Example 12–5  Creating an Instance of ValidateRequest

// Parse the XML document containing the dsig:Signature.
Document sigDoc = //Instance of org.w3c.dom.Document;

// Create Query Key Binding
QueryKeyBinding queryKeyBinding = new QueryKeyBinding(sigDoc);
queryKeyBinding.setTimeInstant(new Date());

// Create the xkms:ValidateRequest.
ValidateRequest validateReq = new ValidateRequest(sigDoc, queryKeyBinding);

Requests of type ValidateRequest must include an xkms:RespondWith attribute. Example 12–6 shows how to add RespondWith to a ValidateRequest:

Example 12–6  Adding RespondWith to a ValidateRequest

// Add xkms:RespondWith as X.509 Certificate.
validateReq.addRespondWith(XKMSURI.respondWith_X509Cert);

oracle.security.xmlsec.xkms.xkiss.ValidateResult

This class represents the XKMS ValidateResult element.

Example 12–7 shows how to create an instance of ValidateResult:

Example 12–7  Creating an Instance of ValidateResult

// Parse the XML document containing the dsig:Signature
Document sigDoc = //Instance of org.w3c.dom.Document;

// Create the xkms:ValidateResult
ValidateResult valRes = new ValidateResult(sigDoc);

// Set ResultMajor to Success.
valRes.setResultCode(XKMSURI.result_major_success, null);

Use the following code to set a status in response to a ValidateRequest:

Example 12–8  Setting a Response Status for a ValidateRequest

// Create a status element and add reasons.
Status responseStatus = new Status(sigDoc);
responseStatus.addValidReason(XKMSURI.reasonCode_IssuerTrust);
responseStatus.addValidReason(XKMSURI.reasonCode_RevocationStatus);
responseStatus.addValidReason(XKMSURI.reasonCode_ValidityInterval);
responseStatus.addValidReason(XKMSURI.reasonCode_Signature);

// Create a xkms:KeyBinding to add status and X509Data
XSKeyInfo xsInfo =
    // Instance of oracle.security.xmlsec.dsig.XSKeyInfo,
    // which contains X509Data
KeyBinding keyBinding = new KeyBinding(sigDoc);
keyBinding.setStatus(responseStatus);
keyBinding.setKeyInfo(xsInfo);

// Add the key binding to the ValidateResult.
valRes.addKeyBinding(keyBinding);
**oracle.security.xmlsec.xkms.xkrss.RecoverRequest**

This class represents the XKMS RecoverRequest element.

**Example 12–9** shows how to create an instance of RecoverRequest:

**Example 12–9  Creating an Instance of RecoverRequest**

// Parse the XML document containing the dsig:Signature.
Document sigDoc = //Instance of org.w3c.dom.Document;

// Create the xkms:RecoverRequest
RecoverRequest recReq = new RecoverRequest(sigDoc);

// Set RespondWith to PrivateKey, so that the RecoverResult contains the private key.
recReq.addRespondWith(XKMSURI.respondWith_PrivateKey);

A RecoverRequest must include the Authentication and RecoverKeyBinding elements. These can be added with the following code:

**Example 12–10  Adding Authentication and RecoverKeyBinding to a RecoverRequest**

// Create an instance of XSSignature.
XSSignature sig =
    // Instance of oracle.security.xmlsec.dsig.XSSignature

// Create an instance of Authentication element.
Authentication auth = new Authentication(sigDoc);

// Set key binding authentication.
auth.setKeyBindingAuthentication(sig);

// Set Authentication for the RecoverRequest.
recReq.setAuthentication(auth);

// Add RecoverKeyBinding to RecoverRequest.
RecoverKeyBinding recKeyBind = new RecoverKeyBinding(sigDoc);

// Add Key Info on the key to be recovered.
XSKeyInfo xsInfo =
    // Instance of oracle.security.xmlsec.dsig.XSKeyInfo
recKeyBind.setKeyInfo(xsInfo);

// Adding status, as known to the key holder, to the KeyBinding
Status keyStatus = new Status(sigDoc);
keyStatus.setStatusValue(XKMSURI.kbs_Indeterminate);
recKeyBind.setStatus(keyStatus);

// Adding RecoverKeyBinding to RecoverRequest.
recReq.setKeyBinding(recKeyBind);

**oracle.security.xmlsec.xkms.xkrss.RecoverResult**

This class represents the xkms:RecoverResult element.

**Example 12–11** shows how to create an instance of RecoverResult:
Example 12–11 Creating an Instance of xkms:RecoverResult

// Parse the XML document containing the dsig:Signature.
Document sigDoc = //Instance of org.w3c.dom.Document;

// Create the xkms:RecoverResult
RecoverResult recResult = new RecoverResult(sigDoc);

//Set ResultMajor to Success.
recResult.setResultCode(XKMSURI.result_major_success, null);

The KeyBinding needs to be set for a RecoverResult. You can accomplish this with the following code:

Example 12–12 Creating a Key Binding for a RecoverResult

//Create a xkms:KeyBinding to add status and X509Data
XSKeyInfo xsInfo =
   //Instance of oracle.security.xmlsec.dsig.XSKeyInfo,
   //which contains X509Data
KeyBinding keyBinding = new KeyBinding(sigDoc);
keyBinding.setKeyInfo(xsInfo);

//Create a status element and add reasons.
//Status is set to Invalid because the service can decide
//to revoke the key binding in the case of recovery.
Status responseStatus = new Status(sigDoc);
responseStatus.addInvalidReason(XKMSURI.reasonCode_IssuerTrust);
responseStatus.addInvalidReason(XKMSURI.reasonCode_RevocationStatus);
responseStatus.addInvalidReason(XKMSURI.reasonCode_ValidityInterval);
responseStatus.addInvalidReason(XKMSURI.reasonCode_Signature);
responseStatus.setStatusValue(XKMSURI.kbs_Invalid);
keyBinding.setStatus(responseStatus);

//Set KeyBinding into RecoverResult
recResult.addKeyBinding(keyBinding);

Finally, Example 12–13 shows how to set the recovered PrivateKey into the RecoverResult:

Example 12–13 Setting the Recovered Private Key into RecoverResult

//Create an Instance of dsig:XEEncryptedData
XEEncryptedData encryptedData = //Instance of
oracle.security.xmlsec.enc.XEEncryptedData

//Create an instance of oracle.security.xmlsec.xkms.xkrss.PrivateKey
PrivateKey privKey = new PrivateKey(sigDoc);
privKey.setEncryptedData(encryptedData);

//Add PrivateKey to RecoverResult
recResult.setPrivateKey(privKey);
Example Programs

For example programs using the Oracle Security Developer Tools, see the Oracle Technology Network Web Site at http://www.oracle.com/technology/index.html.
The following table lists the standards documents and protocols referenced in this document.

<table>
<thead>
<tr>
<th>Document</th>
<th>Reference</th>
</tr>
</thead>
</table>
### Table A–1  (Cont.) Security Standards and Protocols

<table>
<thead>
<tr>
<th>Document</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>Reference</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>xml.com</td>
<td>O'Reilly xml.com, <a href="http://www.xml.com/">http://www.xml.com/</a></td>
</tr>
</tbody>
</table>
3DES
See *Triple Data Encryption Standard (3DES)*.

**access control item (ACI)**
Access control information represents the permissions that various entities or subjects have to perform operations on a given object in the directory. This information is stored in Oracle Internet Directory as user-modifiable operational attributes, each of which is called an access control item (ACI). An ACI determines user access rights to directory data. It contains a set of rules for controlling access to entries (structural access items) and attributes (content access items). Access to both structural and content access items may be granted to one or more users or groups.

**access control list (ACL)**
A list of resources and the user names of people who are permitted access to those resources within a computer system. In Oracle Internet Directory, an ACL is a list of access control item (ACI) attribute values that is associated with directory objects. The attribute values on that list represent the permissions that various directory user entities (or subjects) have on a given object.

**access control policy point (ACP)**
A directory entry that contains access control policy information that applies downward to all entries at lower positions in the directory information tree (DIT). This information affects the entry itself and all entries below it. In Oracle Internet Directory, you can create ACPs to apply an access control policy throughout a subtree of your directory.

**account lockout**
A security feature that locks a user account if repeated failed logon attempts occur within a specified amount of time, based on security policy settings. Account lockout occurs in OracleAS Single Sign-On when a user submits an account and password combination from any number of workstations more times than is permitted by Oracle Internet Directory. The default lockout period is 24 hours.

**ACI**
See *access control item (ACI)*.

**ACL**
See *access control list (ACL)*.
ACP
See access control policy point (ACP).

administrative area
A subtree on a directory server whose entries are under the control of a single administrative authority. The designated administrator controls each entry in that administrative area, as well as the directory schema, access control list (ACL), and attributes for those entries.

Advanced Encryption Standard (AES)
Advanced Encryption Standard (AES) is a symmetric cryptography algorithm that is intended to replace Data Encryption Standard (DES). AES is a Federal Information Processing Standard (FIPS) for the encryption of commercial and government data.

advanced replication
See Oracle Database Advanced Replication.

advanced symmetric replication (ASR)
See Oracle Database Advanced Replication.

AES
See Advanced Encryption Standard (AES).

anonymous authentication
The process by which a directory authenticates a user without requiring a user name and password combination. Each anonymous user then exercises the privileges specified for anonymous users.

API
See application programming interface (API).

application programming interface (API)
A series of software routines and development tools that comprise an interface between a computer application and lower-level services and functions (such as the operating system, device drivers, and other software applications). APIs serve as building blocks for programmers putting together software applications. For example, LDAP-enabled clients access Oracle Internet Directory information through programmatic calls available in the LDAP API.

application service provider
Application Service Providers (ASPs) are third-party entities that manage and distribute software-based services and solutions to customers across a wide area network from a central data center. In essence, ASPs are a way for companies to outsource some or almost all aspects of their information technology needs.

artifact profile
An authentication mechanism which transmits data using a compact reference to an assertion, called an artifact, instead of sending the full assertion. This profile accommodates browsers which handle a limited number of characters.

ASN.1
Abstract Syntax Notation One (ASN.1) is an International Telecommunication Union (ITU) notation used to define the syntax of information data. ASN.1 is used to describe
structured information, typically information that is to be conveyed across some communications medium. It is widely used in the specification of Internet protocols.

**ASR**

See Oracle Database Advanced Replication.

**assertion**

An assertion is a statement used by providers in security domains to exchange information about a subject seeking access to a resource. Identity providers, as well as service providers, exchange assertions about identities to make authentication and authorization decisions, and to determine and enforce security policies protecting the resource.

**asymmetric algorithm**

A cryptographic algorithm that uses different keys for encryption and decryption. See also: public key cryptography.

**asymmetric cryptography**

See public key cryptography.

**attribute**

Directory attributes hold a specific data element such as a name, phone number, or job title. Each directory entry is comprised of 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.

**attribute configuration file**

In an Oracle Directory Integration Platform environment, a file that specifies attributes of interest in a connected directory.

**attribute type**

Attribute types specify information about a data element, such as the data type, maximum length, and whether it is single-valued or multivalued. The attribute type provides the real-world meaning for a value, and specifies the rules for creating and storing specific pieces of data, such as a name or an e-mail address.

**attribute uniqueness**

An Oracle Internet Directory feature that ensures that no two specified attributes have the same value. It enables applications synchronizing with the enterprise directory to use attributes as unique keys.

**attribute value**

Attribute values are the actual data contained within an attribute for a particular entry. For example, for the attribute type email, an attribute value might be sally.jones@oracle.com.

**authentication**

The process of verifying the identity claimed by an entity based on its credentials. Authentication of a user is generally based on something the user knows or has (for example, a password or a certificate).

Authentication of an electronic message involves the use of some kind of system (such as public key cryptography) to ensure that a file or message which claims to originate
from a given individual or company actually does, and a check based on the contents of a message to ensure that it was not modified in transit.

authentication level
An OracleAS Single Sign-On parameter that enables you to specify a particular authentication behavior for an application. You can link this parameter with a specific authentication plugin.

authentication plugin
An implementation of a specific authentication method. OracleAS Single Sign-On has Java plugins for password authentication, digital certificates, Windows native authentication, and third-party access management.

authorization
The process of granting or denying access to a service or network resource. Most security systems are based on a two step process. The first stage is authentication, in which a user proves his or her identity. The second stage is authorization, in which a user is allowed to access various resources based on his or her identity and the defined authorization policy.

authorization policy
Authorization policy describes how access to a protected resource is governed. Policy maps identities and objects to collections of rights according to some system model. For example, a particular authorization policy might state that users can access a sales report only if they belong to the sales group.

basic authentication
An authentication protocol supported by most browsers in which a Web server authenticates an entity with an encoded user name and password passed via data transmissions. Basic authentication is sometimes called plaintext authentication because the base-64 encoding can be decoded by anyone with a freely available decoding utility. Note that encoding is not the same as encryption.

Basic Encoding Rules (BER)
Basic Encoding Rules (BER) are the standard rules for encoding data units set forth in ASN.1. BER is sometimes incorrectly paired with ASN.1, which applies only to the abstract syntax description language, not the encoding technique.

BER
See Basic Encoding Rules (BER).

binding
In networking, binding is the establishment of a logical connection between communicating entities.

In the case of Oracle Internet Directory, binding refers to the process of authenticating to the directory.

The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange is also called a binding.

block cipher
Block ciphers are a type of symmetric algorithm. A block cipher encrypts a message by breaking it down into fixed-size blocks (often 64 bits) and encrypting each block with a key. Some well known block ciphers include Blowfish, DES, and AES.
See also: *stream cipher*.

**Blowfish**

Blowfish is a *symmetric cryptography* algorithm developed by Bruce Schneier in 1993 as a faster replacement for *DES*. It is a *block cipher* using 64-bit blocks and keys of up to 448 bits.

**CA**

See *Certificate Authority (CA)*.

**CA certificate**

A *Certificate Authority (CA)* signs all certificates that it issues with its *private key*. The corresponding Certificate Authority’s *public key* is itself contained within a certificate, called a CA Certificate (also referred to as a root certificate). A browser must contain the CA Certificate in its list of trusted root certificates in order to trust messages signed by the CA’s private key.

**cache**

Generally refers to an amount of quickly accessible memory in your computer. However, on the Web it more commonly refers to where the browser stores downloaded files and graphics on the user’s computer.

**CBC**

See *cipher block chaining (CBC).*

**central directory**

In an Oracle Directory Integration Platform environment, the directory that acts as the central repository. In an Oracle Directory Integration Platform environment, Oracle Internet Directory is the central directory.

**certificate**

A certificate is a specially formatted data structure that associates a *public key* with the identity of its owner. A certificate is issued by a *Certificate Authority (CA)*. It contains the name, serial number, expiration dates, and public key of a particular entity. The certificate is digitally signed by the issuing CA so that a recipient can verify that the certificate is real. Most digital certificates conform to the *X.509* standard.

**Certificate Authority (CA)**

A Certificate Authority (CA) is a trusted third party that issues, renews, and revokes digital *certificates*. The CA essentially vouches for a entity’s identity, and may delegate the verification of an applicant to a *Registration Authority (RA)*. Some well known Certificate Authorities (CAs) include Digital Signature Trust, Thawte, and VeriSign.

**certificate chain**

An ordered list of certificates containing one or more pairs of a user *certificate* and its associated *CA certificate*.

**certificate management protocol (CMP)**

Certificate Management Protocol (CMP) handles all relevant aspects of certificate creation and management. CMP supports interactions between *public key infrastructure (PKI)* components, such as the *Certificate Authority (CA)*, *Registration Authority (RA)*, and the user or application that is issued a certificate.
**certificate request message format (CRMF)**
Certificate Request Message Format (CRMF) is a format used for messages related to the life-cycle management of X.509 certificates, as described in the RFC 2511 specification.

**certificate revocation list (CRL)**
A Certificate Revocation List (CRL) is a list of digital certificates which have been revoked by the Certificate Authority (CA) that issued them.

**change logs**
A database that records changes made to a directory server.

**cipher**
See cryptographic algorithm.

**cipher block chaining (CBC)**
Cipher block chaining (CBC) is a mode of operation for a block cipher. CBC uses what is known as an initialization vector (IV) of a certain length. One of its key characteristics is that it uses a chaining mechanism that causes the decryption of a block of ciphertext to depend on all the preceding ciphertext blocks. As a result, the entire validity of all preceding blocks is contained in the immediately previous ciphertext block.

**cipher suite**
In Secure Sockets Layer (SSL), a set of authentication, encryption, and data integrity algorithms used for exchanging messages between network nodes. During an SSL handshake, the two nodes negotiate to see which cipher suite they will use when transmitting messages back and forth.

**ciphertext**
Ciphertext is the result of applying a cryptographic algorithm to readable data (plaintext) in order to render the data unreadable by all entities except those in possession of the appropriate key.

**circle of trust**
A trust relationship among a set of identity providers and service providers that allows a principal to use a single federated identity and single sign-on (SSO) when conducting business transactions with providers within that set.

Businesses federate or affiliate together into circles of trust based on Liberty-enabled technology and on operational agreements that define trust relationships between the businesses.

See also: federated identity management (FIM), Liberty Alliance.

**claim**
A claim is a declaration made by an entity (for example, a name, identity, key, group, and so on).

**client SSL certificates**
A type of certificate used to identify a client machine to a server through Secure Sockets Layer (SSL) (client authentication).
cluster
A collection of interconnected usable whole computers that is used as a single computing resource. Hardware clusters provide high availability and scalability.

CMP
See certificate management protocol (CMP).

CMS
See Cryptographic Message Syntax (CMS).

code signing certificates
A type of certificate used to identify the entity who signed a Java program, Java Script, or other signed file.

cold backup
In Oracle Internet Directory, this refers to the procedure of adding a new directory system agent (DSA) node to an existing replicating system by using the database copy procedure.

concurrency
The ability to handle multiple requests simultaneously. Threads and processes are examples of concurrency mechanisms.

concurrent clients
The total number of clients that have established a session with Oracle Internet Directory.

concurrent operations
The number of operations that are being executed on Oracle Internet Directory from all of the concurrent clients. Note that this is not necessarily the same as the concurrent clients, because some of the clients may be keeping their sessions idle.

confidentiality
In cryptography, confidentiality (also known as privacy) is the ability to prevent unauthorized entities from reading data. This is typically achieved through encryption.

configset
See configuration set entry.

configuration set entry
An Oracle Internet Directory entry holding the configuration parameters for a specific instance of the directory server. Multiple configuration set entries can be stored and referenced at runtime. The configuration set entries are maintained in the subtree specified by the subConfigSubEntry attribute of the directory-specific entry (DSE), which itself resides in the associated directory information base (DIB) against which the servers are started.

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 the Oracle Database or its Oracle System Identifier (SID) for Oracle release 8.0 or version 7 databases. The
network route provides, at a minimum, the location of the listener through use of a network address.

**connected directory**
In an Oracle Directory Integration Platform environment, an information repository requiring full synchronization of data between Oracle Internet Directory and itself—for example, an Oracle human resources database.

**consumer**
A directory server that is the destination of replication updates. Sometimes called a slave.

**contention**
Competition for resources.

**context prefix**
The distinguished name (DN) of the root of a naming context.

**CRL**
See certificate revocation list (CRL).

**CRMF**
See certificate request message format (CRMF).

**cryptographic algorithm**
A cryptographic algorithm is a defined sequence of processes to convert readable data (plaintext) to unreadable data (ciphertext) and vice versa. These conversions require some secret knowledge, normally contained in a key. Examples of cryptographic algorithms include DES, AES, Blowfish, and RSA.

**Cryptographic Message Syntax (CMS)**
Cryptographic Message Syntax (CMS) is a syntax defined in RFC 3369 for signing, digesting, authenticating, and encrypting digital messages.

**cryptography**
The process of protecting information by transforming it into an unreadable format. The information is encrypted using a key, which makes the data unreadable, and is then decrypted later when the information needs to be used again. See also public key cryptography and symmetric cryptography.

**dads.conf**
A configuration file for Oracle HTTP Server that is used to configure a database access descriptor (DAD).

**DAS**
See Oracle Delegated Administration Services. (DAS).

**Data Encryption Standard (DES)**
Data Encryption Standard (DES) is a widely used symmetric cryptography algorithm developed in 1974 by IBM. It applies a 56-bit key to each 64-bit block of data. DES and 3DES are typically used as encryption algorithms by S/MIME.
data integrity
The guarantee that the contents of the message received were not altered from the contents of the original message sent.

See also: integrity.

database access descriptor (DAD)
Database connection information for a particular Oracle Application Server component, such as the OracleAS Single Sign-On schema.

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

default identity management realm
In a hosted environment, one enterprise—for example, an application service provider—makes Oracle components available to multiple other enterprises and stores information for them. In such hosted environments, the enterprise performing the hosting is called the default identity management realm, and the enterprises that are hosted are each associated with their own identity management realm in the directory information tree (DIT).

default knowledge reference
A knowledge reference that is returned when the base object is not in the directory, and the operation is performed in a naming context not held locally by the server. A default knowledge reference typically sends the user to a server that has more knowledge about the directory partitioning arrangement.

default realm location
An attribute in the root Oracle Context that identifies the root of the default identity management realm.

defederation
The act of unlinking a user’s account from an identity provider or service provider.

Delegated Administration Services
See Oracle Delegated Administration Services.

degenerate administrator
In a hosted environment, one enterprise—for example, an application service provider—makes Oracle components available to multiple other enterprises and stores information for them. In such an environment, a global administrator performs activities that span the entire directory. Other administrators—called delegated administrators—may exercise roles in specific identity management realms, or for specific applications.

DER
See Distinguished Encoding Rules (DER).

DES
See Data Encryption Standard (DES).

DIB
See directory information base (DIB).
**Diffie-Hellman**

Diffie-Hellman (DH) is a public key cryptography protocol that allows two parties to establish a shared secret over an unsecure communications channel. First published in 1976, it was the first workable public key cryptographic system.

See also: **symmetric algorithm**.

**digest**

See **message digest**.

**digital certificate**

See **certificate**.

**digital signature**

A digital signature is the result of a two-step process applied to a given block of data. First, a **hash function** is applied to the data to obtain a result. Second, that result is encrypted using the signer’s **private key**. Digital signatures can be used to ensure integrity, message authentication, and non-repudiation of data. Examples of digital signature algorithms include **DSA**, **RSA**, and **ECDSA**.

**Digital Signature Algorithm (DSA)**

The Digital Signature Algorithm (DSA) is an **asymmetric algorithm** that is used as part of the Digital Signature Standard (DSS). It cannot be used for encryption, only for digital signatures. The algorithm produces a pair of large numbers that enable the authentication of the signatory, and consequently, the integrity of the data attached. DSA is used both in generating and verifying digital signatures.

See also: **Elliptic Curve Digital Signature Algorithm (ECDSA)**.

**directory**

See **Oracle Internet Directory**, **Lightweight Directory Access Protocol (LDAP)**, and **X.500**.

**directory information base (DIB)**

The complete set of all information held in the directory. The DIB consists of entries that are related to each other hierarchically in a **directory information tree (DIT)**.

**directory information tree (DIT)**

A hierarchical tree-like structure consisting of the **DNs** of the entries.

**directory integration and provisioning server**

In an Oracle Directory Integration Platform environment, the server that drives the synchronization of data between Oracle Internet Directory and a **connected directory**.

**directory integration profile**

In an Oracle Directory Integration Platform environment, an entry in Oracle Internet Directory that describes how Oracle Directory Integration Platform communicates with external systems and what is communicated.

**Directory Manager**

See **Oracle Directory Manager**.

**directory naming context**

See **naming context**.
**directory provisioning profile**
A special kind of **directory integration profile** that describes the nature of provisioning-related notifications that Oracle Directory Integration Platform sends to the directory-enabled applications.

**directory replication group (DRG)**
The directory servers participating in a **replication agreement**.

**directory server instance**
A discrete invocation of a directory server. Different invocations of a directory server, each started with the same or different configuration set entries and startup flags, are said to be different directory server instances.

**directory synchronization profile**
A special kind of **directory integration profile** that describes how synchronization is carried out between Oracle Internet Directory and an external system.

**directory system agent (DSA)**
The **X.500** term for a directory server.

**directory-specific entry (DSE)**
An entry specific to a directory server. Different directory servers may hold the same **directory information tree (DIT)** name, but have different contents—that is, the contents can be specific to the directory holding it. A DSE is an entry with contents specific to the directory server holding it.

**directory user agent (DUA)**
The software that accesses a directory service on behalf of the directory user. The directory user may be a person or another software element.

**DIS**
See **directory integration and provisioning server**.

**Distinguished Encoding Rules (DER)**
Distinguished Encoding Rules (DER) are a set of rules for encoding **ASN.1** objects in byte-sequences. DER is a special case of **Basic Encoding Rules (BER)**.

**distinguished name (DN)**
A **X.500** distinguished name (DN) is a unique name for a node in a directory tree. A DN is used to provide a unique name for a person or any other directory entry. A DN is a concatenation of selected attributes from each node in the tree along the path from the root node to the named entry’s node. For example, in LDAP notation, the DN for a person named John Smith working at Oracle’s US office would be: "cn=John Smith, ou=People, o=Oracle, c=us".

**DIT**
See **directory information tree (DIT)**.

**DN**
See **distinguished name (DN)**.
Document Type Definition (DTD)
A Document Type Definition (DTD) is a document that specifies constraints on the
tags and tag sequences that are valid for a given XML document. DTDs follow the
rules of Simple Generalized Markup Language (SGML), the parent language of XML.

domain
A domain includes the Web site and applications that enable a principal to utilize
resources. A federated site acts as an identity provider (also known as the source
domain), a service provider (also known as the destination domain), or both.

domain component attribute
The domain component (dc) attribute can be used in constructing a distinguished
name (DN) from a domain name. For example, using a domain name such as
"oracle.com", one could construct a DN beginning with "dc=oracle, dc=com", and then
use this DN as the root of its subtree of directory information.

DRG
See directory replication group (DRG).

DSA
See Digital Signature Algorithm (DSA) or directory system agent (DSA).

DSE
See directory-specific entry (DSE).

DTD
See Document Type Definition (DTD).

ECC
See Elliptic Curve Cryptography (ECC).

ECDSA
See Elliptic Curve Digital Signature Algorithm (ECDSA).

EJB
See Enterprise Java Bean (EJB).

Elliptic Curve Cryptography (ECC)
Elliptic Curve Cryptography (ECC) is an alternative to the RSA encryption system
which is based on the difficulty of solving elliptic curve discrete logarithm problems
rather than on factoring large numbers. Developed and marketed by Certicom, ECC is
especially suitable for environments, such as wireless devices and PC cards, where
computational power is limited and high speed is required. For any given key size
(measured in bits) ECC provides more security (is harder to decrypt without the key)
than RSA.

Elliptic Curve Digital Signature Algorithm (ECDSA)
The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analog of
the Digital Signature Algorithm (DSA) standard. The advantages of ECDSA
compared to RSA-like schemes are shorter key lengths and faster signing and
decryption. For example, a 160 (210) bit ECC key is expected to give the same security
as a 1024 (2048) bit RSA key, and the advantage increases as level of security is raised.
**encryption**
Encryption is the process of converting plaintext to ciphertext by applying a cryptographic algorithm.

**encryption certificate**
An encryption certificate is a certificate containing a public key that is used to encrypt electronic messages, files, documents, or data transmission, or to establish or exchange a session key for these same purposes.

**end-to-end security**
This is a property of message-level security that is established when a message traverses multiple applications within and between business entities and is secure over its full route through and between the business entities.

**Enterprise Java Bean (EJB)**
Enterprise JavaBeans (EJBs) are a Java API developed by Sun Microsystems that defines a component architecture for multi-tier client/server systems. Because EJB systems are written in Java, they are platform independent. Being object oriented, they can be implemented into existing systems with little or no recompiling and configuring.

**Enterprise Manager**
See Oracle Enterprise Manager.

**entry**
An entry is a unique record in a directory that describes an object, such as a person. An entry consists of attributes and their associated attribute values, as dictated by the object class that describes that entry object. All entries in an LDAP directory structure are uniquely identified through their distinguished name (DN).

**export agent**
In an Oracle Directory Integration Platform environment, an agent that exports data out of Oracle Internet Directory.

**export data file**
In an Oracle Directory Integration Platform environment, the file that contains data exported by an export agent.

**export file**
See export data file.

**external agent**
A directory integration agent that is independent of Oracle Directory Integration Platform server. Oracle Directory Integration Platform server does not provide scheduling, mapping, or error handling services for it. An external agent is typically used when a third party metadirectory solution is integrated with Oracle Directory Integration Platform.

**external application**
Applications that do not delegate authentication to the OracleAS Single Sign-On server. Instead, they display HTML login forms that ask for application user names and passwords. At the first login, users can choose to have the OracleAS Single
Sign-On server retrieve these credentials for them. Thereafter, they are logged in to these applications transparently.

**failover**

The process of failure recognition and recovery. In an Oracle Application Server Cold Failover Cluster (Identity Management), an application running on one cluster node is transparently migrated to another cluster node. During this migration, clients accessing the service on the cluster see a momentary outage and may need to reconnect once the failover is complete.

**fan-out replication**

Also called a point-to-point replication, a type of replication in which a supplier replicates directly to a consumer. That consumer can then replicate to one or more other consumers. The replication can be either full or partial.

**Federal Information Processing Standards (FIPS)**

Federal Information Processing Standards (FIPS) are standards for information processing issued by the US government Department of Commerce's National Institute of Standards and Technology (NIST).

**federated identity management (FIM)**

The agreements, standards, and technologies that make identity and entitlements portable across autonomous domains. FIM makes it possible for an authenticated user to be recognized and take part in personalized services across multiple domains. It avoids pitfalls of centralized storage of personal information, while allowing users to link identity information between different accounts. Federated identity requires two key components: trust and standards. The trust model of federated identity management is based on circle of trust. The standards are defined by the Liberty Alliance Project.

**federation**

See identity federation.

**filter**

A filter is an expression that defines the entries to be returned from a request or search on a directory. Filters are typically expressed as DNs, for example: `cn=susie smith,o=acme,c=us`.

**FIM**

See federated identity management (FIM).

**FIPS**

See Federal Information Processing Standards (FIPS).

**forced authentication**

The act of forcing a user to reauthenticate if he or she has been idle for a preconfigured amount of time. Oracle Application Server Single Sign-On enables you to specify a global user inactivity timeout. This feature is intended for installations that have sensitive applications.

**GET**

An authentication method whereby login credentials are submitted as part of the login URL.
global administrator
In a hosted environment, one enterprise—for example, an application service provider—makes Oracle components available to multiple other enterprises and stores information for them. In such an environment, a global administrator performs activities that span the entire directory.

global unique identifier (GUID)
An identifier generated by the system and inserted into an entry when the entry is added to the directory. In a multimaster replicated environment, the GUID, not the DN, uniquely identifies an entry. The GUID of an entry cannot be modified by a user.

global user inactivity timeout
An optional feature of Oracle Application Server Single Sign-On that forces users to reauthenticate if they have been idle for a preconfigured amount of time. The global user inactivity timeout is much shorter than the single sign-out session timeout.

globalization support
Multilanguage support for graphical user interfaces. Oracle Application Server Single Sign-On supports 29 languages.

globally unique user ID
A numeric string that uniquely identifies a user. A person may change or add user names, passwords, and distinguished names, but her globally unique user ID always remains the same.

grace login
A login occurring within the specified period before password expiration.

group search base
In the Oracle Internet Directory default directory information tree (DIT), the node in the identity management realm under which all the groups can be found.

guest user
One who is not an anonymous user, and, at the same time, does not have a specific user entry.

GUID
See global unique identifier (GUID).

handshake
A protocol two computers use to initiate a communication session.

hash
A number generated from a string of text with an algorithm. The hash value is substantially smaller than the text itself. Hash numbers are used for security and for faster access to data.
See also: hash function.

hash function
In cryptography, a hash function or one-way hash function is an algorithm that produces a given value when applied to a given block of data. The result of a hash function can be used to ensure the integrity of a given block of data. For a hash
function to be considered secure, it must be very difficult, given a known data block and a known result, to produce another data block that produces the same result.

**Hashed Message Authentication Code (HMAC)**

Hashed Message Authentication Code (HMAC) is a hash function technique used to create a secret hash function output. This strengthens existing hash functions such as MD5 and SHA. It is used in transport layer security (TLS).

**HMAC**

See [Hashed Message Authentication Code (HMAC)].

**HTTP**

The Hyper Text Transfer Protocol (HTTP) is the protocol used between a Web browser and a server to request a document and transfer its contents. The specification is maintained and developed by the World Wide Web Consortium.

**HTTP Redirect Profile**

A [federation profile] which indicates that the requested resource resides under a different URL.

**HTTP Server**

See [Oracle HTTP Server].

**httpd.conf**

The file used to configure [Oracle HTTP Server].

**iASAdmins**

The administrative group responsible for user and group management functions in Oracle Application Server. The OracleAS Single Sign-On administrator is a member of the group iASAdmins.

**identity federation**

The linking of two or more accounts a [principal] may hold with one or more identity providers or service providers within a given [circle of trust].

When users federate the otherwise isolated accounts they have with businesses, known as their local identities, they create a relationship between two entities, an association comprising any number of service providers and identity providers.

See also: [identity provider], [service provider].

**identity management**

The process by which the complete security lifecycle for network entities is managed in an organization. It typically refers to the management of an organization’s application users, where steps in the security life cycle include account creation, suspension, privilege modification, and account deletion. The network entities managed may also include devices, processes, applications, or anything else that needs to interact in a networked environment. Entities managed by an identity management process may also include users outside of the organization, for example customers, trading partners, or Web services.

**identity management infrastructure database**

The database that contains data for OracleAS Single Sign-On and Oracle Internet Directory.
**Identity Management Realm**
A collection of identities, all of which are governed by the same administrative policies. In an enterprise, all employees having access to the intranet may belong to one realm, while all external users who access the public applications of the enterprise may belong to another realm. An identity management realm is represented in the directory by a specific **entry** with a special **object class** associated with it.

**Identity Management Realm-Specific Oracle Context**
An Oracle Context contained in each identity management realm. It stores the following information:

- User naming policy of the identity management realm—that is, how users are named and located.
- Mandatory authentication attributes.
- Location of groups in the identity management realm.
- Privilege assignments for the identity management realm—for example: who has privileges to add more users to the realm.
- Application specific data for that realm including authorizations.

**Identity Provider**
One of the three primary roles defined in the **identity federation** protocols supported by Oracle Identity Federation. The other primary roles are **service provider** and **principal**. The identity provider is responsible for managing and authenticating a set of identities within a given **circle of trust**.

A service provider, in turn, provides services or goods to a principal based on the identity provider’s authentication of a principal’s identity.

Identity providers are service providers offering business incentives so that other service providers affiliate with them. An identity provider typically authenticates and asserts a principal’s identity.

**Import Agent**
In an Oracle Directory Integration Platform environment, an agent that imports data into Oracle Internet Directory.

**Import Data File**
In an Oracle Directory Integration Platform environment, the file containing the data imported by an **import agent**.

**Infrastructure Tier**
The Oracle Application Server components responsible for identity management. These components are OracleAS Single Sign-On, Oracle Delegated Administration Services, and Oracle Internet Directory.

**Inherit**
When an **object class** has been derived from another class, it also derives, or inherits, many of the characteristics of that other class. Similarly, an attribute subtype inherits the characteristics of its supertype.

**Instance**
See **directory server instance**.
integrity
In cryptography, integrity is the ability to detect if data has been modified by entities that are not authorized to modify it.

Internet Directory
See Oracle Internet Directory.

Internet Engineering Task Force (IETF)
The principal body engaged in the development of new Internet standard specifications. It is an international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

Internet Message Access Protocol (IMAP)
A protocol allowing a client to access and manipulate electronic mail messages on a server. It permits manipulation of remote message folders, also called mailboxes, in a way that is functionally equivalent to local mailboxes.

J2EE
See Java 2 Platform, Enterprise Edition (J2EE).

Java 2 Platform, Enterprise Edition (J2EE)
Java 2 Platform, Enterprise Edition (J2EE) is an environment for developing and deploying enterprise applications, defined by Sun Microsystems Inc. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, Web-based applications.

Java Server Page (JSP)
JavaServer Pages (JSP), a server-side technology, are an extension to the Java servlet technology that was developed by Sun Microsystems. JSPs have dynamic scripting capability that works in tandem with HTML code, separating the page logic from the static elements (the design and display of the page). Embedded in the HTML page, the Java source code and its extensions help make the HTML more functional, being used in dynamic database queries, for example.

JSP
See Java Server Page (JSP).

key
A key is a data structure that contains some secret knowledge necessary to successfully encrypt or decrypt a given block of data. The larger the key, the harder it is to crack a block of encrypted data. For example, a 256-bit key is more secure than a 128-bit key.

key pair
A public key and its associated private key.
See also: public/private key pair.

knowledge reference
The access information (name and address) for a remote directory system agent (DSA) and the name of the directory information tree (DIT) subtree that the remote DSA holds. Knowledge references are also called referrals.
**latency**
The time a client has to wait for a given directory operation to complete. Latency can be defined as wasted time. In networking discussions, latency is defined as the travel time of a packet from source to destination.

**LDAP**

**LDAP connection cache**
To improve throughput, the OracleAS Single Sign-On server caches and then reuses connections to Oracle Internet Directory.

**LDAP Data Interchange Format (LDIF)**
A common, text-based format for exchanging directory data between systems. The set of standards for formatting an input file for any of the LDAP command-line utilities.

**LDIF**
See LDAP Data Interchange Format (LDIF).

**legacy application**
Older application that cannot be modified to delegate authentication to the OracleAS Single Sign-On server. Also known as an external application.

**Liberty Alliance**
The Liberty Alliance Project is a consortium of companies, non-profits, and non-government organizations around the globe. It is committed to developing an open standard for federated identity management (FIM) and identity-based Web services supporting current and emerging network devices.

**Liberty ID-FF**
Liberty Identity Federation Framework (Liberty ID-FF) provides an architecture for Web-based single sign-on (SSO) with federated identities.

**Lightweight Directory Access Protocol (LDAP)**
A set of protocols for accessing information in directories. LDAP supports TCP/IP, which is necessary for any type of Internet access. Its framework of design conventions supports industry-standard directory products, such as Oracle Internet Directory. Because it is a simpler version of the X.500 standard, LDAP is sometimes called X.500 light.

**load balancer**
Hardware devices and software that balance connection requests between two or more servers, either due to heavy load or failover. BigIP, Alteon, or Local Director are all popular hardware devices. Oracle Application Server Web Cache is an example of load balancing software.

**logical host**
In an Oracle Application Server Cold Failover Cluster (Identity Management), one or more disk groups and pairs of host names and IP addresses. It is mapped to a physical host in the cluster. This physical host impersonates the host name and IP address of the logical host.
MAC

See message authentication code (MAC).

man-in-the-middle

A security attack characterized by the third-party, surreptitious interception of a message. The third-party, the man-in-the-middle, decrypts the message, re-encrypts it (with or without alteration of the original message), and retransmits 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.

mapping rules file

In an Oracle Directory Integration Platform environment, the file that specifies mappings between Oracle Internet Directory attributes and those in a connected directory.

master definition site (MDS)

In replication, a master definition site is the Oracle Internet Directory database from which the administrator runs the configuration scripts.

master site

In replication, a master site is any site other than the master definition site (MDS) that participates in LDAP replication.

matching rule

In a search or compare operation, determines equality between the attribute value sought and the attribute value stored. For example, matching rules associated with the telephoneNumber attribute could cause "(650) 123-4567" to be matched with either "(650) 123-4567" or "6501234567" or both. When you create an attribute, you associate a matching rule with it.

MD2

Message Digest Two (MD2) is a message digest hash function. The algorithm processes input text and creates a 128-bit message digest which is unique to the message and can be used to verify data integrity. MD2 was developed by Ron Rivest for RSA Security and is intended to be used in systems with limited memory, such as smart cards.

MD4

Message Digest Four (MD4) is similar to MD2 but designed specifically for fast processing in software.

MD5

Message Digest Five (MD5) is a message digest hash function. The algorithm processes input text and creates a 128-bit message digest which is unique to the message and can be used to verify data integrity. MD5 was developed by Ron Rivest after potential weaknesses were reported in MD4. MD5 is similar to MD4 but slower because more manipulation is made to the original data.

MDS

See master definition site (MDS).

message authentication

The process of verifying that a particular message came from a particular entity.
See also: **authentication**.

**message authentication code (MAC)**
The Message Authentication Code (MAC) is a result of a two-step process applied to a given block of data. First, the result of a **hash function** is obtained. Second, that result is encrypted using a **secret key**. The MAC can be used to authenticate the source of a given block of data.

**message digest**
The result of a **hash function**.
See also: **hash**.

**metadirectory**
A directory solution that shares information between all enterprise directories, integrating them into one virtual directory. It centralizes administration, thereby reducing administrative costs. It synchronizes data between directories, thereby ensuring that it is consistent and up-to-date across the enterprise.

**middle tier**
That portion of an OracleAS Single Sign-On instance that consists of the Oracle HTTP Server and OC4J. The OracleAS Single Sign-On middle tier is situated between the identity management infrastructure database and the client.

**mod_osso**
A module on the Oracle HTTP Server that enables applications protected by OracleAS Single Sign-On to accept HTTP headers in lieu of a user name and password once the user has logged into the OracleAS Single Sign-On server. The values for these headers are stored in the **mod_osso cookie**.

**mod_osso cookie**
User data stored on the HTTP server. The cookie is created when a user authenticates. When the same user requests another application, the Web server uses the information in the mod_osso cookie to log the user in to the application. This feature speeds server response time.

**mod_proxy**
A module on the Oracle HTTP Server that makes it possible to use **mod_osso** to enable single sign-on to legacy, or **external applications**.

**MTS**
See **shared server**.

**multimaster replication**
Also called peer-to-peer or n-way replication, a type of replication that enables multiple sites, acting as equals, to manage groups of replicated data. In a multimaster replication environment, each node is both a supplier and a consumer node, and the entire directory is replicated on each node.

**name identifier profile**
A **federation** profile which allows a provider to inform it’s peers when assigning or updating a name identifier for one of their common users.
naming attribute
The attribute used to compose the RDN of a new user entry created through Oracle Delegated Administration Services or Oracle Internet Directory Java APIs. The default value for this is \textit{cn}.

naming context
A subtree that resides entirely on one server. It must be contiguous, that is, it must begin at an entry that serves as the top of the subtree, and extend downward to either leaf entries or knowledge references (also called referrals) to subordinate naming contexts. It can range in size from a single entry to the entire directory information tree (DIT).

native agent
In an Oracle Directory Integration Platform environment, an agent that runs under the control of the directory integration and provisioning server. It is in contrast to an external agent.

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 wish to connect, for example:

\texttt{CONNECT \textit{username/password@net_service_name}}

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

- Local configuration file, \texttt{tnsnames.ora}, on each client
- Directory server
- Oracle Names server
- External naming service, such as NDS, NIS or CDS

Net Services
See Oracle Net Services.

nickname attribute
The attribute used to uniquely identify a user in the entire directory. The default value for this is \textit{uid}. Applications use this to resolve a simple user name to the complete distinguished name. The user nickname attribute cannot be multi-valued—that is, a given user cannot have multiple nicknames stored under the same attribute name.

non-repudiation
In cryptography, the ability to prove that a given digital signature was produced with a given entity’s private key, and that a message was sent untampered at a given point in time.

OASIS
Organization for the Advancement of Structured Information Standards. OASIS is a worldwide not-for-profit consortium that drives the development, convergence and adoption of e-business standards.
object class
In LDAP, object classes are used to group information. Typically an object class models a real-world object such as a person or a server. Each directory entry belongs to one or more object classes. The object class determines the attributes that make up an entry. One object class can be derived from another, thereby inheriting some of the characteristics of the other class.

OC4J
See Oracle Containers for J2EE (OC4J).

OCA
See Oracle Certificate Authority.

OCI
See Oracle Call Interface (OCI).

OCSP
See Online Certificate Status Protocol (OCSP).

OEM
See Oracle Enterprise Manager.

OID
See Oracle Internet Directory.

OID Control Utility
A command-line tool for issuing run-server and stop-server commands. The commands are interpreted and executed by the OID Monitor process.

OID Database Password Utility
The utility used to change the password with which Oracle Internet Directory connects to an Oracle Database.

OID Monitor
The Oracle Internet Directory component that initiates, monitors, and terminates the Oracle Internet Directory Server processes. It also controls the replication server if one is installed, and Oracle Directory Integration Platform Server.

Online Certificate Status Protocol (OCSP)
Online Certificate Status Protocol (OCSP) is one of two common schemes for checking the validity of digital certificates. The other, older method, which OCSP has superseded in some scenarios, is certificate revocation list (CRL). OCSP is specified in RFC 2560.

one-way function
A function that is easy to compute in one direction but quite difficult to reverse compute, that is, to compute in the opposite direction.

one-way hash function
A one-way function that takes a variable sized input and creates a fixed size output. See also: hash function.
Oracle Application Server Single Sign-On
OracleAS Single Sign-On consists of program logic that enables you to log in securely to applications such as expense reports, mail, and benefits. These applications take two forms: partner applications and external applications. In both cases, you gain access to several applications by authenticating only once.

Oracle Call Interface (OCI)
An application programming interface (API) that enables you to create applications that use the native procedures or function calls of a third-generation language to access an Oracle Database server and control all phases of SQL statement execution.

Oracle Certificate Authority
Oracle Application Server Certificate Authority is a Certificate Authority (CA) for use within your Oracle Application Server environment. OracleAS Certificate Authority uses Oracle Internet Directory as the storage repository for certificates. OracleAS Certificate Authority integration with OracleAS Single Sign-On and Oracle Internet Directory provides seamless certificate provisioning mechanisms for applications relying on them. A user provisioned in Oracle Internet Directory and authenticated in OracleAS Single Sign-On can choose to request a digital certificate from OracleAS Certificate Authority.

Oracle CMS
Oracle CMS implements the IETF Cryptographic Message Syntax (CMS) protocol. CMS defines data protection schemes that allow for secure message envelopes.

Oracle Containers for J2EE (OC4J)
A lightweight, scalable container for Java 2 Platform, Enterprise Edition (J2EE).

Oracle Context

Oracle Crypto
Oracle Crypto is a pure Java library that provides core cryptography algorithms.

Oracle Database Advanced Replication
A feature in the Oracle Database that enables database tables to be kept synchronized across two Oracle databases.

Oracle Delegated Administration Services
A set of individual, pre-defined services—called Oracle Delegated Administration Services units—for performing directory operations on behalf of a user. Oracle Internet Directory Self-Service Console makes it easier to develop and deploy administration solutions for both Oracle and third-party applications that use Oracle Internet Directory.

Oracle Directory Integration and Provisioning
A collection of interfaces and services for integrating multiple directories by using Oracle Internet Directory and several associated plug-ins and connectors. A feature of Oracle Internet Directory that enables an enterprise to use an external user repository to authenticate to Oracle products.
**Oracle Directory Integration and Provisioning Server**
In an Oracle Directory Integration Platform environment, a daemon process that monitors Oracle Internet Directory for change events and takes action based on the information present in the directory integration profile.

**Oracle Directory Integration Platform**
A component of Oracle Internet Directory. It is a framework developed to integrate applications around a central LDAP directory like Oracle Internet Directory.

**Oracle Directory Manager**
A Java-based tool with a graphical user interface for administering Oracle Internet Directory.

**Oracle Enterprise Manager**
A separate Oracle product that combines a graphical console, agents, common services, and tools to provide an integrated and comprehensive systems management platform for managing Oracle products.

**Oracle HTTP Server**
Software that processes Web transactions that use the Hypertext Transfer Protocol (HTTP). Oracle uses HTTP software developed by the Apache Group.

**Oracle Identity Management**
An infrastructure enabling deployments to manage centrally and securely all enterprise identities and their access to various applications in the enterprise.

**Oracle Internet Directory**
A general purpose directory service that enables retrieval of information about dispersed users and network resources. It combines Lightweight Directory Access Protocol (LDAP) Version 3 with the high performance, scalability, robustness, and availability of the Oracle Database.

**Oracle Liberty SDK**
Oracle Liberty SDK implements the Liberty Alliance Project specifications enabling federated single sign-on between third-party Liberty-compliant applications.

**Oracle Net Services**
The foundation of the Oracle family of networking products, allowing services and their client applications to reside on different computers and communicate. The main function of Oracle Net Services is to establish network sessions and transfer data between a client application and a server. Oracle Net Services is located on each computer in the network. Once a network session is established, Oracle Net Services acts as a data courier for the client and the server.

**Oracle PKI certificate usages**
Defines Oracle application types that a certificate supports.

**Oracle PKI SDK**
Oracle PKI SDK implements the security protocols that are necessary within public key infrastructure (PKI) implementations.
Oracle SAML
Oracle SAML provides a framework for the exchange of security credentials among disparate systems and applications in an XML-based format as outlined in the OASIS specification for the Security Assertions Markup Language (SAML).

Oracle Security Engine

Oracle S/MIME
Oracle S/MIME implements the Secure/Multipurpose Internet Mail Extension (S/MIME) specifications from the Internet Engineering Task Force (IETF) for secure e-mail.

Oracle Wallet Manager
A Java-based application that security administrators use to manage public-key security credentials on clients and servers.

See also: Oracle Advanced Security Administrator’s Guide.

Oracle Web Services Security
Oracle Web Services Security provides a framework for authentication and authorization using existing security technologies as outlined in the OASIS specification for Web Services Security.

Oracle XML Security
Oracle XML Security implements the W3C specifications for XML Encryption and XML Signature.

OracleAS Portal
An OracleAS Single Sign-On partner application that provides a mechanism for integrating files, images, applications, and Web sites. The External Applications portlet provides access to external applications.

other information repository
In an Oracle Directory Integration Platform environment, in which Oracle Internet Directory serves as the central directory, any information repository except Oracle Internet Directory.

OWM
See Oracle Wallet Manager.

partition
A unique, non-overlapping directory naming context that is stored on one directory server.

partner application
An Oracle Application Server application or non-Oracle application that delegates the authentication function to the OracleAS Single Sign-On server. This type of application spares users from reauthenticating by accepting mod_osso headers.

peer-to-peer replication
Also called multimaster replication or n-way replication. A type of replication that enables multiple sites, acting as equals, to manage groups of replicated data. In such a
replication environment, each node is both a supplier and a consumer node, and the entire directory is replicated on each node.

**PKCS#1**
The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS#1 provides recommendations for the implementation of public-key cryptography based on the RSA algorithm, covering the following aspects: cryptographic primitives; encryption schemes; signature schemes; ASN.1 syntax for representing keys and for identifying the schemes.

**PKCS#5**
The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS#5 provides recommendations for the implementation of password-based cryptography.

**PKCS#7**
The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS #7 describes general syntax for data that may have cryptography applied to it, such as digital signatures and digital envelopes.

**PKCS#8**
The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS #8 describes syntax for private key information, including a private key for some public key algorithms and a set of attributes. The standard also describes syntax for encrypted private keys.

**PKCS#10**
The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS #10 describes syntax for a request for certification of a public key, a name, and possibly a set of attributes.

**PKCS#12**
The Public Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories. PKCS #12 describes a transfer syntax for personal identity information, including private keys, certificates, miscellaneous secrets, and extensions. Systems (such as browsers or operating systems) that support this standard allow a user to import, export, and exercise a single set of personal identity information—typically in a format called a wallet.

**PKI**
See public key infrastructure (PKI).

**plaintext**
Plaintext is readable data prior to a transformation to ciphertext using encryption, or readable data that is the result of a transformation from ciphertext using decryption.

**point-to-point replication**
Also called fan-out replication is a type of replication in which a supplier replicates directly to a consumer. That consumer can then replicate to one or more other consumers. The replication can be either full or partial.

**policy precedence**
In Oracle Application Server Certificate Authority (OCA), policies are applied to incoming requests in the order that they are displayed on the main policy page. When
the OCA policy processor module parses policies, those that appear toward the top of the policy list are applied to requests first. Those that appear toward the bottom of the list are applied last and take precedence over the others. Only enabled policies are applied to incoming requests.

**policy.properties**

A multipurpose configuration file for Oracle Application Server Single Sign-On that contains basic parameters required by the single sign-on server. Also used to configure advanced features of OracleAS Single Sign-On, such as multilevel authentication.

**POSIX**

Portable Operating System Interface for UNIX. A set of programming interface standards governing how to write application source code so that the applications are portable between operating systems. A series of standards being developed by the [Internet Engineering Task Force (IETF)](https://www.ietf.org).

**POST Profile**

An authentication method whereby login credentials are submitted within the body of the login form.

**predicates**

In Oracle Application Server Certificate Authority (OCA), a policy predicate is a logical expression that can be applied to a policy to limit how it is applied to incoming certificate requests or revocations. For example, the following predicate expression specifies that the policy in which it appears can have a different effect for requests or revocations from clients with DNs that include "ou=sales,o=acme,c=us":

```
Type=="client" AND DN=="ou=sales,o=acme,c=us"
```

**principal**

One of the three primary roles defined in the identity federation protocols supported by Oracle Identity Federation. The other roles are identity provider and service provider.

A principal is any entity capable of using a service and capable of acquiring a federated identity. Typically, a principal is a person or user, or a system entity whose identity can be authenticated.

**primary node**

In an Oracle Application Server Cold Failover Cluster (Identity Management), the cluster node on which the application runs at any given time.

See also: secondary node.

**private key**

A private key is the secret key in a public/private key pair used in public key cryptography. An entity uses its private key to decrypt data that has been encrypted with its public key. The entity can also use its private key to create digital signatures. The security of data encrypted with the entity’s public key as well as signatures created by the private key depends on the private key remaining secret.

**private key cryptography**

See symmetric cryptography.
profile
See directory integration profile.

Project Liberty
See Liberty Alliance.

provisioned applications
Applications in an environment where user and group information is centralized in Oracle Internet Directory. These applications are typically interested in changes to that information in Oracle Internet Directory.

provisioning
The process of providing users with access to applications and other resources that may be available in an enterprise environment.

provisioning agent
An application or process that translates Oracle-specific provisioning events to external or third-party application-specific events.

provisioning integration profile
A special kind of directory integration profile that describes the nature of provisioning-related notifications that Oracle Directory Integration Platform sends to the directory-enabled applications.

proxy server
A server between a client application, such as a Web browser, and a real server. It intercepts all requests to the real server to see if it can fulfill the requests itself. If not, it forwards the request to the real server. In OracleAS Single Sign-On, proxies are used for load balancing and as an extra layer of security.

See also: load balancer.

proxy user
A kind of user typically employed in an environment with a middle tier such as a firewall. In such an environment, the end user authenticates to the middle tier. The middle tier then logs into the directory on the end user’s behalf. A proxy user has the privilege to switch identities and, once it has logged into the directory, switches to the end user’s identity. It then performs operations on the end user’s behalf, using the authorization appropriate to that particular end user.

public key
A public key is the non-secret key in a public/private key pair used in public key cryptography. A public key allows entities to encrypt data that can only then be decrypted with the public key’s owner using the corresponding private key. A public key can also be used to verify digital signatures created with the corresponding private key.

public key certificate
See certificate.

public key cryptography
Public key cryptography (also known as asymmetric cryptography) uses two keys, one public and the other private. These keys are called a key pair. The private key must be kept secret, while the public key can be transmitted to any party. The private key and
the public key are mathematically related. A message that is signed by a private key can be verified by the corresponding public key. Similarly, a message encrypted by the public key can be decrypted by the private key. This method ensures privacy because only the owner of the private key can decrypt the message.

**public key encryption**

The process in which 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 the recipient’s private key.

**public key infrastructure (PKI)**

A public key infrastructure (PKI) is a system that manages the issuing, distribution, and authentication of public keys and private keys. A PKI typically comprises the following components:

- A **Certificate Authority (CA)** that is responsible for generating, issuing, publishing and revoking digital certificates.
- A **Registration Authority (RA)** that is responsible for verifying the information supplied in requests for certificates made to the CA.
- A directory service where a certificate or certificate revocation list (CRL) gets published by the CA and where they can be retrieved by relying third parties.
- Relying third parties that use the certificates issued by the CA and the public keys contained therein to verify digital signatures and encrypt data.

**public/private key pair**

A mathematically related set of two numbers 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 available only to their owners. Data encrypted with a public key can only be decrypted with its associated private key and vice versa. Data encrypted with a public key cannot be decrypted with the same public key.

**RC2**

Rivest Cipher Two (RC2) is a 64-bit block cipher developed by Ronald Rivest for RSA Security, and was designed as a replacement for Data Encryption Standard (DES).

**RC4**

Rivest Cipher Four (RC4) is a stream cipher developed by Ronald Rivest for RSA Security. RC4 allows variable key lengths up to 1024 bits. RC4 is most commonly used to secure data communications by encrypting traffic between Web sites that use the Secure Sockets Layer (SSL) protocol.

**RDN**

See relative distinguished name (RDN).

**readable data**

Data prior to a transformation to ciphertext via encryption or data that is the result of a transformation from ciphertext via decryption.

**realm**

See identity management realm.
realm search base
An attribute in the root Oracle Context that identifies the entry in the directory information tree (DIT) that contains all identity management realms. This attribute is used when mapping a simple realm name to the corresponding entry in the directory.

referral
Information that a directory server provides to a client and which points to other servers the client must contact to find the information it is requesting.

See also: knowledge reference.

Registration Authority (RA)
The Registration Authority (RA) is responsible for verifying and enrolling users before a certificate is issued by a Certificate Authority (CA). The RA may assign each applicant a relative distinguished value or name for the new certificate applied. The RA does not sign or issue certificates.

registry entry
An entry containing runtime information associated with invocations of Oracle Internet Directory servers, called a directory server instance. Registry entries are stored in the directory itself, and remain there until the corresponding directory server instance stops.

relational database
A structured collection of data that stores data in tables consisting of one or more rows, each containing the same set of columns. Oracle makes it very easy to link the data in multiple tables. This is what makes Oracle a relational database management system, or RDBMS. It stores data in two or more tables and enables you to define relationships between the tables. The link is based on one or more fields common to both tables.

relative distinguished name (RDN)
The local, most granular level entry name. It has no other qualifying entry names that would serve to uniquely address the entry. In the example, cn=Smith, o=acme, c=US, the RDN is cn=Smith.

remote master site (RMS)
In a replicated environment, any site, other than the master definition site (MDS), that participates in Oracle Database Advanced Replication.

replica
Each copy of a naming context that is contained within a single server.

replication agreement
A special directory entry that represents the replication relationship among the directory servers in a directory replication group (DRG).

response time
The time between the submission of a request and the completion of the response.

RFC
The Internet Request For Comments (or RFC) documents are the written definitions of the protocols and policies of the Internet. The Internet Engineering Task Force (IETF) facilitates the discussion, development, and establishment of new standards. A
standard is published using the RFC acronym and a reference number. For example, the official standard for e-mail is RFC 822.

**root CA**
In a hierarchical **public key infrastructure (PKI)**, the root **Certificate Authority (CA)** is the CA whose **public key** serves as the most trusted datum for a security domain.

**root directory specific entry (DSE)**
An entry storing operational information about the directory. The information is stored in a number of attributes.

**root DSE**
See **root directory specific entry (DSE)**.

**root Oracle Context**
In the Oracle Identity Management infrastructure, the root Oracle Context is an entry in Oracle Internet Directory containing a pointer to the default identity management realm in the infrastructure. It also contains information on how to locate an identity management realm given a simple name of the realm.

**RSA**
RSA is a **public key cryptography** algorithm named after its inventors (Rivest, Shamir, and Adelman). The RSA algorithm is the most commonly used encryption and authentication algorithm and is included as part of the Web browsers from Netscape and Microsoft, and many other products.

**RSAES-OAEP**
The **RSA Encryption Scheme - Optimal Asymmetric Encryption Padding** (RSAES-OAEP) is a public key encryption scheme combining the **RSA** algorithm with the OAEP method. Optimal Asymmetric Encryption Padding (OAEP) is a method for encoding messages developed by Mihir Bellare and Phil Rogaway.

**S/MIME**
See **Secure/Multipurpose Internet Mail Extension (S/MIME)**.

**SAML**
See **Security Assertions Markup Language (SAML)**.

**SASL**
See **Simple Authentication and Security Layer (SASL)**.

**scalability**
The ability of a system to provide throughput in proportion to, and limited only by, available hardware resources.

**schema**
The collection of **attributes**, **object classes**, and their corresponding **matching rules**.

**secondary node**
In an Oracle Application Server Cold Failover Cluster (Identity Management), the cluster node to which an application is moved during a failover.

See also: **primary node**.
**secret key**

A secret key is the key used in a symmetric algorithm. Since a secret key is used for both encryption and decryption, it must be shared between parties that are transmitting ciphertext to one another but must be kept secret from all unauthorized entities.

**secret key cryptography**

See symmetric cryptography.

**Secure Hash Algorithm (SHA)**

Secure Hash Algorithm (SHA) is a hash function algorithm that produces a 160-bit message digest based upon the input. The algorithm is used in the Digital Signature Standard (DSS). With the introduction of the Advanced Encryption Standard (AES) which offers three key sizes: 128, 192 and 256 bits, there has been a need for a companion hash algorithm with a similar level of security. The newer SHA-256, SHA-284 and SHA-512 hash algorithms comply with these enhanced requirements.

**Secure Sockets Layer (SSL)**

Secure Sockets Layer (SSL) is a protocol designed by Netscape Communications to enable encrypted, authenticated communications across networks (such as the Internet). SSL uses the public key encryption system from RSA, which also includes the use of a digital certificate. SSL provides three elements of secure communications: confidentiality, authentication, and integrity.

SSL has evolved into Transport Layer Security (TLS). TLS and SSL are not interoperable. However, a message sent with TLS can be handled by a client that handles SSL.

**Secure/Multipurpose Internet Mail Extension (S/MIME)**

Secure/Multipurpose Internet Mail Extension (S/MIME) is an Internet Engineering Task Force (IETF) standard for securing MIME data through the use of digital signatures and encryption.

**Security Assertions Markup Language (SAML)**

An XML-based framework which defines mechanisms for exchanging security information about a subject by making assertions about the subject that are used to make access control decisions. SAML enables the exchange of authentication and authorization information between identity providers and service providers who otherwise may not be able to interoperate.

SAML 2.0 is a major revision of the standard which updates SAML 1.1 and combines input from both Shibboleth and Liberty ID-FF specifications. A key aspect of SAML 2.0 is the ability for two sites to establish and maintain an identifier for a user, with that user’s cooperation. Additional features include privacy mechanisms and support for global logout.

**security token**

In the Liberty protocol, refers to a set of security information that represents and substantiates a claim.

**server certificate**

A certificate that attests to the identity of an organization that uses a secure Web server to serve data. A server certificate must be associated with a public/private key pair issued by a mutually trusted Certificate Authority (CA). Server certificates are required for secure communications between a browser and a Web server.
service provider
One of the three primary roles defined in the identity federation protocols supported by Oracle Identity Federation. The other roles are identity provider and principal.

A service provider, which is the relying party in SAML, provides services or goods to a principal while relying on an identity provider to authenticate the principal’s identity.

service time
The time between the initiation of a request and the completion of the response to the request.

session key
A secret key that is used for the duration of one message or communication session.

SGA
See System Global Area (SGA).

SHA
See Secure Hash Algorithm (SHA).

shared server
A server that is configured to allow many user processes to share very few server processes, so the number of users that can be supported is increased. With shared server configuration, many user processes connect to a dispatcher. The dispatcher directs multiple incoming network session requests to a common queue. An idle shared server process from a shared pool of server processes picks up a request from the queue. This means a small pool of server processes can server a large amount of clients. Contrast with dedicated server.

sibling
An entry that has the same parent as one or more other entries.

Signed Public Key And Challenge (SPKAC)
Signed Public Key And Challenge (SPKAC) is a proprietary protocol used by the Netscape Navigator browser to request certificates.

simple authentication
The process by which the client identifies itself to the server by means of a DN and a password which are not encrypted when sent over the network. In the simple authentication option, the server verifies that the DN and password sent by the client match the DN and password stored in the directory.

Simple Authentication and Security Layer (SASL)
Simple Authentication and Security Layer (SASL) is a method for adding authentication and authorization capabilities to application protocols. SASL provides a security layer between the protocol and the connection, so that users can be authenticated to a server. A security layer can also be negotiated to protect subsequent protocol interactions.

Simple Object Access Protocol (SOAP)
Simple Object Access Protocol (SOAP) is an XML-based protocol that defines a framework for exchanging messages between systems over the Internet. A common protocol for Web Services, SOAP is used with transport protocols such as HTTP and FTP. A SOAP message consists of three parts — an envelope that describes the
message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

**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 sign-off**
The process by which you terminate an OracleAS Single Sign-On session and log out of all active partner applications simultaneously. You can do this by logging out of the application that you are working in.

**single sign-on (SSO)**
In a federated environment, single sign-on enables users to sign on once with a member of a federated group of identity providers and service providers, and later use resources available from members without needing to sign on again.

**single sign-on SDK**
Legacy APIs to enable OracleAS Single Sign-On partner applications for single sign-on. The SDK consists of PL/SQL and Java APIs as well as sample code that demonstrates how these APIs are implemented. This SDK is now deprecated and mod_osso is used instead.

**single sign-on server**
Program logic that enables users to log in securely to single sign-on applications such as expense reports, mail, and benefits.

**SLAPD**
Standalone LDAP daemon. An LDAP directory server service that is responsible for most functions of a directory except replication.

**slave**
See consumer.

**smart knowledge reference**
A knowledge reference that is returned when the knowledge reference entry is in the scope of the search. It points the user to the server that stores the requested information.

**SOAP**

**specific administrative area**
Administrative areas control:
- Subschema administration
- Access control administration
- Collective attribute administration

A specific administrative area controls one of these aspects of administration. A specific administrative area is part of an autonomous administrative area.
SPKAC
See Signed Public Key And Challenge (SPKAC).

sponsor node
In replication, the node that is used to provide initial data to a new node.

SSL
See Secure Sockets Layer (SSL).

SSO
See single sign-on (SSO).

stream cipher
Stream ciphers are a type of symmetric algorithm. A stream cipher encrypts in small units, often a bit or a byte at a time, and implements some form of feedback mechanism so that the key is constantly changing. RC4 is an example of a stream cipher.

See also: block cipher.

subACLSubentry
A specific type of subentry that contains access control list (ACL) information.

subclass
An object class derived from another object class. The object class from which it is derived is called its superclass.

subentry
A type of entry containing information applicable to a group of entries in a subtree. The information can be of these types:
- Access control policy points
- Schema rules
- Collective attributes

Subentries are located immediately below the root of an administrative area.

subordinate CA
In a hierarchical public key infrastructure (PKI), the subordinate Certificate Authority (CA) is a CA whose certificate signature key is certified by another CA, and whose activities are constrained by that other CA.

subordinate reference
A knowledge reference pointing downward in the directory information tree (DIT) to a naming context that starts immediately below an entry

subschema DN
The list of directory information tree (DIT) areas having independent schema definitions.

subSchemaSubentry
A specific type of subentry containing schema information.
subtree
A section of a directory hierarchy, which is also called a directory information tree (DIT). The subtree typically starts at a particular directory node and includes all subdirectories and objects below that node in the directory hierarchy.

subtype
An attribute with one or more options, in contrast to that same attribute without the options. For example, a commonName (cn) attribute with American English as an option is a subtype of the commonName (cn) attribute without that option. Conversely, the commonName (cn) attribute without an option is the supertype of the same attribute with an option.

success URL
When using Oracle Application Server Single Sign-On, the URL to the routine responsible for establishing the session and session cookies for an application.

super user
A special directory administrator who typically has full access to directory information.

superclass
The object class from which another object class is derived. For example, the object class person is the superclass of the object class organizationalPerson. The latter, namely, organizationalPerson, is a subclass of person and inherits the attributes contained in person.

superior reference
A knowledge reference pointing upward to a directory system agent (DSA) that holds a naming context higher in the directory information tree (DIT) than all the naming contexts held by the referencing DSA.

supertype
An attribute without options, in contrast to the same attribute with one or more options. For example, the commonName (cn) attribute without an option is the supertype of the same attribute with an option. Conversely, a commonName (cn) attribute with American English as an option is a subtype of the commonName (cn) attribute without that option.

supplier
In replication, the server that holds the master copy of the naming context. It supplies updates from the master copy to the consumer server.

symmetric algorithm
A symmetric algorithm is a cryptographic algorithm that uses the same key for encryption and decryption. There are essentially two types of symmetric (or secret key) algorithms — stream ciphers and block ciphers.

symmetric cryptography
Symmetric cryptography (or shared secret cryptography) systems use the same key to encipher and decipher data. The problem with symmetric cryptography is ensuring a secure method by which the sender and recipient can agree on the secret key. If a third party were to intercept the secret key in transit, they could then use it to decipher anything it was used to encipher. Symmetric cryptography is usually faster than
asymmetric cryptography, and is often used when large quantities of data need to be exchanged. DES, RC2, and RC4 are examples of symmetric cryptography algorithms.

**symmetric key**
See secret key.

**System Global Area (SGA)**
A group of shared memory structures that contain data and control information for one Oracle database instance. If multiple users are concurrently connected to the same instance, the data in the instance SGA is shared among the users. Consequently, the SGA is sometimes referred to as the "shared global area." The combination of the background processes and memory buffers is called an Oracle instance.

**system operational attribute**
An attribute holding information that pertains to the operation of the directory itself. Some operational information is specified by the directory to control the server, for example, the time stamp for an entry. Other operational information, such as access information, is defined by administrators and is used by the directory program in its processing.

**think time**
The time the user is not engaged in actual use of the processor.

**third-party access management system**
Non-Oracle single sign-on system that can be modified to use OracleAS Single Sign-On to gain access to Oracle Application Server applications.

**throughput**
The number of requests processed by Oracle Internet Directory for each unit of time. This is typically represented as "operations per second."

**Time Stamp Protocol (TSP)**
Time Stamp Protocol (TSP), as specified in RFC 3161, defines the participating entities, the message formats, and the transport protocol involved in time stamping a digital message. In a TSP system, a trusted third-party Time Stamp Authority (TSA) issues time stamps for messages.

**TLS**
See Transport Layer Security (TLS).

**Transport Layer Security (TLS)**
A protocol providing communications privacy over the Internet. The protocol enables client/server applications to communicate in a way that prevents eavesdropping, tampering, or message forgery.

**Triple Data Encryption Standard (3DES)**
Triple Data Encryption Standard (3DES) is based on the Data Encryption Standard (DES) algorithm developed by IBM in 1974, and was adopted as a national standard in 1977. 3DES uses three 64-bit long keys (overall key length is 192 bits, although actual key length is 56 bits). Data is encrypted with the first key, decrypted with the second key, and finally encrypted again with the third key. This makes 3DES three times slower than standard DES but also three times more secure.
trusted certificate
A third party identity that is qualified with a level of trust. The trust is used when an identity is being validated as the entity it claims to be. Typically, trusted certificates come from a **Certificate Authority (CA)** you trust to issue user certificates.

trustpoint
See **trusted certificate**.

TSP
See **Time Stamp Protocol (TSP)**.

Unicode
A type of universal character set, a collection of 64K characters encoded in a 16-bit space. It encodes nearly every character in just about every existing character set standard, covering most written scripts used in the world. It is owned and defined by Unicode Inc. Unicode is canonical encoding which means its value can be passed around in different locales. But it does not guarantee a round-trip conversion between it and every Oracle character set without information loss.

UNIX Crypt
The UNIX encryption algorithm.

URI
Uniform Resource Identifier (URI). A way to identify any point of content on the Web, whether it be a page of text, a video or sound clip, a still or animated image, or a program. The most common form of URI is the Web page address, which is a particular form or subset of URI called a **URL**.

URL
Uniform Resource Locator (URL). The address of a file accessible on the Internet. The file can be a text file, HTML page, image file, a program, or any other file supported by HTTP. The URL contains the name of the protocol required to access the resource, a domain name that identifies a specific computer on the Internet, and a hierarchical description of the file location on the computer.

URLC token
The OracleAS Single Sign-On code that passes authenticated user information to the **partner application**. The partner application uses this information to construct the session cookie.

user name mapping module
A OracleAS Single Sign-On Java module that maps a user **certificate** to the user’s nickname. The nickname is then passed to an authentication module, which uses this nickname to retrieve the user’s certificate from the directory.

user search base
In the Oracle Internet Directory default **directory information tree (DIT)**, the node in the identity management realm under which all the users are placed.

UTC (Coordinated Universal Time)
The standard time common to every place in the world. Formerly and still widely called Greenwich Mean Time (GMT) and also World Time, UTC nominally reflects the
mean solar time along the Earth’s prime meridian. UTC is indicated by a z at the end of the value, for example, 200011281010z.

**UTF-8**
A variable-width 8-bit encoding of Unicode that uses sequences of 1, 2, 3, or 4 bytes for each character. Characters from 0-127 (the 7-bit ASCII characters) are encoded with one byte, characters from 128-2047 require two bytes, characters from 2048-65535 require three bytes, and characters beyond 65535 require four bytes. The Oracle character set name for this is AL32UTF8 (for the Unicode 3.1 standard).

**UTF-16**
16-bit encoding of Unicode. The Latin-1 characters are the first 256 code points in this standard.

**verification**
Verification is the process of ensuring that a given digital signature is valid, given the public key that corresponds to the private key purported to create the signature and the data block to which the signature purportedly applies.

**virtual host**
A single physical Web server machine that is hosting one or more Web sites or domains, or a server that is acting as a proxy to other machines (accepts incoming requests and reroutes them to the appropriate server).

In the case of OracleAS Single Sign-On, virtual hosts are used for load balancing between two or more OracleAS Single Sign-On servers. They also provide an extra layer of security.

**virtual host name**
In an Oracle Application Server Cold Failover Cluster (Identity Management), the host name corresponding to a particular virtual IP address.

**virtual IP address**
In an Oracle Application Server Cold Failover Cluster (Identity Management), each physical node has its own physical IP address and physical host name. To present a single system image to the outside world, the cluster uses a dynamic IP address that can be moved to any physical node in the cluster. This is called the virtual IP address.

**wait time**
The time between the submission of the request and initiation of the response.

**wallet**
An abstraction used to store and manage security credentials for an individual entity. It implements the storage and retrieval of credentials for use with various cryptographic services. A wallet resource locator (WRL) provides all the necessary information to locate the wallet.

**Wallet Manager**
See Oracle Wallet Manager.

**Web service**
A Web service is application or business logic that is accessible using standard Internet protocols, such as HTTP, XML, and SOAP. Web Services combine the best aspects of component-based development and the World Wide Web. Like components, Web
Services represent black-box functionality that can be used and reused without regard to how the service is implemented.

**Web Services Description Language (WSDL)**

Web Services Description Language (WSDL) is the standard format for describing a Web service using XML. A WSDL definition describes how to access a Web service and what operations it will perform.

**WSDL**

See Web Services Description Language (WSDL).

**WS-Federation**

Web Services Federation Language (WS-Federation) is a specification developed by Microsoft, IBM, BEA, VeriSign, and RSA Security. It defines mechanisms to allow federation between entities using different or like mechanisms by allowing and brokering trust of identities, attributes, and authentication between participating Web services.

See also: Liberty Alliance.

**X.500**

X.500 is a standard from the International Telecommunication Union (ITU) that defines how global directories should be structured. X.500 directories are hierarchical with different levels for each category of information, such as country, state, and city.

**X.509**

X.509 is the most widely used standard for defining digital certificates. A standard from the International Telecommunication Union (ITU), for hierarchical directories with authentication services, used in many public key infrastructure (PKI) implementations.

**XKMS**

The XML Key Management Specification (XKMS), developed by the World Wide Web Consortium (W3C), specifies protocols for distributing and registering public keys. XKMS comprises two parts: the XML Key Information Service Specification (X-KISS), which defines a protocol for a Trust service that resolves public key information; and the XML Key Registration Service Specification (X-KRSS), which defines a protocol for a web service that accepts registration of public key information.

**XML**

Extensible Markup Language (XML) is a specification developed by the World Wide Web Consortium (W3C). XML is a pared-down version of Standard Generalized Mark-Up Language (SGML), designed especially for Web documents. XML is a metalanguage (a way to define tag sets) that allows developers to define their own customized markup language for many classes of documents.

**XML canonicalization (C14N)**

This is a process by which two logically equivalent XML documents can be resolved to the same physical representation. This has significance for digital signatures because a signature can only verify against the same physical representation of the data against which it was originally computed. For more information, see the W3C’s XML Canonicalization specification.
Index

A
algorithms, 3-1
  asymmetric, 1-2
  Diffie-Hellman, 2-9
hash, 1-3
key agreement, 2-9
message digest, 2-8
signature, 2-7
symmetric, 1-2

C
certificate authority, 1-4
ciphers, 2-5
  symmetric, 2-5
CMP, 1-5, 7-1
CMS, 1-4
  authenticated data, 5-17
  constructing objects, 5-4
  detached objects, 5-12
  digested data, 5-7
  encrypted data, 5-12
  enveloped data, 5-14
  object types, 5-4
  reading objects, 5-5
  signed data, 5-9
CRMF, 7-1
cryptography, 1-1
  algorithms, 1-2

D
DER, 8-9
digital certificates, 4-1
DTD, 8-1

E
ECC, 1-11
ECDSA, 2-3
Elliptic Curve Cryptography, 1-11
Elliptic Curve Digital Signature Algorithm, 2-3
Enhanced Security Services, 6-14

F
federation, 1-9
FIM, 11-1

H
HMAC, 2-9, 3-2, 5-18

I
identity provider, 1-8, 1-10

J
Java API Reference
  Oracle CMS, 5-25
  Oracle Crypto, 2-11
  Oracle Liberty SDK 1.1, 11-9
  Oracle Liberty SDK 1.2, 11-19
  Oracle PKI SDK CMP, 7-3
  Oracle PKI SDK LDAP, 7-9
  Oracle PKI SDK OCSP, 7-5
  Oracle PKI SDK TSP, 7-7
  Oracle SAML, 9-6
  Oracle SAML 2.0, 9-10
  Oracle Security Engine, 4-5
  Oracle S/MIME, 6-14
  Oracle Web Services Security, 10-6
  Oracle XKMS, 12-7
JCE, 3-1

K
  key agreement, 2-9, 5-17
  key encryption, 5-17
  key pairs
  generating, 2-3
  key transport, 5-17

L
LDAP, 1-5, 7-7
Liberty Alliance, 11-1
Liberty protocol
  authentication context, 11-9
  authorization request, 11-3
authorization response, 11-4
base message class, 11-9
federation termination notification, 11-4
logout request, 11-5
logout response, 11-6
register name ID request, 11-6
register name ID response, 11-7

M
MAC, 1-1, 1-13, 2-3, 2-9, 5-18
Message Authentication Code, 1-1, 2-3, 5-18
message digests, 2-8

O
OCSP, 1-5, 7-3
Oracle CMS, 1-12, 5-1
to developing applications with, 5-3
technology setup, 5-2
features and benefits, 5-1
system requirements, 5-2
Oracle Crypto, 1-11
core classes, 2-2
technology setup, 2-2
eample programs, 2-11
features and benefits, 2-1
supported algorithms, 2-1
Oracle JCE Provider, 1-13, 3-1
technology setup, 3-3
features and benefits, 3-1
supported algorithms, 3-1
Oracle Liberty SDK, 1-14, 11-1
core classes, 11-3
features and benefits, 11-1
initialization, 11-8
supporting classes, 11-8
Oracle Liberty SDK 1.1
technology setup, 11-2
Oracle PKI SDK, 1-12, 7-1
Oracle PKI SDK CMP, 1-13, 7-1
technology setup, 7-2
features and benefits, 7-3
Oracle PKI SDK LDAP, 1-12, 7-7
technology setup, 7-8
features and benefits, 7-7
Oracle PKI SDK OCSP, 1-12, 7-3
technology setup, 7-4
features and benefits, 7-3
Oracle PKI SDK TSP, 1-12, 7-5
technology setup, 7-6
features and benefits, 7-5
Oracle SAML, 1-14, 9-1
core classes, 9-3
technology setup, 9-2
features and benefits, 9-1
SAML 2.0 core classes, 9-8
SAML 2.0 technology setup, 9-7

P
password based encryption, 2-6
PBE objects
generating, 2-7
PEM, 8-9
PKCS 10 certificate requests, 4-4
PKCS#12, 8-9
PKCS#7, 8-9
PKI, 7-1
to and CMP, 1-5
to and CMS, 1-4
to and LDAP, 1-5
to and OCSP, 1-5
to and S/MIME, 1-4
to and TSP, 1-5
benefits, 1-5
digital certificates, 1-4
key pairs, 1-3
principal, 1-9
PRNG, 2-10
seeding, 2-10
pseudo-random numbers, 2-10
public key infrastructure, 1-3

R
RSA ciphers, 2-6
generating, 2-6
SAML, 1-6, 9-1
  2.0, 1-7
  and XML security, 1-9
  assertion element, 9-4
  Oracle SAML 1.0/1.1 packages, 9-2
  Oracle SAML 2.0 packages, 9-6
  profiles, 1-9
  request and response cycle, 1-8
  request element, 9-4
  response element, 9-5
  service provider, 1-8
  signatures, 2-7
  single sign-on, 1-10
  S/MIME, 1-4
    new message, 6-5
  SOAP, 1-6
  SSO, 11-1
  symmetric ciphers
    generating, 2-5
  symmetric key pairs
    generating, 2-4

T
  TSP, 1-5, 7-5

W
  WSS, 1-6, 10-1
    security elements, 10-4
    SOAP message, 10-4

X
  X.500, 4-3
  X.500 names, 4-3
  X509, 1-4, 3-3
  X.509 certificates, 4-5
  XKMS, 8-1, 12-1
  XML, 8-1
    cipher data, 8-8
    encryption, 8-6
    key encryption, 8-7
    security requirements, 8-1
    signature creation, 8-4, 8-5
    signature verification, 8-5
  XML security
    common questions, 8-9