

# OraRSA

## Table of contents

---

Introduction .....	3
Install .....	4
Setup .....	4
Load the JAR files .....	4
Register ORA_RSA package .....	4
Permissions .....	4
Upgrade .....	4
Uninstall .....	5
Switch from Trial to Production version .....	5
RSA (X.509) Keys .....	6
Constants .....	7
Functions .....	8
ENCRYPT .....	8
ENCRYPT_WITH_MODULUS .....	8
DECRYPT .....	8
DECRYPT_WITH_MODULUS .....	9
SIGN .....	9
VERIFY .....	9
GETVERSION .....	9
IS_TRIAL_VERSION .....	10
GET_RSA_ERROR .....	10
Exception Handling .....	11
Support and Contact .....	12

## Introduction

---

OraRSA (ORA\_RSA) is a PL/SQL package for the Oracle(c) Database product family, version 11 and above, that offers RSA cryptography procedures.

Internally the package is implemented as a set of Java Stored Procedures.

In order to perform asymmetric RSA cryptography operations the package utilizes existing public and private [RSA \(X.509\) keys](#).

# Install

---

## Setup

### Step 0) Extracting the package

**Extract** the distribution ZIP archive to a folder on your development disk drive (referred below as "**extraction folder**")

### Load the JAR files

#### Step 1) Load the JAR files

Load the JAR files using the *loadjava.sh/.bat* utility located in the Oracle Database instance /BIN/ folder. (For example on Windows *C:\app\<User>\product\<oracle version>\db\_1\BIN\loadjava.bat*)

**Note:** Replace below *user/pass* with a database schema/user name and password applicable for your Oracle Database.

*[Windows environment]*

```
loadjava.bat -r -v -u user/pass [extraction folder]\SetupFiles\jce-jdk13-152.jar  
loadjava.bat -r -v -u user/pass [extraction folder]\SetupFiles\ora-rsa-1.1.0.jar
```

*[Unix environment]*

```
loadjava.sh -r -v -u user/pass [extraction folder]/SetupFiles/jce-jdk13-152.jar  
loadjava.sh -r -v -u user/pass [extraction folder]/SetupFiles/ora-rsa-1.1.0.jar
```

### Register ORA\_RSA package

#### Step 2) Registering the ORA\_RSA PL/SQL package

Using your favorite PL/SQL execution environment (SQL\*Plus, Oracle(c) SQLDeveloper(c), etc.) execute the scripts:

```
[extraction folder]/SetupFiles/Ora_RSA_Package.sql  
[extraction folder]/SetupFiles/Ora_RSA_Package_Body.sql
```

### Permissions

#### Step 3) Permissions

Using your favorite PL/SQL execution environment (SQL\*Plus, Oracle(c) SQLDeveloper(c), etc.) execute:

```
CALL dbms_java.grant_permission( 'user',  
'SYS:java.security.SecurityPermission', 'putProviderProperty.BC', ' ' );  
CALL dbms_java.grant_permission( 'user',  
'SYS:java.security.SecurityPermission', 'insertProvider.BC', ' ' );  
commit;
```

Note: Replace above 'user' with the database user/schema name that will execute the code.

### Upgrade

Upgrading to a new version of ORA\_RSA requires **first to unload the old version JAR files** from the Oracle© database:

```
dropjava.sh/.bat -r -v -u user/pass [extraction folder]\SetupFiles\jce-jdk13-152.jar
```

```
dropjava.sh/.bat -r -v -u user/pass [extraction folder]\SetupFiles\ora-rsa-1.0.0.jar
```

Then continue to [Setup](#)

Note: Replace above user/password with the database user/schema name and password used in the initial setup.

## Uninstall

The uninstall process is similar to the [Upgrade](#) and requires **first to unload the old version JAR files** from the Oracle© database:

```
dropjava.sh/.bat -r -v -u user/pass [extraction folder]\SetupFiles\jce-jdk13-152.jar  
dropjava.sh/.bat -r -v -u user/pass [extraction folder]\SetupFiles\ora-rsa-1.1.0.jar
```

and afterwards to drop the ORA\_RSA package using your favorite PL/SQL execution environment (SQL\*Plus, Oracle(c) SQLDeveloper(c), etc).

```
SQL> drop package ORA_RSA;  
SQL> drop package body ORA_RSA;
```

Note: Replace above user/pass with the database user/schema name and password that were used in step [Setup](#)

## Switch from Trial to Production version

To switch from an evaluation (trial) version of the software to a licensed production version, please execute the steps in the [Upgrade process](#).

## RSA (X.509) Keys

---

In order to perform RSA asymmetric cryptography, you will need RSA keys (also known as X.509 keys or certificates). The package assumes that you already have such keys generated by other software tools (like OpenSSL e.g.)

The PL/SQL functions provided by the package accept the keys transparently either as [local file system path locations](#) or serialized in text format (PEM format) or binary format (DER format). RSA keys can be met in a wide variety of file formats and the package internally tries to recognize the format of the provided key/certificate and utilize it.

## Constants

---

The ORA\_RSA package defines the following constants (used for [exception handling](#)):

```
-- wrong password specified for a private RSA key
RSA_WRONG_PASSWORD_ERR CONSTANT INTEGER := 871;
-- error loading RSA key
RSA_KEY_ERR CONSTANT INTEGER := 872;
-- RSA operation error
RSA_ENCRYPTION_ERR CONSTANT INTEGER := 873;
-- I/O error
RSA_GENERAL_IO_ERR CONSTANT INTEGER := 770;
```

## Functions

---

The API list is available online at <http://www.didisoft.com/ora-rsa/tutorial/functions/>

### ENCRYPT

**function ENCRYPT(message RAW, public\_key RAW) return RAW**

**Description:**

RSA encrypts data with an asymmetric public key.

**Parameters:**

*message* - message to be encrypted

*public\_key* - the bytes of the RSA (X.509) public key

**Result**

the encrypted field

**Online example:**

<http://www.didisoft.com/ora-rsa/tutorial/encrypting/>

### ENCRYPT\_WITH\_MODULUS

**function ENCRYPT\_WITH\_MODULUS(message RAW, public\_key\_modulus VARCHAR2, public\_key\_exponent VARCHAR2) return RAW**

**Description:**

RSA encrypts data with an asymmetric public key supplied as RSA algorithm parameters

**Parameters:**

*message* - message to be encrypted

*public\_key\_modulus* - the modulus of the RSA public key as hexadecimal string

*public\_key\_exponent* - the exponent of the RSA public key as hexadecimal string

**Result**

the encrypted field

**Online example:**

<http://www.didisoft.com/ora-rsa/tutorial/encrypting/>

### DECRYPT

**function DECRYPT(data RAW, private\_key RAW) return RAW**

**function DECRYPT(data RAW, private\_key RAW, key\_password varchar2) return RAW**

Decrypts an RSA encrypted field with a specified RSA private key.

**Parameters:**

*message* - [encrypted message](#) to be decrypted

*private\_key* - the RSA private key bytes PEM encoded or DER encoded, or in .pfx/.p12 format

*key\_password* - password that unlocks the private key (optional)

**Result**

the decrypted data



**Online example:**

<http://www.didisoft.com/ora-rsa/tutorial/decrypting/>

## DECRYPT\_WITH\_MODULUS

**function DECRYPT\_WITH\_MODULUS(data RAW, private\_key\_modulus VARCHAR2,  
private\_key\_exponent VARCHAR2) return RAW**

Decrypts an RSA encrypted field with a specified RSA private key supplied as raw RSA algorithm parameters

**Parameters:**

*message* - [encrypted message](#) to be decrypted

*private\_key\_modulus* - the modulus of the RSA private key as hexadecimal string

*private\_key\_exponent* - the exponent of the RSA private key as hexadecimal string

**Result**

the decrypted data

**Online example:**

<http://www.didisoft.com/ora-rsa/tutorial/decrypting/>

## SIGN

**function SIGN(message RAW, private\_key RAW) return BLOB**

**function SIGN(message RAW, private\_key RAW, private\_key\_password varchar2) return BLOB**

**Description:**

Produces an RSA digital signature for using a specified RSA private key.

**Parameters:**

*message* - data for which a digital signature will be generated

*private\_key* - the RSA private key bytes PEM encoded or DER encoded, or in .pfx/.p12 format

*key\_password* - password that unlocks the private key (optional)

**Result**

RSA digital signature over SHA-1 hash of the input data

## VERIFY

**function VERIFY(message RAW, signature RAW, public\_key RAW) return pls\_integer**

verifies an RSA digital signature [computed over data](#) field with a specified public key.

**Parameters:**

*message* - signed message for which the signature was computed

*signature* - digital signature of the message

*public\_key* - absolute file path on the server to the public key or the public key PEM or DER encoded

**Result**

1 - if the signature was validated with the provided public key

0 - if the signature cannot be validated with the provided public key

## GETVERSION

**function GETVERSION return VARCHAR2**

**Description:**

Returns the build version of the package

**Result**

the build version number of the package

**Example**

```
select ORA_RSA.GETVERSION from dual
```

## IS\_TRIAL\_VERSION

**function IS\_TRIAL\_VERSION return pls\_integer**

**Description:**

Returns is the installed package a trial (evaluation) or production version

**Result**

1 - if the package is a 30 day trial (evaluation) version  
0 - if the package is a production version

**Example**

```
select ORA_RSA.IS_TRIAL_VERSION from dual
```

## GET\_RSA\_ERROR

**function GET\_RSA\_ERROR return pls\_integer**

**Description:**

Returns a value used for error identification

**Result**

Value	Meaning
ORA_RSA.RSA_WRONG_PASSWORD_ERR	The provided password for an RSA key is wrong
ORA_RSA.RSA_KEY_ERR	Broken or illegal RSA key provided
ORA_RSA.RSA_ENCRYPTION_ERR	RSA operation error
ORA_RSA.RSA_GENERAL_IO_ERR	Data I/O error

*Note: The values for the constants above can be seen in section [Constants](#).*

**Example**

Check section [exception handling](#)

## Exception Handling

---

Upon error the ORA\_RSA package functions throw exception of type **RSA\_EXCEPTION**.

In order to identify the exact cause of the error the helper function **GET\_RSA\_ERROR** can be used as shown below:

```
BEGIN
    ORA_RSA.<method>( ...
EXCEPTION
    WHEN ORA_RSA.RSA_EXCEPTION THEN
        BEGIN
            IF ORA_RSA.GET_RSA_ERROR() = ORA_RSA.RSA_WRONG_PASSWORD_ERR THEN
                DBMS_OUTPUT.PUT_LINE('The password for the private key is not
matching: ' || SQLERRM);
            ELSIF ORA_RSA.GET_RSA_ERROR() = ORA_RSA.RSA_KEY_ERR THEN
                DBMS_OUTPUT.PUT_LINE('The provided key is not a valid RSA key. ');
            ELSIF ORA_RSA.GET_RSA_ERROR() = ORA_RSA.RSA_ENCRYPTION_ERR THEN
                DBMS_OUTPUT.PUT_LINE('Error when performing RSA operation: ' ||
SQLERRM);
            ELSIF ORA_RSA.GET_RSA_ERROR() = ORA_RSA.RSA_GENERAL_IO_ERR THEN
                DBMS_OUTPUT.PUT_LINE('I/O error: ' || SQLERRM);
            END IF;
        END;

    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('General error : ' || SQLERRM );
END;
/
```

## Support and Contact

---

### Technical support

To receive general information or **technical support**, please contact us at [support@didisoft.com](mailto:support@didisoft.com).

### Sales

For questions related to sales, volume licensing, or OEM licensing, please contact us at [sales@didisoft.com](mailto:sales@didisoft.com).

### Product Updates

If you have purchased the library you can access our [Customers' Portal](#) where you can download new versions.

### Newsletter

To receive product update news, you can subscribe to our [Newsletter](#)

For further information, visit us at [www.didisoft.com](http://www.didisoft.com)

If you have any ideas, wishes, questions or criticism, don't hesitate to contact us. We will be glad to hear from you.