57
DBMS_CRYPTO

`DBMS_CRYPTO` provides an interface to encrypt and decrypt stored data, and can be used in conjunction with PL/SQL programs running network communications. It provides support for several industry-standard encryption and hashing algorithms, including the Advanced Encryption Standard (AES) encryption algorithm. AES has been approved by the National Institute of Standards and Technology (NIST) to replace the Data Encryption Standard (DES).

> **Note:**
>
> - Starting from Oracle Database 23ai Release, the use of the `MD4` algorithm is desupported.
> - Starting from Oracle Database 21c Release, the use of the `MD5` and `MD4` algorithms are deprecated.
> - Starting from Oracle Database 21c Release, the use of the `SHA-1` algorithm for `SQLNET` and `DBMS_CRYPTO` is deprecated.
>
> Oracle recommends that you do not use deprecated values in new applications. Support for deprecated features is for backward compatibility only.

This chapter contains the following topics:

> **See Also:**
>
> - *Oracle Database Security Guide* for more information about using this package and about encrypting data in general.

DBMS_CRYPTO Overview

DBMS_CRYPTO contains basic cryptographic functions and procedures. To use this package correctly and securely, a general level of security expertise is assumed.

The DBMS_CRYPTO package enables encryption and decryption for common Oracle datatypes, including RAW and large objects (LOBs), such as images and sound. Specifically, it supports BLOBs and CLOBs. In addition, it provides Globalization Support for encrypting data across different database character sets. It also supports asymmetric key functions.

The following cryptographic algorithms are supported:

- Data Encryption Standard (DES), Triple DES (3DES, 2-key and 3-key) (deprecated)
- Advanced Encryption Standard (AES)
- MD5 (deprecated), SHA-1 (deprecated), and SHA-2 and SHA-3 cryptographic hashes
- MD5, SHA-1, and SHA-2 and SHA-3 Message Authentication Code (MAC), SHA-3 KMACXOF
- SM2, SM3, SM4 (SM stands for ShangMi)

Block cipher modifiers are also provided with DBMS_CRYPTO. You can choose from several padding options, including PKCS (Public Key Cryptographic Standard) #5, and from six block cipher chaining modes, including Cipher Block Chaining (CBC).

Table 57-1 summarizes the DBMS_CRYPTO package features.

Table 57-1 DBMS_CRYPTO Features

Package Feature	DBMS_CRYPTO
Cryptographic algorithms	DES, 3DES, AES, 3DES_2KEY, SM4
PKENCRYPT and PKDECRYPT algorithm	PKENCRYPT_RSA_PKCS1_OAEP_SHA2, SM4
Padding forms	PAD_PKCS5 (PKCS5 padding), PAD_ZERO (zeroes padding) PAD_NONE (no padding), PAD_ORCL (Oracle padding)
Block cipher chaining modes	CBC, CFB, ECB, OFB, GCM, CCM, XTS
Cryptographic hash algorithms	MD5, SHA-1, SHA-2 (SHA-256, SHA-384, SHA-512), SM3, (MD5 and SHA-1 are deprecated; MD4 is desupported) HASH_SHAKE128, HASH_SHAKE256
Keyed hash (MAC) algorithms	HMAC_MD5, HMAC_SH1, HMAC_SH256, HMAC_SH384, HMAC_SH512, HMAC_SHA3_224, HMAC_SHA3_256, HMAC_SHA3_384, HMAC_SHA3_512
Keccak MAC algorithms	KMACXOF_128, KMACXOF_256
Cryptographic pseudo-random number generator	RAW, NUMBER, BINARY_INTEGER
Database types	RAW, CLOB, BLOB

The DBMS_CRYPTO package replaces DBMS_OBFUSCATION_TOOLKIT, providing greater ease of use and support for a range of algorithms to accommodate new and existing systems. Specifically, 3DES_2KEY are provided for backward compatibility. It is not recommended that you use these algorithms because they do not provide the same level of security as provided by 3DES, AES, MD5, SHA-1, or SHA-2.

DBMS_CRYPTO Security Model

Oracle Database installs this package in the SYS schema. You can then grant package access to existing users and roles as needed.

DBMS_CRYPTO Constants

The DBMS_CRYPTO package uses the constants listed and described in this topic.

Table 57-2 DBMS_CRYPTO Constants

Name	Type	Value	Description

AES_CCM_NONE	PLS_INTEGER	ENCRYPT_AES + CHAIN_CCM + PAD_NONE	If you are using DBMS_CRYPTO to encrypt a plaintext with AES algorithm in CCM mode, then set the typ argument of DBMS_CRYPTO.ENCRYPT to the value AES_CCM_NONE to ensure that padding is disabled, because CCM mode cannot be used together with padding. Provide one extra input AAD (Additional Authenticated Data, optional) and one extra output TAG (non-optional). For example:
			
			ciphertext := dbms_crypto.encrypt
			(src => plaintext,
			typ => DBMS_CRYPTO.AES_CCM_NONE ,
			key => key_information,
			iv => iv_information,
			aad => aad_information, tag);
			plaintext := dbms_crypto.decrypt
			(src => plaintext,
			typ => DBMS_CRYPTO.AES_CCM_NONE ,
			key => key_information,
			iv => iv_information,
			aad => aad_information,
			tag=> tag_information);

Table 57-2 (Cont.) DBMS_CRYPTO Constants

Name	Type	Value	Description
AES_GCM_NONE	PLS_INTEGER	ENCRYPT_AES + CHAIN_GCM + PAD_NONE	If you are using DBMS_CRYPTO to encrypt a plaintext with AES algorithm in GCM mode, then set the typ argument of DBMS_CRYPTO.ENCRYPT to the value AES_GCM_NONE to ensure that padding is disabled, because GCM mode cannot be used together with padding. Provide one extra input AAD (Additional Authenticated Data, optional) and one extra output TAG (non-optional). For example: `ciphertext := dbms_crypto.encrypt (src => plaintext, typ => DBMS_CRYPTO.AES_GCM_NONE , key => key_information, iv => iv_information, aad => aad_information, tag); plaintext := dbms_crypto.decrypt (src => plaintext, typ => DBMS_CRYPTO.AES_GCM_NONE , key => key_information, iv => iv_information, aad => aad_information, tag=> tag_information);`
AES_XTS_NONE	PLS_INTEGER	ENCRYPT_AES + CHAIN_XTS + PAD_NONE	Constant for AES encryption algorithm in XTS mode. You can use this constant with DBMS_CRYPTO.ENCRYPT. To use AES with XTS mode, the input must be at least one full block (16 bytes).
DES3_CBC_NONE	PLS_INTEGER	ENCRYPT_3DES + CHAIN_CBC + PAD_NONE	If you are using DBMS_CRYPTO to decipher a triple-DES ciphertext that you created in the past using the desupported DBMS_OBFUSCATION_TOOLKIT, then set the typ argument of DBMS_CRYPTO.decrypt to the value DBMS_CRYPTO.DES3_CBC_NONE to ensure that the PKCS#5 padding is disabled. For example: `plaintext := DBMS_CRYPTO.decrypt (src => ciphertext_from_legacy_DES3Encrypt ,typ => DBMS_CRYPTO.DES3_CBC_NONE ,key => key_information ,iv => hextoraw(DBMS_CRYPTO.LEGACY_DEFAULT_IV) );`
AES_CBC_PKCS5	PLS_INTEGER	ENCRYPT_AES + CHAIN_CBC + PAD_PKCS5	Constant for AES encryption algorithm in CBC mode and PKCS5 padding. You can use this constant with DBMS_CRYPTO.ENCRYPT.

Table 57-2 (Cont.) DBMS_CRYPTO Constants

Name	Type	Value	Description
LEGACY_DEFAULT_IV	VARCHAR2(16)	0123456789ABCDEF	If you did not provide the IV argument when creating a triple-DES ciphertext using the desupported DBMS_OBFUSCATION_TOOLKIT, then provide IV as hextoraw(DBMS_CRYPTO.LEGACY_DEFAULT_IV) when invoking DBMS_CRYPTO to decrypt the triple-DES ciphertext. For example: `plaintext := DBMS_CRYPTO.decrypt (src => ciphertext_from_legacy_DES3Encrypt ,typ => DBMS_CRYPTO.DES3_CBC_NONE ,key => key_information ,iv => hextoraw(dbms_crypto.LEGACY_DEFAULT_IV) );`
SM4_CFB_NONE	PLS_INTEGER	ENCRYPT_SM4 + CHAIN_CFB + PAD_NONE	Cipher suite for the SM4 encryption algorithm in Ciphertext Feedback (CFB) mode. You can only use CFB mode with no padding mode PAD_NONE.
SM4_OFB_NONE	PLS_INTEGER	ENCRYPT_SM4 + CHAIN_OFB + PAD_NONE	Cipher suite for the SM4 encryption algorithm in Output Feedback (OFB) mode. You can only use OFB mode with no padding mode PAD_NONE.

DBMS_CRYPTO Datatypes

Parameters for the DBMS_CRYPTO subprograms use these datatypes.

Table 57-3 DBMS_CRYPTO Datatypes

Type	Description
BLOB	A source or destination binary LOB
CLOB	A source or destination character LOB (excluding NCLOB)
PLS_INTEGER	Specifies a cryptographic algorithm type (used with BLOB, CLOB, and RAW datatypes)
RAW	A source or destination RAW buffer

DBMS_CRYPTO Algorithms

The DBMS_CRYPTO package contains predefined cryptographic algorithms, modifiers, and cipher suites.

These are shown in the following tables.

Table 57-4 DBMS_CRYPTO Cryptographic Hash Functions

Name	Description
HASH_MD5	Produces a 128-bit hash, but is more complex than MD4 (which is desupported). Note that MD5 is deprecated.
HASH_SH1	Secure Hash Algorithm (SHA-1) (deprecated). Produces a 160-bit hash.
HASH_SH256	SHA-2, produces a 256-bit hash.
HASH_SH384	SHA-2, produces a 384-bit hash.
HASH_SH512	SHA-2, produces a 512-bit hash.
HASH_SHA3_224	SHA-3, produces a 224-bit hash.
HASH_SHA3_256	SHA-3, produces a 256-bit hash.
HASH_SHA3_384	SHA-3, produces a 384-bit hash.
HASH_SHA3_512	SHA-3, produces a 512-bit hash.
HASH_SHAKE128	Produces variable length hash with 128-bit security level; used for the DBMS_CRYPTO.HASH_LEN only.
HASH_SHAKE256	Produces variable length hash with 256-bit security level; used for the DBMS_CRYPTO.HASH_LEN only.
HASH_SM3	Produces a 256-bit hash

Table 57-5 DBMS_CRYPTO MAC (Message Authentication Code) Functions

Name	Description
HMAC_MD5 (deprecated)	Same as MD5 (deprecated) hash function, except it requires a secret key to verify the hash value.
HMAC_SH1 (deprecated)	Same as SHA hash function, except it requires a secret key to verify the hash value. Complies with IETF RFC 2104 standard.
HMAC_SH256	Same as SHA-2 256-bit hash function, except it requires a secret key to verify the hash value.
HMAC_SH384	Same as SHA-2 384-bit hash function, except it requires a secret key to verify the hash value.
HMAC_SH512	Same as SHA-2 512-bit hash function, except it requires a secret key to verify the hash value.
HMAC_SHA3_224	Same as SHA-3 224-bit hash function, except it requires a secret key to verify the hash value.
HMAC_SHA3_256	Same as SHA-3 256-bit hash function, except it requires a secret key to verify the hash value.
HMAC_SHA3_384	Same as SHA-3 384-bit hash function, except it requires a secret key to verify the hash value.
HMAC_SHA3_512	Same as SHA-3 512-bit hash function, except it requires a secret key to verify the hash value.

Table 57-6 DBMS_CRYPTO KMACXOF Functions

Name	Description
KMACXOF_128	Variable length KMAC with 128-bit security level. KMAC stands for KECCAK Message Authentication Code.

Table 57-6 (Cont.) DBMS_CRYPTO KMACXOF Functions

Name	Description
KMACXOF_256	Variable length KMAC with 256-bit security level.

Table 57-7 DBMS_CRYPTO Encryption Algorithms

Name	Description
ENCRYPT_DES (deprecated)	Data Encryption Standard. Block cipher. Uses key length of 56 bits.
ENCRYPT_3DES_2KEY (deprecated)	Data Encryption Standard. Block cipher. Operates on a block 3 times with 2 keys. Effective key length of 112 bits.
ENCRYPT_3DES (deprecated)	Data Encryption Standard. Block cipher. Operates on a block 3 times.
ENCRYPT_AES128	Advanced Encryption Standard. Block cipher. Uses 128-bit key size.
ENCRYPT_AES192	Advanced Encryption Standard. Block cipher. Uses 192-bit key size.
ENCRYPT_AES256	Advanced Encryption Standard. Block cipher. Uses 256-bit key size.
PKENCRYPT_RSA_PKCS1_OAEP (deprecated)	Public key encryption algorithm; only allowed for decryption.
PKENCRYPT_RSA_PKCS1_OAEP_SHA2	Public key encryption algorithm.
PKENCRYPT_SM2	Provides Chinese SM2 signature and encryption algorithm support
ENCRYPT_SM4	Block cipher used in the Chinese National Standard for Wireless LAN WAPI (WLAN Authentication and Privacy Infrastructure) and also used with Transport Layer Security

Table 57-8 DBMS_CRYPTO Block Cipher Suites

Name	Description
AES_CBC_PKCS5	ENCRYPT_AES + CHAIN_CBC + PAD_PKCS5
AES_CCM_NONE	ENCRYPT_AES + CHAIN_GCM + PAD_NONE
AES_GCM_NONE	ENCRYPT_AES + CHAIN_CCM + PAD_NONE
AES_XTS_NONE	ENCRYPT_AES + CHAIN_XTS + PAD_NONE
DES_CBC_PKCS5	ENCRYPT_DES[1] + CHAIN_CBC[2]+ PAD_PKCS5[3]
DES3_CBC_PKCS5	ENCRYPT_3DES[1] + CHAIN_CBC[2] + PAD_PKCS5[3]

1 See Table 57-7
2 See Table 57-9
3 See Table 57-10

Table 57-9 DBMS_CRYPTO Block Cipher Chaining Modifiers

Name	Description
CHAIN_CBC	Cipher Block Chaining. Plaintext is XORed with the previous ciphertext block before it is encrypted.

Table 57-9 (Cont.) DBMS_CRYPTO Block Cipher Chaining Modifiers

Name	Description
CHAIN_CCM	The Counter with CBC-MAC (CCM) mode. It is a generic authenticated encryption block cipher mode. It can be used with AES (AES_CCM_NONE). The encryption of AES_CCM_NONE takes one extra input AAD (Additional Authenticated Data, optional) and one extra output TAG (non-optional). The decryption of AES_CCM_NONE takes two extra inputs AAD and TAG.
CHAIN_CFB	Cipher-Feedback. Enables encrypting units of data smaller than the block size.
CHAIN_ECB	Electronic Codebook. Encrypts each plaintext block independently.
CHAIN_GCM	The Galois/Counter Mode. It is a generic authenticated encryption block cipher mode. It can be used with AES (AES_GCM_NONE). The encryption of AES_GCM_NONE takes one extra input AAD (Additional Authenticated Data, optional) and one extra output TAG (non-optional). The decryption of AES_GCM_NONE takes two extra inputs AAD and TAG.
CHAIN_OFB	Output-Feedback. Enables running a block cipher as a synchronous stream cipher. Similar to CFB, except that *n* bits of the previous output block are moved into the right-most positions of the data queue waiting to be encrypted.
CHAIN_XTS	XEX-based modified-codebook mode with ciphertext stealing (XTS), a chain mode for AES. It can be used with the AES algorithm in DBMS_CRYPTO and PAD_NONE

Table 57-10 DBMS_CRYPTO Block Cipher Padding Modifiers

Name	Description
PAD_NONE	Provides option to specify no padding. Caller must ensure that blocksize is correct, else the package returns an error.
PAD_ORCL	Provides padding that adds a single byte containing the pad length to the end of the crypto vector.
PAD_ZERO	Provides padding consisting of zeroes

DBMS_CRYPTO Restrictions

The VARCHAR2 datatype is not directly supported by DBMS_CRYPTO. Before you can perform cryptographic operations on data of the type VARCHAR2, you must convert it to the uniform database character set AL32UTF8, and then convert it to the RAW datatype. After performing these conversions, you can then encrypt it with the DBMS_CRYPTO package.

> **See Also:**
>
> DBMS_CRYPTO Operational Notes for information about the conversion rules for converting datatypes.
>
> DBMS_CRYPTO Examples for examples of using the call to UTL_I18N.STRING_TO_RAW to allow VARCHAR2 to be encrypted.

DBMS_CRYPTO Exceptions

The following table lists exceptions that have been defined for DBMS_CRYPTO.

Table 57-11 DBMS_CRYPTO Exceptions

Exception	Code	Description
CipherSuiteInvalid	28827	The specified cipher suite is not defined.
CipherSuiteNull	28829	No value has been specified for the cipher suite to be used.
KeyNull	28239	The encryption key has not been specified or contains a NULL value.
KeyBadSize	28234	DES keys: Specified key size is too short. DES keys must be at least 8 bytes (64 bits). AES keys: Specified key size is not supported. AES keys must be 128, 192, or 256 bits in length.
DoubleEncryption	28233	Source data was previously encrypted.

DBMS_CRYPTO Operational Notes

This section describes several DBMS_CRYPTO operational notes.

- When to Use Encrypt and Decrypt Procedures or Functions
- When to Use Hash or Message Authentication Code (MAC) Functions
- About Generating and Storing Encryption Keys
- Conversion Rules

When to Use Encrypt and Decrypt Procedures or Functions

This package includes both ENCRYPT and DECRYPT procedures and functions. The procedures are used to encrypt or decrypt LOB datatypes (overloaded for CLOB and BLOB datatypes). In contrast, the ENCRYPT and DECRYPT functions are used to encrypt and decrypt RAW datatypes. Data of type VARCHAR2 must be converted to RAW before you can use DBMS_CRYPTO functions to encrypt it.

When to Use Hash or Message Authentication Code (MAC) Functions

This package includes two different types of one-way hash functions: the HASH function and the MAC function. Hash functions operate on an arbitrary-length input message, and return a fixed-length hash value. One-way hash functions work in one direction only. It is easy to compute a hash value from an input message, but it is extremely difficult to generate an input message that hashes to a particular value. Note that hash values should be at least 256 bits in length to be considered secure.

You can use hash values to verify whether data has been altered. For example, before storing data, the user runs DBMS_CRYPTO.HASH against the stored data to create a hash value. On returning the stored data, the user can again run the hash function against it, using the same algorithm. If the second hash value is identical to the first one, then the data has not been altered. Hash values are similar to "file fingerprints" and are used to ensure data integrity.

The HASH function included with DBMS_CRYPTO, is a one-way hash function that you can use to generate a hash value from either RAW or LOB data. The MAC function is also a one-way hash function, but with the addition of a secret key. It works the same way as the DBMS_CRYPTO.HASH function, except only someone with the key can verify the hash value.

MACs can be used to authenticate files between users. They can also be used by a single user to determine if her files have been altered, perhaps by a virus. A user could compute the MAC of his files and store that value in a table. If the user did not use a MAC function, then the virus could compute the new hash value after infection and replace the table entry. A virus cannot do that with a MAC because the virus does not know the key.

About Generating and Storing Encryption Keys

The DBMS_CRYPTO package can generate random material for encryption keys, but it does not provide a mechanism for maintaining them. Application developers must take care to ensure that the encryption keys used with this package are securely generated and stored. Also note that the encryption and decryption operations performed by DBMS_CRYPTO occur on the server, not on the client. Consequently, if the key is sent over the connection between the client and the server, the connection must be protected by using network encryption. Otherwise, the key is vulnerable to capture over the wire.

Although DBMS_CRYPTO cannot generate keys on its own, it does provide tools you can use to aid in key generation. For example, you can use the RANDOMBYTES function to generate random material for keys.

When generating encryption keys for DES, it is important to remember that some numbers are considered weak and semiweak keys. Keys are considered weak or semiweak when the pattern of the algorithm combines with the pattern of the initial key value to produce ciphertext that is more susceptible to cryptanalysis. To avoid this, filter out the known weak DES keys. Lists of the known weak and semiweak DES keys are available on several public Internet sites.

See Also:

- *Oracle Database Security Guide* for information about configuring network encryption and SSL
- RANDOMBYTES Function

Conversion Rules

- To convert VARCHAR2 to RAW, use the UTL_I18N.STRING_TO_RAW function to perform the following steps:
  1. Convert VARCHAR2 in the current database character set to VARCHAR2 in the AL32UTF8 database character.
  2. Convert VARCHAR2 in the AL32UTF8 database character set to RAW.

  Syntax example:

  ```
  UTL_I18N.STRING_TO_RAW (string, 'AL32UTF8');
  ```
- To convert RAW to VARCHAR2, use the UTL_I18N.RAW_TO_CHAR function to perform the following steps:
  1. Convert RAW to VARCHAR2 in the AL32UTF8 database character set.

2. Convert VARCHAR2 in the AL32UTF8 database character set to VARCHAR2 in the database character set you wish to use.

   Syntax example:

   ```
   UTL_I18N.RAW_TO_CHAR (data, 'AL32UTF8');
   ```

> **See Also:**
>
> UTL_I18N for information about using the UTL_I18N PL/SQL package.

- If you want to store encrypted data of the RAW datatype in a VARCHAR2 database column, then use RAWTOHEX or UTL_ENCODE.BASE64_ENCODE to make it suitable for VARCHAR2 storage. These functions expand data size by 2 and 4/3, respectively.

DBMS_CRYPTO Examples

The examples in this section demonstrate different types of PL/SQL coding using DBMS_CRYPTO functions.

Example 1: AES Encryption with Cipher Block Chaining and PKCS#5 Compliant Padding

This example shows PL/SQL block encrypting and decrypting pre-defined input_string using 256-bit AES algorithm with Cipher Block Chaining and PKCS#5 compliant padding.

```
DECLARE
   input_string       VARCHAR2 (200) :=  'Secret Message';
   output_string      VARCHAR2 (200);
   encrypted_raw      RAW (2000);             -- stores encrypted binary text
   decrypted_raw      RAW (2000);             -- stores decrypted binary text
   num_key_bytes      NUMBER := 256/8;        -- key length 256 bits (32 bytes)
   key_bytes_raw      RAW (32);               -- stores 256-bit encryption key
   encryption_type    PLS_INTEGER :=          -- total encryption type

                         DBMS_CRYPTO.ENCRYPT_AES256
                       + DBMS_CRYPTO.CHAIN_CBC
                       + DBMS_CRYPTO.PAD_PKCS5;

   iv_raw             RAW (16);

BEGIN
   DBMS_OUTPUT.PUT_LINE ( 'Original string: ' || input_string);
   key_bytes_raw := DBMS_CRYPTO.RANDOMBYTES (num_key_bytes);
   iv_raw        := DBMS_CRYPTO.RANDOMBYTES (16);
   encrypted_raw := DBMS_CRYPTO.ENCRYPT
      (
         src => UTL_I18N.STRING_TO_RAW (input_string,  'AL32UTF8'),
         typ => encryption_type,
         key => key_bytes_raw,
         iv  => iv_raw
      );

    -- The encrypted value "encrypted_raw" can be used here

    decrypted_raw := DBMS_CRYPTO.DECRYPT
      (
         src => encrypted_raw,
         typ => encryption_type,
```

```
        key => key_bytes_raw,
        iv  => iv_raw
      );

   output_string := UTL_I18N.RAW_TO_CHAR (decrypted_raw, 'AL32UTF8');

   DBMS_OUTPUT.PUT_LINE ('Decrypted string: ' || output_string);
END;
```

Example 2: PKENCRYPTION and PKDECRYPTION Functions

```
DECLARE
  ip_str     VARCHAR (200) := 'Secret Message';
  op_str     VARCHAR (200);
  -- Use OpenSSL to generate the private and public key (2048 bit RSA key)
  -- openssl genrsa -out private.pem 2048
  -- openssl rsa -in private.pem -outform PEM -pubout -out public.pem
  pubkey    VARCHAR (2000) := 'public_key';
  prvkey    VARCHAR (2000) := 'private_key';
  enc_raw    RAW (2000);
  dec_raw    RAW (2000);
  eType      PLS_INTEGER := DBMS_CRYPTO.PKENCRYPT_RSA_PKCS1_OAEP_SHA2;
  kType      PLS_INTEGER := DBMS_CRYPTO.KEY_TYPE_RSA;

BEGIN

  DBMS_OUTPUT.PUT_LINE('-------------------------------------------------');
  DBMS_OUTPUT.PUT_LINE('Original String := ' || ip_str);
  DBMS_OUTPUT.PUT_LINE('-------------------------------------------------');
  enc_raw:= DBMS_CRYPTO.PKENCRYPT
  (
    src        => UTL_I18N.STRING_TO_RAW(ip_str,'AL32UTF8'),
    pub_key    => UTL_I18N.STRING_TO_RAW( pubkey, 'AL32UTF8'),
    pubkey_alg => kType,
    enc_alg    => eType
  );

  dec_raw := DBMS_CRYPTO.PKDECRYPT
  (
    src        => enc_raw,
    prv_key    => UTL_I18N.STRING_TO_RAW( prvkey, 'AL32UTF8'),
    pubkey_alg => kType,
    enc_alg    => eType
  );
  op_str := UTL_I18N.RAW_TO_CHAR(dec_raw,'AL32UTF8');
  dbms_output.put_line('-------------------------------------------------');
  dbms_output.put_line('Decrypted String := ' || op_str);
  dbms_output.put_line('-------------------------------------------------');
end;
/
```

Example 3: SIGN and VERIFY Functions

```
DECLARE
  ip_str     VARCHAR2 (200) := 'Secret Message';
  -- Use OpenSSL to generate the private and public key (2048 bit RSA key)
  -- openssl genrsa -out private.pem 2048
  -- openssl rsa -in private.pem -outform PEM -pubout -out public.pem
  pubkey    VARCHAR (2000) := 'public_key';
  prvkey    VARCHAR (2000) := 'private_key';
  sign_raw   RAW (2000);
  returnval  BOOLEAN := false;
```

```
  sType      PLS_INTEGER := DBMS_CRYPTO.SIGN_SHA224_RSA;
  kType      PLS_INTEGER := DBMS_CRYPTO.KEY_TYPE_RSA;
BEGIN
  sign_raw := DBMS_CRYPTO.SIGN
  (
   src        => UTL_I18N.STRING_TO_RAW(ip_str,'AL32UTF8'),
   prv_key    => UTL_I18N.STRING_TO_RAW( prvkey, 'AL32UTF8'),
   pubkey_alg => kType,
   sign_alg   => sType
  );
  returnval := DBMS_CRYPTO.VERIFY
  (
   src        => UTL_I18N.STRING_TO_RAW( ip_str,'AL32UTF8'),
   sign       => sign_raw,
   pub_key    => UTL_I18N.STRING_TO_RAW( pubkey, 'AL32UTF8'),
   pubkey_alg => kType,
   sign_alg   => sType
  );
  DBMS_OUTPUT.PUT_LINE('-------------------------------------------------');
  IF returnval THEN
    DBMS_OUTPUT.PUT_LINE('True');
  ELSE
    DBMS_OUTPUT.PUT_LINE('False');
  END IF;
  DBMS_OUTPUT.PUT_LINE('-------------------------------------------------');
END;
/
```

Example 4: HASH_SHA3_256 and HASH_SHAKE256 Hash Functions

```
DECLARE
  ip_str VARCHAR2 (200) := 'Secret Message';
  output_string VARCHAR2 (2000);
  source_text_raw RAW (2000);
  hash_raw RAW (2000);
BEGIN
  source_text_raw := UTL_I18N.STRING_TO_RAW(ip_str,'AL32UTF8');
  hash_raw := dbms_crypto.hash(source_text_raw, dbms_crypto.hash_sha3_256);

  output_string := RAWTOHEX(hash_raw);

  DBMS_OUTPUT.PUT_LINE ('Hash string: ' || output_string);
  DBMS_OUTPUT.PUT_LINE (utl_raw.length(hash_raw);
END;
/


HASH_SHAKE256:

DECLARE
  ip_str VARCHAR2 (200) := 'Secret Message';
  output_string VARCHAR2 (2000);
  source_text_raw RAW (2000);
  hash_raw RAW (2000);
  len PLS_INTEGER := 30;
BEGIN
  source_text_raw := UTL_I18N.STRING_TO_RAW(ip_str,'AL32UTF8');
  hash_raw := dbms_crypto.hash_len(source_text_raw, dbms_crypto.hash_shake256, len);

  output_string := RAWTOHEX(hash_raw);

  DBMS_OUTPUT.PUT_LINE ('Hash string: ' || output_string);
```

```
END;
/
```

Example 5: HMAC_SHA3_256 and SIGN_SHA3_256_RSA Functions

```
DECLARE
  ip_str VARCHAR2 (200) := 'Secret Message';
  output_string VARCHAR2 (2000);
  source_text_raw RAW (2000);
  hmac_raw RAW (2000);
  testkey RAW (2000);
BEGIN
  source_text_raw := UTL_I18N.STRING_TO_RAW(ip_str,'AL32UTF8');
  -- Use your test key
  testkey := HEXTORAW('');
  hmac_raw := dbms_crypto.mac(source_text_raw, dbms_crypto.HMAC_SHA3_256, testkey);

  output_string := RAWTOHEX(hmac_raw);

  DBMS_OUTPUT.PUT_LINE ('Hmac string: ' || output_string);
END;
/

SIGN_SHA3_256_RSA:

DECLARE
  ip_str VARCHAR2 (200) := 'Secret Message';
  -- Use OpenSSL to generate the private and public key (2048 bit RSA key)
  -- openssl genrsa -out private.pem 2048
  -- openssl rsa -in private.pem -outform PEM -pubout -out public.pem
  pubkey VARCHAR (2000) := 'public_key';
  prvkey VARCHAR (2000) := 'private_key';
  sign_raw RAW (2000);
  returnval BOOLEAN := false;
  sType PLS_INTEGER := DBMS_CRYPTO.SIGN_SHA3_256_RSA;
  kType PLS_INTEGER := DBMS_CRYPTO.KEY_TYPE_RSA;
BEGIN
  sign_raw := DBMS_CRYPTO.SIGN
  (
   src => UTL_I18N.STRING_TO_RAW(ip_str,'AL32UTF8'),
   prv_key => UTL_I18N.STRING_TO_RAW( prvkey, 'AL32UTF8'),
   pubkey_alg => kType,
   sign_alg => sType
  );
  returnval := DBMS_CRYPTO.VERIFY
  (
   src => UTL_I18N.STRING_TO_RAW( ip_str,'AL32UTF8'),
   sign => sign_raw,
   pub_key => UTL_I18N.STRING_TO_RAW( pubkey, 'AL32UTF8'),
   pubkey_alg => kType,
   sign_alg => sType
  );
  DBMS_OUTPUT.PUT_LINE('-------------------------------------------------');
  IF returnval THEN
    DBMS_OUTPUT.PUT_LINE('True');
  ELSE
    DBMS_OUTPUT.PUT_LINE('False');
  END IF;
  DBMS_OUTPUT.PUT_LINE('-------------------------------------------------');
END;
/
```

Summary of DBMS_CRYPTO Subprograms

This table lists the `DBMS_CRYPTO` subprograms in alphabetical order and briefly describes them.

Table 57-12 ***DBMS_CRYPTO Package Subprograms***

Subprogram	Description
DECRYPT Function	Decrypts `RAW` data using a stream or block cipher with a user supplied key and optional IV (initialization vector)
DECRYPT Procedures	Decrypts `LOB` data using a stream or block cipher with a user supplied key and optional IV
ECDHDERIVE_SHAREDSECRET Function	Derives shared secret using private key of local application and public key from the remote application
ECDH_GENKEYPAIR Function	Generates an EC public/private key pair
ENCRYPT Function	Encrypts `RAW` data using a stream or block cipher with a user supplied key and optional IV
ENCRYPT Procedures	Encrypts `LOB` data using a stream or block cipher with a user supplied key and optional IV
HASH Function	Applies one of the supported cryptographic hash algorithms (MD5, SHA-1, or SHA-2) to data.
	Note: Starting from Oracle Database 20c Release, the use of MD5 and SHA-1 are deprecated. MD4 is desupported. Oracle recommends that you do not use deprecated values in new applications. Support for deprecated features is for backward compatibility only.
HASH_LEN Function	Similar to the `HASH` function, except that it includes an extra input length that uses the `PLS_INTEGER` type
MAC Function	Applies Message Authentication Code algorithms (MD5, SHA-1, or SHA-2) to data to provide keyed message protection.
	Note: Starting from Oracle Database 20c Release, the use of MD5 and SHA-1 are deprecated. Oracle recommends that you do not use deprecated values in new applications. Support for deprecated features is for backward compatibility only.
KMACXOF Function	Similar to the `MAC` function, except that is includes the `length` and `custStr` fields.

Table 57-12 (Cont.) *DBMS_CRYPTO Package Subprograms*

Subprogram	Description
PKDECRYPT Function	Decrypts RAW data using a private key assisted with key algorithm and encryption algorithm and returns decrypted data.
PKENCRYPT Function	Encrypts RAW data using a public key assisted with key algorithm and encryption algorithm and returns encrypted data
RANDOMBYTES Function	Returns a RAW value containing a cryptographically secure pseudo-random sequence of bytes, and can be used to generate random material for encryption keys
RANDOMINTEGER Function	Returns a random BINARY_INTEGER
RANDOMNUMBER Function	Returns a random 128-bit integer of the NUMBER datatype
SIGN Function	Signs RAW data using a private key assisted with key algorithm and sign algorithm, and returns a signature
VERIFY Function	Verifies RAW data using the signature, public key assisted with key algorithm, and sign algorithm. It returns TRUE if the signature was verified

DECRYPT Functions

These functions decrypt RAW data using a stream or block cipher with a user-supplied key and optional IV (initialization vector).

Syntax

```
DBMS_CRYPTO.DECRYPT(
   src IN RAW,
   typ IN PLS_INTEGER,
   key IN RAW,
   iv  IN RAW DEFAULT NULL)
 RETURN RAW;

DBMS_CRYPTO.DECRYPT (
   src IN RAW,
   typ IN PLS_INTEGER,
   key IN RAW,
   iv IN RAW DEFAULT NULL,
   aad IN RAW DEFAULT NULL,
   tag IN RAW)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(decrypt,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 57-13 DECRYPT Function Parameters

Parameter Name	Description
src	RAW data to be decrypted.

Table 57-13 (Cont.) DECRYPT Function Parameters

Parameter Name	Description
typ	Stream or block cipher type and modifiers to be used.
key	Key to be used for decryption.
iv	Optional initialization vector for block ciphers. Default is NULL.
aad	Additional authenticated data, which is any string that you pass to an Oracle Cloud key management service as part of the request.
tag	Authentication tag that is used for the authentication check.

Usage Notes

- To retrieve original plaintext data, DECRYPT must be called with the same cipher, modifiers, key, and IV that was used to encrypt the data originally.

> **See Also:**
>
> "Usage Notes" for the ENCRYPT function for additional information about the ciphers and modifiers available with this package.

- If VARCHAR2 data is converted to RAW before encryption, then it must be converted back to the appropriate database character set by using the UTL_I18N package.

> **See Also:**
>
> DBMS_CRYPTO Operational Notes for a discussion of the VARCHAR2 to RAW conversion rules

DECRYPT Procedures

These procedures decrypt LOB data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

Syntax

```
DBMS_CRYPTO.DECRYPT(
   dst IN OUT NOCOPY BLOB,
   src IN            BLOB,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW          DEFAULT NULL);

DBMS_CRYPT.DECRYPT(
   dst IN OUT NOCOPY CLOB         CHARACTER SET ANY_CS,
   src IN            BLOB,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW          DEFAULT NULL);

DBMS_CRYPTO.DECRYPT (
```

```
   dst IN OUT NOCOPY BLOB,
   src IN            BLOB,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW DEFAULT NULL,
   aad IN            RAW DEFAULT NULL,
   tag IN            RAW);

DBMS_CRYPTO.DECRYPT (
  dst IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  src IN            BLOB,
  typ IN            PLS_INTEGER,
  key IN            RAW,
  iv  IN            RAW DEFAULT NULL,
  aad IN            RAW DEFAULT NULL,
  tag IN            RAW);
```

Pragmas

```
pragma restrict_references(decrypt,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 57-14 DECRYPT Procedure Parameters

Parameter Name	Description
dst	LOB locator of output data. The value in the output LOB <dst> will be overwritten.
src	LOB locator of input data.
typ	Stream or block cipher type and modifiers to be used.
key	Key to be used for decryption.
iv	Optional initialization vector for block ciphers. Default is all zeroes.
aad	Additional authenticated data, which is any string that you pass to an Oracle Cloud key management service as part of the request.
tag	Authentication tag that is used for the authentication check.

ECDHDERIVE_SHAREDSECRET Function

This function derives shared secret using private key of local application and public key from the remote application.

Syntax

```
DBMS_CRYPTO.
DBMS_CRYPTO.ECDHDERIVE_SHAREDSECRET (
   curveid       IN  BINARY_INTEGER,
   pubkey        IN  RAW,
   privkey       IN  RAW,
   sharedsecret  OUT RAW);
```

Parameters

Table 57-15 HASH Function Parameters

Parameter Name	Description
curveid	Constants to denote the curve name that the algorithm uses. Example: SECP_256_R1
pubkey	Public key from the other side
privkey	Private key
sharedsecret	Shared secret generated from private key and public key from the other side

Usage Note

The supported curve id SECP_256_R1 denotes NIST Recommended Curve secp256r1.

ECDH_GENKEYPAIR Function

This function generates an EC public/private key pair.

Syntax

```
DBMS_CRYPTO.
DBMS_CRYPTO.ECDH_GENKEYPAIR (
   curveid       IN  BINARY_INTEGER,
   pubkey        OUT  RAW,
   privkey       OUT  RAW);
```

Parameters

Table 57-16 HASH Function Parameters

Parameter Name	Description
curveid	Constants to denote the curve name that the algorithm uses. Example: SECP_256_R1
pubkey	Public key
privkey	Private key

Usage Note

The supported curve id SECP_256_R1 denotes NIST Recommended Curve secp256r1.

ENCRYPT Functions

These functions encrypt RAW data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

Syntax

```
DBMS_CRYPTO.ENCRYPT(
   src IN RAW,
   typ IN PLS_INTEGER,
   key IN RAW,
```

```
   iv  IN RAW DEFAULT NULL)
 RETURN RAW;

DBMS_CRYPTO.ENCRYPT (
   src IN RAW,
   typ IN PLS_INTEGER,
   key IN RAW,
   iv  IN RAW DEFAULT NULL,
   aad IN RAW DEFAULT NULL,
   tag OUT RAW)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(encrypt,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 57-17 ENCRYPT Function Parameters

Parameter Name	Description
src	RAW data to be encrypted.
typ	Stream or block cipher type and modifiers to be used.
key	Encryption key to be used for encrypting data.
iv	Optional initialization vector for block ciphers. Default is NULL.
aad	Additional authenticated data, which is any string that you pass to an Oracle Cloud key management service as part of the request.
tag	Authentication tag that is used for the authentication check.

Usage Notes

- Block ciphers may be modified with chaining and padding type modifiers. The chaining and padding type modifiers are added to the block cipher to produce a cipher suite. Cipher Block Chaining (CBC) is the most commonly used chaining type, and PKCS #5 is the recommended padding type. See Table 57-9 and Table 57-10 for block cipher chaining and padding modifier constants that have been defined for this package.
- To improve readability, you can define your own package-level constants to represent the cipher suites you use for encryption and decryption. For example, the following example defines a cipher suite that uses DES, cipher block chaining mode, and no padding:

  ```
  DES_CBC_NONE CONSTANT PLS_INTEGER := DBMS_CRYPTO.ENCRYPT_DES
                                     + DBMS_CRYPTO.CHAIN_CBC
                                     + DBMS_CRYPTO.PAD_NONE;
  ```

  See Table 57-8 for the block cipher suites already defined as constants for this package.
- To encrypt VARCHAR2 data, it should first be converted to the AL32UTF8 character set.

> **See Also:**
>
> The discussion of conversion rules under DBMS_CRYPTO Operational Notes

ENCRYPT Procedures

These procedures encrypt LOB data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

Syntax

```
DBMS_CRYPTO.ENCRYPT(
   dst IN OUT NOCOPY BLOB,
   src IN            BLOB,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW          DEFAULT NULL);

DBMS_CRYPTO.ENCRYPT(
   dst IN OUT NOCOPY BLOB,
   src IN            CLOB         CHARACTER SET ANY_CS,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW          DEFAULT NULL);

DBMS_CRYPTO.ENCRYPT (
   dst IN OUT NOCOPY BLOB,
   src IN            BLOB,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW DEFAULT NULL,
   aad IN            RAW DEFAULT NULL,
   tag OUT           RAW);

DBMS_CRYPTO.ENCRYPT(
   dst IN OUT        NOCOPY BLOB,
   src IN            CLOB CHARACTER SET ANY_CS,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW DEFAULT NULL,
   aad IN            RAW DEFAULT NULL,
   tag OUT           RAW);
```

Pragmas

```
pragma restrict_references(encrypt,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 57-18 ENCRYPT Procedure Parameters

Parameter Name	Description
dst	LOB locator of output data. The value in the output LOB <dst> will be overwritten.
src	LOB locator of input data.
typ	Stream or block cipher type and modifiers to be used.
key	Encryption key to be used for encrypting data.
iv	Optional initialization vector for block ciphers. Default is NULL.

Table 57-18 (Cont.) ENCRYPT Procedure Parameters

Parameter Name	Description
aad	Additional authenticated data, which is any string that you pass to an Oracle Cloud key management service as part of the request.
tag	Authentication tag that is used for the authentication check.

Usage Notes

- ENCRYPT can use the ENCRYPT_SM4 constant for SM4 encryption algorithm.
- See DBMS_DEBUG Operational Notes for more information about the conversion rules for the ENCRYPT procedure.

HASH Function

A one-way hash function takes a variable-length input string, the data, and converts it to a fixed-length (generally smaller) output string called a *hash value*. The hash value serves as a unique identifier (like a fingerprint) of the input data. You can use the hash value to verify whether data has been changed or not.

Note that a one-way hash function is a hash function that works in one direction. It is easy to compute a hash value from the input data, but it is hard to generate data that hashes to a particular value. Consequently, one-way hash functions work well to ensure data integrity. Refer to “When to Use Hash or Message Authentication Code (MAC) Functions” in DBMS_CRYPTO Operational Notes for more information about using one-way hash functions.

This function applies to data one of the supported cryptographic hash algorithms listed in Table 57-4.

Syntax

```
DBMS_CRYPTO.HASH (
   src IN RAW,
   typ IN PLS_INTEGER)
 RETURN RAW;

DBMS_CRYPTO.HASH (
   src IN BLOB,
   typ IN PLS_INTEGER)
 RETURN RAW;

DBMS_CRYPTO.HASH (
   src IN CLOB CHARACTER SET ANY_CS,
   typ IN PLS_INTEGER)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(hash,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 57-19 HASH Function Parameters

Parameter Name	Description
src	The source data to be hashed.
typ	The hash algorithm to be used.

Usage Notes

- Oracle recommends that you use SHA-2 (SHA-256, SHA-384, SHA-512). SHA-1 (HASH_SH1) is deprecated.
- The HASH function can use the HASH_SM3 constant for the SM3 hash algorithm

HASH_LEN Function

HASH_LEN is an extension of the HASH function that can generate variable length hash output.

HASH_LEN includes an extra input length in PLS_INTEGER type, which is hash length. HASH_LEN only supports two types of hash: SHAKE128 and SHAKE256.

Syntax

```
DBMS_CRYPTO.HASH_LEN (
   src IN RAW,
   typ IN PLS_INTEGER,
   length IN PLS_INTEGER)
 RETURN RAW DETERMINISTIC;

DBMS_CRYPTO.HASH_LEN (
   src IN BLOB,
   typ IN PLS_INTEGER,
   length IN PLS_INTEGER)
 RETURN RAW DETERMINISTIC;

DBMS_CRYPTO.HASH_LEN (
   src IN CLOB CHARACTER SET ANY_CS,
   typ IN PLS_INTEGER,
   length IN PLS_INTEGER)
 RETURN RAW DETERMINISTIC;
```

Parameters

Table 57-20 HASH Function Parameters

Parameter Name	Description
src	The source data to be hashed.
typ	The hash algorithm to be used.
len	The variable length for SHAKE128 and SHAKE256 hash algorithms.

Usage Notes

`HASH_LEN` can only use hash types `HASH_SHAKE128` and `HASH_SHAKE256`. Other hash types are invalid for this function.

MAC Function

This function applies Message Authentication Code (MAC) algorithms to data to provide keyed message protection.

A MAC is a key-dependent one-way hash function. MACs have the same properties as the one-way hash function described in HASH Function, but they also include a key. Only someone with the identical key can verify the hash. Also refer to “When to Use Hash or Message Authentication Code (MAC) Functions” in DBMS_CRYPTO Operational Notesfor more information about using MACs.

See Table 57-5 for a list of MAC algorithms that have been defined for this package.

Syntax

```
DBMS_CRYPTO.MAC (
   src IN RAW,
   typ IN PLS_INTEGER,
   key IN RAW)
 RETURN RAW;

DBMS_CRYPTO.MAC (
   src IN BLOB,
   typ IN PLS_INTEGER
   key IN RAW)
 RETURN RAW;

DBMS_CRYPTO.MAC (
   src IN CLOB CHARACTER SET ANY_CS,
   typ IN PLS_INTEGER
   key IN RAW)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(mac,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 57-21 MAC Function Parameters

Parameter Name	Description
`src`	Source data to which MAC algorithms are to be applied.
`typ`	MAC algorithm to be used.
`key`	Key to be used for MAC algorithm.

KMACXOF Function

KMAC stands for KECCAK Message Authentication Code. This function is similar to the `MAC` function except that is includes the `length` and `custStr` fields.

Syntax

```
DBMS_CRYPTO.KMACXOF(
   src IN RAW,
   typ IN PLS_INTEGER,
   key IN RAW,
   length IN PLS_INTEGER,
   custStr IN RAW)
 RETURN RAW;

DBMS_CRYPTO.KMACXOF (
   src IN BLOB,
   typ IN PLS_INTEGER,
   key IN RAW,
   length IN PLS_INTEGER,
   custStr IN RAW)
 RETURN RAW;

DBMS_CRYPTO.KMACXOF (
   src IN CLOB CHARACTER SET ANY_CS,
   typ IN PLS_INTEGER,
   key IN RAW,
   length IN PLS_INTEGER,
   custStr IN RAW)
 RETURN RAW;
```

Parameters

Table 57-22 KMACXOF Function Parameters

Parameter Name	Description
`src`	Source data to which KMACXOF algorithms are to be applied.
`typ`	KMACXOF algorithm to be used.
`key`	Key to be used for KMACXOF algorithm.
`length`	Length of KMACXOF output in bytes.
`custStr`	Custom string for KMACXOF.

PKDECRYPT Function

This function decrypts `RAW` data using a private key assisted with key algorithm and encryption algorithm and returns decrypted data.

Syntax

```
DBMS_CRYPTO.PKDECRYPT(
   src IN RAW,
   prv_key IN RAW,
   pubkey_alg IN BINARY_INTEGER,
   enc_alg  IN BINARY_INTEGER)
 RETURN RAW;
```

Parameters

Table 57-23 PKDECRYPT Function Parameters

Parameter Name	Description
src	RAW data to be decrypted.
prv_key	Private key.
pubkey_alg	Specify the KEY_TYPE_RSA RSA key type.
enc_alg	Specify the algorithm PKENCRYPT_RSA_PKCS1_OAEP, for RSA Public Key Cryptosystem with PKCS1 and OAEP padding.

PKENCRYPT Function

This function encrypts RAW data using a public key assisted with key algorithm and encryption algorithm and returns encrypted data.

Syntax

```
DBMS_CRYPTO.PKENCRYPT(
   src IN RAW,
   pub_key IN RAW,
   pubkey_alg IN BINARY_INTEGER,
   enc_alg  IN BINARY_INTEGER)
 RETURN RAW;
```

Parameters

Table 57-24 PKENCRYPT Function Parameters

Parameter Name	Description
src	RAW data to be encrypted.
pub_key	Public key.
pubkey_alg	Specify the KEY_TYPE_RSA RSA key type.
enc_alg	Specify the algorithm PKENCRYPT_RSA_PKCS1_OAEP, for RSA Public Key Cryptosystem with PKCS1 and OAEP padding.

Usage Notes

You can use the PKENCRYPT_SM2 constant for the SM2 public key encryption algorithm and the KEY_TYPE_SM2 constant for SM2 key type.

RANDOMBYTES Function

This function returns a RAW value containing a cryptographically secure pseudo-random sequence of bytes, which can be used to generate random material for encryption keys.

Syntax

```
DBMS_CRYPTO.RANDOMBYTES (
   number_bytes IN POSITIVE)
 RETURN RAW;
```

Pragmas

```
pragma restrict_references(randombytes,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 57-25 RANDOMBYTES Function Parameter

Parameter Name	Description
number_bytes	The number of pseudo-random bytes to be generated.

Usage Note

The number_bytes value should not exceed the maximum length of a RAW variable.

RANDOMINTEGER Function

This function returns an integer in the complete range available for the Oracle BINARY_INTEGER datatype.

Syntax

```
DBMS_CRYPTO.RANDOMINTEGER
 RETURN BINARY_INTEGER;
```

Pragmas

```
pragma restrict_references(randominteger,WNDS,RNDS,WNPS,RNPS);
```

RANDOMNUMBER Function

This function returns an integer in the Oracle NUMBER datatype in the range of [0..2**128-1].

Syntax

```
DBMS_CRYPTO.RANDOMNUMBER
 RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(randomnumber,WNDS,RNDS,WNPS,RNPS);
```

SIGN Function

This function signs RAW data using a private key assisted with key algorithm and sign algorithm, and returns a signature.

Syntax

```
DBMS_CRYPTO.SIGN(
   src IN RAW,
   prv_key IN RAW,
   pubkey_alg IN BINARY_INTEGER,
   sign_alg IN BINARY_INTEGER)
 RETURN RAW;
```

Parameters

Table 57-26 SIGN Function Parameters

Parameter Name	Description
src	RAW data to be signed.
prv_key	Private key.
pubkey_alg	Specify the KEY_TYPE_RSA RSA key type for RSA algorithms and KEY_TYPE_ECDSA ECDSA key type for ECDSA algorithms.
sign_alg	Specify one of the algorithms that are listed in the Usage Notes.

Usage Notes

Table 57-27 Signature Type Algorithms

Hash Algorithm	Description
SIGN_SHA1_RSA	SHA1 hash function with RSA
SIGN_SHA1_RSA_X931	SHA1 hash function with RSA and X931 padding
SIGN_SHA224_ECDSA	SHA 224-bit hash function with ECDSA
SIGN_SHA224_RSA	SHA 224-bit hash function with RSA
SIGN_SHA256_ECDSA	SHA 256-bit hash function with ECDSA
SIGN_SHA256_RSA	SHA 256-bit hash function with RSA
SIGN_SHA256_RSA_X931	SHA 256-bit hash function with RSA and X931 padding
SIGN_SHA3_224_ECDSA	SHA-3 224-bit hash function with ECDSA
SIGN_SHA3_224_RSA	SHA-3 234-bit hash function with RSA
SIGN_SHA3_256_ECDSA	SHA-3 256-bit hash function with ECDSA
SIGN_SHA3_256_RSA	SHA-3 256-bit hash function with RSA
SIGN_SHA3_384_ECDSA	SHA-3 384-bit hash function with ECDSA
SIGN_SHA3_384_RSA	SHA-3 384-bit hash function with RSA
SIGN_SHA3_512_ECDSA	SHA-3 512-bit hash function with ECDSA
SIGN_SHA3_512_RSA	SHA-3 512-bit hash function with RSA
SIGN_SHA384_ECDSA	SHA 384-bit hash function with ECDSA
SIGN_SHA384_RSA	SHA 384-bit hash function with RSA
SIGN_SHA384_RSA_X931	SHA 384-bit hash function with RSA and X931 padding
SIGN_SHA512_ECDSA	SHA 512-bit hash function with ECDSA
SIGN_SHA512_RSA	SHA 512-bit hash function with RSA
SIGN_SHA512_RSA_X931	SHA 512-bit hash function with RSA and X931 padding
SIGN_SM3_SM2	SM3 256-bit hash function with SM2

Usage Notes

You can use the SIGN_SM3_SM2 constant for the SM3_SM2 encryption algorithm

VERIFY Function

This function verifies `RAW` data using the signature, public key assisted with key algorithm, and sign algorithm. It returns `TRUE` if the signature was verified.

Syntax

```
DBMS_CRYPTO.VERIFY(
   src IN RAW,
   sign IN RAW,
   pub_key IN RAW,
   pubkey_alg IN BINARY_INTEGER,
   sign_alg  IN BINARY_INTEGER)
 RETURN BOOLEAN;
```

Parameters

Table 57-28 VERIFY Function Parameters

Parameter Name	Description
`src`	`RAW` data to be verified.
`sign`	Message signature.
`pub_key`	Public key.
`pubkey_alg`	Specify the KEY_TYPE_RSA RSA key type for RSA algorithms and KEY_TYPE_ECDSA ECDSA key type for ECDSA algorithms.
`sign_alg`	Specify one of the algorithms that are listed the Usage Notes.

Usage Notes

Table 57-29 Verify Type Algorithms

Hash Algorithm	Description
`SIGN_SHA1_RSA`	SHA hash function with RSA
`SIGN_SHA1_RSA_X931`	SHA hash function with RSA and X931 padding
`SIGN_SHA224_ECDSA`	SHA 224-bit hash function with ECDSA
`SIGN_SHA224_RSA`	SHA 224-bit hash function with RSA
`SIGN_SHA256_ECDSA`	SHA 256-bit hash function with ECDSA
`SIGN_SHA256_RSA`	SHA 256-bit hash function with RSA
`SIGN_SHA256_RSA_X931`	SHA 256-bit hash function with RSA and X931 padding
`SIGN_SHA3_224_ECDSA`	SHA-3 224-bit hash function with ECDSA
`SIGN_SHA3_224_RSA`	SHA-3 234-bit hash function with RSA
`SIGN_SHA3_256_ECDSA`	SHA-3 256-bit hash function with ECDSA
`SIGN_SHA3_256_RSA`	SHA-3 256-bit hash function with RSA
`SIGN_SHA3_384_ECDSA`	SHA-3 384-bit hash function with ECDSA
`SIGN_SHA3_384_RSA`	SHA-3 384-bit hash function with RSA

Table 57-29 (Cont.) Verify Type Algorithms

Hash Algorithm	Description
SIGN_SHA3_512_ECDSA	SHA-3 512-bit hash function with ECDSA
SIGN_SHA3_512_RSA	SHA-3 512-bit hash function with RSA
SIGN_SHA384_ECDSA	SHA 384-bit hash function with ECDSA
SIGN_SHA384_RSA	SHA 384-bit hash function with RSA
SIGN_SHA384_RSA_X931	SHA 384-bit hash function with RSA and X931 padding
SIGN_SHA512_ECDSA	SHA 512-bit hash function with ECDSA
SIGN_SHA512_RSA	SHA 512-bit hash function with RSA
SIGN_SHA512_RSA_X931	SHA 512-bit hash function with RSA and X931 padding
SIGN_SM3_SM2	SM3 256-bit hash function with SM2

Usage Notes

You can use the SIGN_SM3_SM2 constant for the SM3_SM2 encryption algorithm