# 24 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 encryptio and hashing algorithms, including the Advanced Encryption Standard (AES) encryption algorithm. AES has bee approved by the National Institute of Standards and Technology (NIST) to replace the Data Encryption Standar (DES).

**See Also:**

*Oracle Database Security Guide* for further information about using this package and about encrypting data in general.

This chapter contains the following topics:

- Using the DBMS_CRYPTO Subprograms
  - Overview
  - Security Model
  - Types
  - Algorithms
  - Restrictions
  - Exceptions
  - Operational Notes
- Summary of DBMS_CRYPTO Subprograms

## Using the DBMS_CRYPTO Subprograms

- Overview
- Security Model
- Types
- Algorithms
- Restrictions
- Exceptions

## Overview

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

The `DBMS_CRYPTO` package enables encryption and decryption for common Oracle datatypes, including `RAW` an
large objects (`LOB`s), such as images and sound. Specifically, it supports `BLOB`s and `CLOB`s. In addition, it provid
Globalization Support for encrypting data across different database character sets.

The following cryptographic algorithms are supported:

- Data Encryption Standard (DES), Triple DES (3DES, 2-key and 3-key)
- Advanced Encryption Standard (AES)
- MD5, MD4, and SHA-1 cryptographic hashes
- MD5 and SHA-1 Message Authentication Code (MAC)

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 four block cipher chaining modes, including Cipher Block Chaining (CBC).

Table 24-1 lists the `DBMS_CRYPTO` package features in comparison to the other PL/SQL encryption package, the `DBMS_OBFUSCATION_TOOLKIT`.

***Table 24-1 DBMS_CRYPTO and DBMS_OBFUSCATION_TOOLKIT Feature Comparison***

| Package Feature | DBMS_CRYPTO | DBMS_OBFUSCATION_TOOLKIT |
|---|---|---|
| Cryptographic algorithms | DES, 3DES, AES, RC4, 3DES_2KEY | DES, 3DES |
| Padding forms | PKCS5, zeroes | none supported |
| Block cipher chaining modes | CBC, CFB, ECB, OFB | CBC |
| Cryptographic hash algorithms | MD5, SHA-1, MD4 | MD5 |
| Keyed hash (MAC) algorithms | HMAC_MD5, HMAC_SH1 | none supported |

| | | |
|---|---|---|
| random number generator | TEGER | |
| Database types | RAW, CLOB, BLOB | RAW, VARCHAR2 |

DBMS_CRYPTO is intended to replace the DBMS_OBFUSCATION_TOOLKIT, providing greater ease of use and supp for a range of algorithms to accommodate new and existing systems. Specifically, 3DES_2KEY and MD4 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, or SHA-1.

## Security Model

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

## Types

Parameters for the DBMS_CRYPTO subprograms use these datatypes:

***Table 24-2 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 |

## Algorithms

The following cryptographic algorithms, modifiers, and cipher suites are predefined in this package.

***Table 24-3 DBMS_CRYPTO Cryptographic Hash Functions***

| Name | Description |
|---|---|
| `HASH_MD4` | Produces a 128-bit hash, or message digest of the input message |
| `HASH_MD5` | Also produces a 128-bit hash, but is more complex than MD4 |
| `HASH_SH1` | Secure Hash Algorithm (SHA). Produces a 160-bit hash. |

***Table 24-4 DBMS_CRYPTO MAC (Message Authentication Code) Functions***

| Name | Description |
|---|---|
| HMAC_MD5[Foot 1] | Same as MD5 hash function, except it requires a secret key to verify the hash value. |
| HMAC_SH1[Footref 1] | Same as SHA hash function, except it requires a secret key to verify the hash value. |

[Footnote 1] Complies with IETF RFC 2104 standard

***Table 24-5 DBMS_CRYPTO Encryption Algorithms***

| Name | Description |
|---|---|
| `ENCRYPT_DES` | Data Encryption Standard. Block cipher. Uses key length of 56 bits. |
| `ENCRYPT_3DES_2KEY` | Data Encryption Standard. Block cipher. Operates on a block 3 times with 2 keys. Effective key length of 112 bits. |
| `ENCRYPT_3DES` | Data Encryption Standard. Block cipher. Operates on a block 3 times. |
| ENCRYPT_AES128 | Advanced Encryption Standard. Block cipher. Uses 128-bit key size. |

| | |
|---|---|
| | cipher. Uses 192-bit key size. |
| ENCRYPT_AES256 | Advanced Encryption Standard. Block cipher. Uses 256-bit key size. |
| `ENCRYPT_RC4` | Stream cipher. Uses a secret, randomly generated key unique to each session. |

*Table 24-6 DBMS_CRYPTO Block Cipher Suites*

| Name | Description |
|---|---|
| `DES_CBC_PKCS5` | ENCRYPT_DES[Foot 1] + CHAIN_CBC[Foot 2] + PAD_PKCS5[Foot 3] |
| `DES3_CBC_PKCS5` | `ENCRYPT_3DES`[Footref 1] +`CHAIN_CBC`[Footref 2] + PAD_PKCS5[Footref 3] |

Footnote 1 See Table 24-5, "DBMS_CRYPTO Encryption Algorithms"

Footnote 2 See Table 24-7, "DBMS_CRYPTO Block Cipher Chaining Modifiers"

Footnote 3 See Table 24-8, "DBMS_CRYPTO Block Cipher Padding Modifiers"

*Table 24-7 DBMS_CRYPTO Block Cipher Chaining Modifiers*

| Name | Description |
|---|---|
| `CHAIN_ECB` | Electronic Codebook. Encrypts each plaintext block independently. |
| `CHAIN_CBC` | Cipher Block Chaining. Plaintext is XORed with the previous ciphertext block before it is encrypted. |
| `CHAIN_CFB` | Cipher-Feedback. Enables encrypting units of data smaller than the block size. |
| `CHAIN_OFB` | Output-Feedback. Enables running a block cipher as a synchronous stream cipher. Similar to CFB, except |

| | |
|---|---|
| | of the data queue waiting to be encrypted. |

*Table 24-8 DBMS_CRYPTO Block Cipher Padding Modifiers*

| Name | Description |
|---|---|
| `PAD_PKCS5` | Provides padding which complies with the PKCS #5: Password-Based Cryptography Standard |
| `PAD_NONE` | Provides option to specify no padding. Caller must ensure that blocksize is correct, else the package returns an error. |
| `PAD_ZERO` | Provides padding consisting of zeroes. |

## 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, then convert it to the `RAW`datatype. After performing these conversions, you can then encrypt it with the`DBMS_CRYPTO` package.

**See Also:**

"Conversion Rules" for information about converting datatypes.

## Exceptions

Table 24-9 lists exceptions that have been defined for `DBMS_CRYPTO`.

*Table 24-9 DBMS_CRYPTO Exceptions*

| Exception | Code | Description |
|---|---|---|
| `CipherSuiteInvalid` | `28827` | The specified cipher suite is not defined. |

| | | |
|---|---|---|
| | | 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).<br><br>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. |

## 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 encr
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`** mu
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. Ha
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 extreme
difficult to generate an input message that hashes to a particular value. Note that hash values should be at leas
128 bits in length to be considered secure.

runs `DBMS_CRYPTO.HASH` against the stored data to create a hash value. When she retrieves the stored data at later date, she 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 v the hash value.

MACs can be used to authenticate files between users. They can also be used by a single user to determine if 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 ar 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 use 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 genera For example, you can use the `RANDOMBYTES`function to generate random material for keys. (Calls to the `RANDOMBYTES` function behave like calls to the `DESGETKEY` and `DES3GETKEY` functions of the`DBMS_OBFUSCATION_TOOLKIT` package.)

When generating encryption keys for DES, it is important to remember that some numbers are considered weal and semiweak keys. Keys are considered weak or semiweak when the pattern of the algorithm combines with t pattern of the initial key value to produce ciphertext that is more susceptible to cryptanalysis. To avoid this, filter 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 Advanced Security Administrator's Guide* for information about configuring network encryption and SSL.
- "Key Management" for a full discussion about securely storing encryption keys
- "RANDOMBYTES Function"

## Conversion Rules

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 wish to use.

Syntax example:

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

**See Also:**

Chapter 170, "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.

## Examples

The following listing shows PL/SQL block encrypting and decrypting pre-defined '**`input_string`**' using 256-bit 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
```

```
BEGIN
   DBMS_OUTPUT.PUT_LINE ( 'Original string: ' || input_string);
   key_bytes_raw := DBMS_CRYPTO.RANDOMBYTES (num_key_bytes);
   encrypted_raw := DBMS_CRYPTO.ENCRYPT
      (
         src => UTL_I18N.STRING_TO_RAW (input_string,  'AL32UTF8'),
         typ => encryption_type,
         key => key_bytes_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
      );
   output_string := UTL_I18N.RAW_TO_CHAR (decrypted_raw, 'AL32UTF8');

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

## Summary of DBMS_CRYPTO Subprograms

***Table 24-10 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 |
| 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 |

| | |
|---|---|
| | cryptographic hash algorithms (MD4, MD5, or SHA-1) to data |
| MAC Function | Applies Message Authentication Code algorithms (MD5 or SHA-1) to data to provide keyed message protection |
| 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 |

## DECRYPT Function

This function decrypts `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;
```

### Pragmas

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

### Parameters

***Table 24-11 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`. |

### Usage Notes

- To retrieve original plaintext data, `DECRYPT` must be called with the same cipher, modifiers, key, and IV that wa 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:**
>
> "Conversion Rules" for a discussion of the `VARCHAR2` to `RAW` conversion process.

## 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(
   src IN            BLOB,
   typ IN            PLS_INTEGER,
   key IN            RAW,
   iv  IN            RAW          DEFAULT NULL);
```

## Pragmas

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

## Parameters

***Table 24-12 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. |

## ENCRYPT Function

This function encrypts **`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)
```

## Pragmas

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

## Parameters

***Table 24-13 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`. |

## 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 24-7 and Table 24 8 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 us for encryption and decryption. For example, the following example defines a cipher suite that uses DES, ciphe 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 24-6 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:**
>
> "Conversion Rules" for a discussion of the conversion process.

## 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);
```

### Pragmas

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

### Parameters

***Table 24-14 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. |

ciphers. Default is **`NULL`**.

## Usage Notes

See "Conversion Rules" for usage notes about using 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 (general
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 va
from the input data, but it is hard to generate data that hashes to a particular value. Consequently, one-way has
functions work well to ensure data integrity. Refer to "When to Use Hash or Message Authentication Code (MAC
Functions" for more information about using one-way hash functions.

This function applies to data one of the supported cryptographic hash algorithms listed in Table 24-3.

## 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 24-15 HASH Function Parameters***

|  |  |
|---|---|
| `src` | The source data to be hashed. |
| `typ` | The hash algorithm to be used. |

## Usage Note

Oracle recommends that you use the SHA-1 (Secure Hash Algorithm), specified with the constant, **`HASH_SH1`**, because it is more resistant to brute-force attacks than MD4 or MD5. If you must use a Message Digest algorith then MD5 provides greater security than MD4.

## MAC Function

A Message Authentication Code, or 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 some with the identical key can verify the hash. Also refer to "When to Use Hash or Message Authentication Code (M Functions" for more information about using MACs.

This function applies MAC algorithms to data to provide keyed message protection. See Table 24-4 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 24-16 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. |

## RANDOMBYTES Function

This function returns a `RAW` value containing a cryptographically secure pseudo-random sequence of bytes, whi can be used to generate random material for encryption keys. The `RANDOMBYTES` function is based on the RSA X9.31 PRNG (Pseudo-Random Number Generator), and it draws its entropy (seed) from the`sqlnet.ora` file parameter `SQLNET.CRYPTO_SEED`.

### Syntax

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

### Pragmas

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

### Parameters

*Table 24-17 RANDOMBYTES Function Parameter*

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

### Usage Note

- The **`SQLNET.CRYPTO_SEED`** parameter can be set by entering 10 to 70 random characters with the following syntax in the **`sqlnet.ora`** file:

  ```
  SQLNET.CRYPTO_SEED = <10 to 70 random characters>
  ```

  **See Also:**

  *Oracle Database Advanced Security Administrator's Guide* for more information about the **`SQLNET.CRYPTO_SEED`** parameter and its use.

---

## 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);
```