

SEC Utility - User's Guide v24.0

Author: Philippe Debois (European Commission - DIGIT)

Table of Contents

- [1. Introduction](#)
- [2. Installation](#)
- [3. Components](#)
- [5. Operations](#)
 - [5.1. Encryption](#)
 - [5.2. Decryption](#)
- [6. Example](#)
 - [Save Non-Encrypted Credentials](#)
 - [Encrypt Credentials](#)
 - [Verify Decryption](#)
 - [Check Encrypted Passwords](#)
 - [Check Encryption Keys](#)
 - [Example of Inserted Values](#)

1. Introduction

Applications that need to access protected resources via secure protocols must authenticate using passwords or secrets (e.g., SSH keys, DevOps secrets). Embedding passwords directly in source code (in plain text or base64 encoded) poses significant security risks, as these credentials can be overlooked and potentially accessed by unauthorized individuals if the source code is published in repositories like GitHub.

Oracle Wallet is a secure Oracle container for storing credentials, such as certificates or private keys, but it does not solve the issue outlined above. Passwords stored in an Oracle Wallet cannot be retrieved by a user or application; they are only accessible by Oracle for verification purposes, such as when connecting to a database.

The SEC utility is a PL/SQL solution that allows you to encrypt and securely store credentials (e.g., username and password, or client ID and client secret) in Oracle tables. These credentials can later

be retrieved and decrypted as needed. For security, it is recommended to:

- **Wrap provided packages:** Deploy wrapped packages to prevent easy code visibility, as some online tools can unwrap PL/SQL code.
- **Use separate schemas for packages and tables:** By placing packages and tables in different schemas, you minimize the risk. Even if someone has access to the data, they cannot call stored procedures to decrypt it.

The package's `encrypt` and `decrypt` functions also require a "magic password" as a parameter. If this password is not provided, these functions have no effect.

While encryption alone cannot address all security risks, selectively encrypting sensitive data before storing it in the database enhances security. This utility leverages the Oracle-provided `DBMS_CRYPTO` package.

Feature	Description
Encryption	Uses the <code>DBMS_CRYPTO</code> package to encrypt and decrypt sensitive data before storage in Oracle tables.
Data Types Supported	Enables encryption for common Oracle data types, including RAW and large objects (LOBs), such as images and audio.
Security Recommendations	Wrap packages upon deployment and use separate schemas for packages and tables.
Magic Password Requirement	Encrypt and decrypt functions require a magic password parameter for execution.

Note: The `DBMS_CRYPTO` package does not grant permissions to the DBA group or to PUBLIC by default. Ensure you have the necessary privileges to use this package.

2. Installation

The steps to install this tool, which is part of the EC PL/SQL toolkit, are the following:

- Set `DBM_USERNAME` , `DBM_PASSWORD` , and `DBM_DATABASE` environment variables corresponding to the schema in which you want to install the tool.
- While located in the home directory of the toolkit, execute the following shell command:
`dbm-cli install sec .`

3. Components

The following SEC objects are created in the database schema:

Object Type	Object Name	Description
PACKAGE	SEC_UTILITY_KRN	Main package of the SEC utility (KRN = Kernel)
PACKAGE	SEC_UTILITY_VAR	Global variables of the SEC utility
PACKAGE	SEC_UTILITY_LIC	Licence terms
TABLE	SEC_CRYPTO_CREDENTIALS	Stores username and encrypted password information
TABLE	SEC_CRYPTO_SECRETS	Stores the encrypted key used for encryption/decryption per user

5. Operations

The tool offers two operations described below.

5.1. Encryption

To perform encryption, call:

```
sec_utility_krn.encrypt(username, password, unlock_code);
```

This function executes the following steps:

- 1. Unlock Code Verification:**
The `unlock_code` parameter acts as a "magic password," known only to authorized users. This parameter must match the value of the constant `k_free_password`, which is defined in the `SEC_UTILITY_VAR.PKS` package specification. Without the correct unlock code, the function will be blocked and ineffective.
Important:
`k_free_password` must be a Base64-encoded string. To encode a string, you can use tools like base64decode.org.
- 2. Generating Secure Encryption Material:**
The function creates a secure pseudo-random sequence of bytes, which serves as a foundation for generating encryption keys.
- 3. Encrypting Data:**
The raw data is encrypted using a stream or block cipher and the key provided by the user.

4. Storing Encrypted Information:

The `sec_crypto_secrets` table stores the account name and encryption key used for data encryption.

5. Returning Encrypted Password:

The user's password is returned in its encrypted form.

5.2. Decryption

To perform decryption, execute:

```
sec_utility_krn.decrypt(username, password, unlock_code);
```

The `decrypt` procedure performs the following steps:

1. Unlock Code Verification:

The unlock code provided by the user is verified to ensure authorized access.

2. Decrypting the Encryption Key:

The encryption key stored in the `sec_crypto_secrets` table is decrypted.


3. Returning Decrypted Password:

The procedure returns the decrypted value of the `p_password` parameter.

6. Example

Save Non-Encrypted Credentials

```
INSERT INTO sec_crypto_credentials (username, password) VALUES ('client-id', '.....');  
INSERT INTO sec_crypto_credentials (username, password) VALUES ('client-secret', '....
```



Encrypt Credentials

This will save the encryption key in the `sec_crypto_secrets` table:

```
UPDATE sec_crypto_credentials  
SET password = sec_utility_krn.encrypt(username, password, 'unlock code');
```

Note: The unlock code stored in `SEC_UTILITY_VAR` should be provided.

Verify Decryption

```
SELECT sec_utility_krn.decrypt(username, password, 'unlock code')
FROM sec_crypto_credentials;
```

Note: The unlock code stored in SEC_UTILITY_VAR should be provided.

Check Encrypted Passwords

View the encrypted passwords stored for each user:

```
SELECT * FROM sec_crypto_credentials;
```

Check Encryption Keys

View the encryption keys used to encrypt/decrypt each user's password:

```
SELECT * FROM sec_crypto_secrets;
```

Example of Inserted Values

This example shows the inserted values after encryption:

```
INSERT INTO sec_crypto_credentials (username, password) VALUES ('client-id', '76425957!
INSERT INTO sec_crypto_credentials (username, password) VALUES ('client-secret', '52696
INSERT INTO sec_crypto_secrets (username, secret) VALUES ('client-id', '65DD47671BCB17:
INSERT INTO sec_crypto_secrets (username, secret) VALUES ('client-secret', '7DEED0B62D!
COMMIT;
```

