# PL/SQL Encryption & Security Project

This project implements **encryption, decryption, and security** features in Oracle PL/SQL using **DBMS\_CRYPTO**. It ensures **data protection** for sensitive information like Social Security Numbers (SSNs) and Credit Card Numbers (CCNs). Additionally, it includes **role-based access control** and **audit logging**.

**PL-SQL-Encryption**

PL-SQL-Encryption

│── sql\_scripts/

│ │── 01\_create\_tables.sql

│ │── 02\_create\_functions.sql

│ │── 03\_create\_procedures.sql

│ │── 04\_create\_views.sql

│ │── 05\_create\_roles\_permissions.sql

│ │── 06\_insert\_sample\_data.sql

│ │── 07\_test\_queries.sql

│── packages/

│ │── encryption\_pkg.pks -- Package Specification

│ │── encryption\_pkg.pkb -- Package Body

│── logs/

│ │── security\_audit\_log.txt

**sql\_scripts/**

This folder contains **modular SQL scripts**.

**01\_create\_tables.sql (Creates required tables)**

CREATE TABLE secure\_data (

    id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    user\_id NUMBER UNIQUE NOT NULL,

    encrypted\_ssn RAW(2000) NOT NULL,

    encrypted\_ccn RAW(2000) NOT NULL

);

CREATE TABLE user\_keys (

    user\_id NUMBER PRIMARY KEY,

    key\_value RAW(32) NOT NULL

);

CREATE TABLE security\_audit (

    audit\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    user\_name VARCHAR2(100),

    action VARCHAR2(50),

    accessed\_data VARCHAR2(4000),

    access\_time TIMESTAMP DEFAULT SYSTIMESTAMP

);

**02\_create\_functions.sql (Contains encryption, decryption, and masking functions)**

CREATE OR REPLACE FUNCTION encrypt\_data (p\_plain\_text VARCHAR2, p\_key RAW) RETURN RAW IS

BEGIN

    RETURN DBMS\_CRYPTO.ENCRYPT(UTL\_RAW.CAST\_TO\_RAW(p\_plain\_text), DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC, p\_key);

END encrypt\_data;

/

CREATE OR REPLACE FUNCTION decrypt\_data (p\_encrypted RAW, p\_key RAW) RETURN VARCHAR2 IS

BEGIN

    RETURN UTL\_RAW.CAST\_TO\_VARCHAR2(DBMS\_CRYPTO.DECRYPT(p\_encrypted, DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC, p\_key));

END decrypt\_data;

/

CREATE OR REPLACE FUNCTION mask\_sensitive\_data (p\_data VARCHAR2, p\_mask\_type VARCHAR2) RETURN VARCHAR2 IS

BEGIN

    IF p\_mask\_type = 'SSN' THEN

        RETURN 'XXX-XX-' || SUBSTR(p\_data, -4);

    ELSIF p\_mask\_type = 'CCN' THEN

        RETURN 'XXXX-XXXX-XXXX-' || SUBSTR(p\_data, -4);

    ELSE

        RETURN 'DATA MASKED';

    END IF;

END mask\_sensitive\_data;

/

**03\_create\_procedures.sql (Contains procedures for inserting & retrieving data)**

CREATE OR REPLACE PROCEDURE insert\_encrypted\_data (p\_user\_id NUMBER, p\_ssn VARCHAR2, p\_ccn VARCHAR2) AS

    l\_key RAW(32);

    l\_encrypted\_ssn RAW(2000);

    l\_encrypted\_ccn RAW(2000);

BEGIN

    SELECT key\_value INTO l\_key FROM user\_keys WHERE user\_id = p\_user\_id;

    l\_encrypted\_ssn := encrypt\_data(p\_ssn, l\_key);

    l\_encrypted\_ccn := encrypt\_data(p\_ccn, l\_key);

    INSERT INTO secure\_data (user\_id, encrypted\_ssn, encrypted\_ccn)

    VALUES (p\_user\_id, l\_encrypted\_ssn, l\_encrypted\_ccn);

    COMMIT;

END insert\_encrypted\_data;

/

CREATE OR REPLACE PROCEDURE retrieve\_decrypted\_data (p\_user\_id IN NUMBER) AS

    l\_ssn VARCHAR2(20);

    l\_ccn VARCHAR2(20);

    l\_key RAW(32);

    l\_user VARCHAR2(100);

BEGIN

    l\_user := SYS\_CONTEXT('USERENV', 'SESSION\_USER');

    INSERT INTO security\_audit (user\_name, action, accessed\_data)

    VALUES (l\_user, 'DECRYPTION ATTEMPT', 'User ID: ' || p\_user\_id);

    SELECT key\_value INTO l\_key FROM user\_keys WHERE user\_id = p\_user\_id;

    SELECT decrypt\_data(encrypted\_ssn, l\_key), decrypt\_data(encrypted\_ccn, l\_key)

    INTO l\_ssn, l\_ccn

    FROM secure\_data WHERE user\_id = p\_user\_id;

    DBMS\_OUTPUT.PUT\_LINE('Decrypted SSN: ' || l\_ssn);

    DBMS\_OUTPUT.PUT\_LINE('Decrypted CCN: ' || l\_ccn);

END retrieve\_decrypted\_data;

/

**04\_create\_views.sql (Contains secure views)**

CREATE OR REPLACE VIEW secure\_data\_view AS

SELECT

    sd.id,

    sd.user\_id,

    CASE

        WHEN EXISTS (

            SELECT 1 FROM user\_role\_privs

            WHERE grantee = SYS\_CONTEXT('USERENV', 'SESSION\_USER')

            AND granted\_role = 'DATA\_SECURITY\_ROLE'

        )

        THEN decrypt\_data(sd.encrypted\_ssn, uk.key\_value)

        ELSE mask\_sensitive\_data(decrypt\_data(sd.encrypted\_ssn, uk.key\_value), 'SSN')

    END AS ssn,

    CASE

        WHEN EXISTS (

            SELECT 1 FROM user\_role\_privs

            WHERE grantee = SYS\_CONTEXT('USERENV', 'SESSION\_USER')

            AND granted\_role = 'DATA\_SECURITY\_ROLE'

        )

        THEN decrypt\_data(sd.encrypted\_ccn, uk.key\_value)

        ELSE mask\_sensitive\_data(decrypt\_data(sd.encrypted\_ccn, uk.key\_value), 'CCN')

    END AS ccn

FROM secure\_data sd

JOIN user\_keys uk ON sd.user\_id = uk.user\_id;

/

**05\_create\_roles\_permissions.sql (Defines security roles)**

CREATE ROLE DATA\_SECURITY\_ROLE;

GRANT SELECT, UPDATE, INSERT ON secure\_data TO DATA\_SECURITY\_ROLE;

GRANT EXECUTE ON retrieve\_decrypted\_data TO DATA\_SECURITY\_ROLE;

GRANT SELECT ON secure\_data\_view TO DATA\_SECURITY\_ROLE;

GRANT DATA\_SECURITY\_ROLE TO admin\_user;

**06\_insert\_sample\_data.sql (Inserts test data)**

EXEC insert\_encrypted\_data(1, '123-45-6789', '4111-1111-1111-1111');

EXEC insert\_encrypted\_data(2, '987-65-4321', '5500-0000-0000-0004');

**07\_test\_queries.sql (Test cases)**

SELECT \* FROM secure\_data\_view;

EXEC retrieve\_decrypted\_data(1);

SELECT \* FROM security\_audit ORDER BY access\_time DESC;

**packages/**

If you want to wrap encryption logic in a package:

**encryption\_pkg.pks (Package Specification)**

-- Create a package to encrypt and decrypt data (Package Specification)

CREATE OR REPLACE PACKAGE encryption\_pkg AS

    FUNCTION encrypt\_data(p\_plain\_text VARCHAR2, p\_key RAW) RETURN RAW;

    FUNCTION decrypt\_data(p\_encrypted RAW, p\_key RAW) RETURN VARCHAR2;

    FUNCTION mask\_sensitive\_data(p\_data VARCHAR2, p\_mask\_type VARCHAR2) RETURN VARCHAR2;

END encryption\_pkg;

/

**encryption\_pkg.pkb (Package Body)**

-- Create a package to encrypt and decrypt data (Package Body)

REATE OR REPLACE PACKAGE BODY encryption\_pkg AS

    FUNCTION encrypt\_data(p\_plain\_text VARCHAR2, p\_key RAW) RETURN RAW IS

    BEGIN

        RETURN DBMS\_CRYPTO.ENCRYPT(UTL\_RAW.CAST\_TO\_RAW(p\_plain\_text), DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC, p\_key);

    END encrypt\_data;

    FUNCTION decrypt\_data(p\_encrypted RAW, p\_key RAW) RETURN VARCHAR2 IS

    BEGIN

        RETURN UTL\_RAW.CAST\_TO\_VARCHAR2(DBMS\_CRYPTO.DECRYPT(p\_encrypted, DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC, p\_key));

    END decrypt\_data;

    FUNCTION mask\_sensitive\_data(p\_data VARCHAR2, p\_mask\_type VARCHAR2) RETURN VARCHAR2 IS

    BEGIN

        IF p\_mask\_type = 'SSN' THEN

            RETURN 'XXX-XX-' || SUBSTR(p\_data, -4);

        ELSIF p\_mask\_type = 'CCN' THEN

            RETURN 'XXXX-XXXX-XXXX-' || SUBSTR(p\_data, -4);

        ELSE

            RETURN 'DATA MASKED';

        END IF;

    END mask\_sensitive\_data;

END encryption\_pkg;

/

## Step-by-Step Guide: PL/SQL Encryption & Security Project

This project implements **encryption, decryption, and security** features in Oracle PL/SQL using **DBMS\_CRYPTO**. It ensures **data protection** for sensitive information like Social Security Numbers (SSNs) and Credit Card Numbers (CCNs). Additionally, it includes **role-based access control** and **audit logging**.

**3. Installation & Setup**

**Step 1: Create Database Tables**

Run the script **01\_create\_tables.sql** to create necessary tables.

@sql\_scripts/01\_create\_tables.sql;

**Step 2: Create Functions**

Run the script **02\_create\_functions.sql** to define encryption, decryption, and masking logic.

@sql\_scripts/02\_create\_functions.sql;

**Step 3: Create Stored Procedures**

Run the script **03\_create\_procedures.sql** to define procedures for inserting and retrieving encrypted data.

@sql\_scripts/03\_create\_procedures.sql;

**Step 4: Create Secure Views**

Run the script **04\_create\_views.sql** to create a secure view that conditionally decrypts or masks data.

@sql\_scripts/04\_create\_views.sql;

**Step 5: Set Up Roles & Permissions**

Run the script **05\_create\_roles\_permissions.sql** to define security roles and grant access permissions.

@sql\_scripts/05\_create\_roles\_permissions.sql;

**Step 6: Insert Sample Data**

Run the script **06\_insert\_sample\_data.sql** to populate tables with encrypted data.

@sql\_scripts/06\_insert\_sample\_data.sql;

**Step 7: Run Test Queries**

Use **07\_test\_queries.sql** to test encryption, decryption, and security audit logs.

@sql\_scripts/07\_test\_queries.sql;

**4. Package Details**

**encryption\_pkg.pks (Package Specification)**

-- Create a package to encrypt and decrypt data (Package Specification)

CREATE OR REPLACE PACKAGE encryption\_pkg AS

    FUNCTION encrypt\_data(p\_plain\_text VARCHAR2, p\_key RAW) RETURN RAW;

    FUNCTION decrypt\_data(p\_encrypted RAW, p\_key RAW) RETURN VARCHAR2;

    FUNCTION mask\_sensitive\_data(p\_data VARCHAR2, p\_mask\_type VARCHAR2) RETURN VARCHAR2;

END encryption\_pkg;

/

**encryption\_pkg.pkb (Package Body)**

-- Create a package to encrypt and decrypt data (Package Body)

REATE OR REPLACE PACKAGE BODY encryption\_pkg AS

    FUNCTION encrypt\_data(p\_plain\_text VARCHAR2, p\_key RAW) RETURN RAW IS

    BEGIN

        RETURN DBMS\_CRYPTO.ENCRYPT(UTL\_RAW.CAST\_TO\_RAW(p\_plain\_text), DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC, p\_key);

    END encrypt\_data;

    FUNCTION decrypt\_data(p\_encrypted RAW, p\_key RAW) RETURN VARCHAR2 IS

    BEGIN

        RETURN UTL\_RAW.CAST\_TO\_VARCHAR2(DBMS\_CRYPTO.DECRYPT(p\_encrypted, DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC, p\_key));

    END decrypt\_data;

    FUNCTION mask\_sensitive\_data(p\_data VARCHAR2, p\_mask\_type VARCHAR2) RETURN VARCHAR2 IS

    BEGIN

        IF p\_mask\_type = 'SSN' THEN

            RETURN 'XXX-XX-' || SUBSTR(p\_data, -4);

        ELSIF p\_mask\_type = 'CCN' THEN

            RETURN 'XXXX-XXXX-XXXX-' || SUBSTR(p\_data, -4);

        ELSE

            RETURN 'DATA MASKED';

        END IF;

    END mask\_sensitive\_data;

END encryption\_pkg;

/

**5. Logs & Documentation**

* **logs/security\_audit\_log.txt**: Stores access logs.

**6. Cleanup & Reset**

To remove all objects and reset the database, run:

@cleanup.sh;

**7. Conclusion**

This guide ensures a **structured**, **secure**, and **efficient** implementation of PL/SQL encryption. Let me know if you need any modifications or additional enhancements!