日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2022年1月17日月曜日

DBのRSA暗号の互換性の確認

RSA暗号がOracle Database 21c（19cにはバックポート）のDBMS_CRYPTOパッケージに追加されています。

DBMS_CRYPTO.PKENCRYPT/PKDECRYPTで暗号化/復号する、Javaに関してはjavax.crypto.Cipherで暗号化/復号するのは問題なくできます。また、DBMS_CRYPTO.SIGN/VERIFYでの署名/検証、java.security.Signatureによる署名/検証も同様です。

DBMS_CRYPTO.PKENCRYPTによる暗号化、javax.crypto.Cipherによる復号、またはその逆、およびDBMS_CRYPTO.SIGNによる署名、java.security.Signatureによる検証、またはその逆の互換性について確認した作業について記載します。

署名と検証

DBMS_CRYPTO.SIGNおよびVERIFYがサポートしている署名のハッシュ・アルゴリズムはマニュアルの以下に記載されています。

https://docs.oracle.com/en/database/oracle/oracle-database/21/arpls/DBMS_CRYPTO.html#GUID-E33DC872-3C26-4E7F-85F8-4D865FE73805

Table 47-22 SHA Hash Algorithmsまたは**Table 47-24 SHA Hash Algorithms**にリストがあります。

JavaのSignatureクラスで利用できる標準のアルゴリズムは以下にリストされています。

https://docs.oracle.com/en/java/javase/17/docs/specs/security/standard-names.html#signature-algorithms

DB	Java	結果
SIGN_SHA1_RSA	SHA1withRSA	○
SIGN_SHA1_RSA_X931	N/A	N/A
SIGN_SHA224_RSA	SHA224withRSA	○
SIGN_SHA256_RSA	SHA256withRSA	○
SIGN_SHA256_RSA_X931	N/A	N/A
SIGN_SHA384_RSA	SHA384withRSA	○
SIGN_SHA384_RSA_X931	N/A	N/A
SIGN_SHA512_RSA	SHA512withRSA	○
SIGN_SHA512_RSA_X931	N/A	N/A

検証に使用したスクリプトは以下になります。DBとJavaの実装で、同じ公開鍵、秘密鍵を使用するため、JavaはOracle Databaseに実装されているJavaを使っています。DBMS_CRYPTOパッケージは秘密鍵のフォーマットとして、PKCS#1とPKCS#8の形式の両方をサポートしていたので、Javaで扱いやすいPKCS#8のフォーマットを採用しています。

PL/SQLのテスト・スクリプト

```
/*
 * Steps to prepare RSA key pair using OpenSSL
 *
 * Generate key pair in PKCS#1
 * openssl genrsa -out private.pem 2048
 *
 * Get public key for l_pubkey
 * openssl rsa -in private.pem -outform PEM -pubout -out public.pem
 *
 * Get private key for l_prvkey in PKCS#8
 * openssl pkcs8 -topk8 -nocrypt -in private.pem -outform PEM -out pk8.pem
 */
set echo off
set serveroutput on
set pages 1000 lines 2000

accept OP_SIGN number default 0 prompt 'Sign   - DB 0, otherwise Java: '
accept OP_VERI number default 0 prompt 'Verify - DB 0, otherwise Java: '
prompt Candidates:
prompt SIGN_SHA1_RSA
prompt SIGN_SHA1_RSA_X931
prompt SIGN_SHA224_RSA
prompt SIGN_SHA256_RSA
prompt SIGN_SHA256_RSA_X931
prompt SIGN_SHA384_RSA
prompt SIGN_SHA384_RSA_X931
prompt SIGN_SHA512_RSA
prompt SIGN_SHA512_RSA_X931
accept OP_ALG char   default 'SIGN_SHA1_RSA' prompt 'SHA Hash Algorithm: '

declare
    l_pubkey varchar2(4000) := regexp_replace(
q'~
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxopp+WyhYyeeiDJXGTmj
JueYELRMT7S3KdDwY1JqXzFrR5aDgsxP81nXW/KuzwKWtaylbSa7q4HtI11mHnW0
/WTDiCQE8UvmUeXuOzp8X81HETIZY/bp/TC2/NrX5oU1ZHxMemDJOGG6HQYfSwQe
BbgAf0r4g+2kQ8j0AF+eUwAzjtruRfMj6eP3bqg2H4Wdf//j1asntdUkvOJZkk36
WLNXZnzXlfYkfQ2DkLzwbScO2vLY7vXfH13tPPUqn86UC3LTF58DKmQV3ZN0C9ra
ZLPHya0qD2w1/8sjuRx9hWe0MNeRrrr1qCo5AWiYG+O9I+zu0A9iSNuwqBa2Xwv6
lwIDAQAB
-----END PUBLIC KEY-----
~'
, '(-+((BEGIN|END) (PUBLIC|PRIVATE) KEY)-+\s?|\s)'
, '');
```

```
   l_prvkey varchar2(4000) := regexp_replace(
q'~
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDGimn5bKFjJ56I
MlcZOaMm55gQtExPtLcp0PBjUmpfMWtHloOCzE/zWddb8q7PApa1rKVtJrurge0j
XWYedbT9ZMOIJATxS+ZR5e47OnxfzUcRMhlj9un9MLb82tfmhTVkfEx6YMk4Ybod
Bh9LBB4FuAB/SviD7aRDyPQAX55TADOO2u5F8yPp4/duqDYfhZ1//+PVqye11SS8
4lmSTfpYs1dmfNeV9iR9DYOQvPBtJw7a8tju9d8fXe089SqfzpQLctMXnwMqZBXd
k3QL2tpks8fJrSoPbDX/yyO5HH2FZ7Qw15GuuvWoKjkBaJgb470j7O7QD2JI27Co
FrZfC/qXAgMBAAECggEAV0yrdnpWqHPWWIrSE4xtegDhs46j83RCCcgOu42UqlDf
nDXJk+zkxttdP9gBF5NK042bm9gpMyvJKwC5k3EsaTMyax1zcupt1tskmrSH1dQ8
iIof8f2zO5p9C8fqoTk3lfy14CXItNr77CEyB1lDHj6fMVm01sXasdedusBQrKum
FmQ/k47lxElc/ljOBv8737Hs/PFkZmBZuAAGO3FERevWv7pqy/wYPIh6Iutd/Bk5
Zc+lh/03Qv1lG7os7U4QOF7kte4gUMTy0H66l+kkUccydphUNb3NrFu3br0K+CTx
I3mzg319Xt+nsPdciK4VmEsRIjN2xEmgJ8G9ETeceQKBgQD0cI0jcI06zYdg9Ej/
3e/7ugbFxqz7EructzJPWwuW1WAltR7TdqRkoeCYtkOdUWp+CLxV+q+oiKiIqsgz
RVnzX/unjsALv3CHsfYoWsX7ZbAJwjCyiMW0R8moL4VP4P7EnnY8b0vad3qd4Fyj
Vhk+0uMK1VETcfrHLqS1xzM1PQKBgQDP7ikRTyu/GTBl56I3y3lfy43HdkMJjsIm
pT19+75M4noJTYE1XF5jjvKLwB7dA4/03lIpzRmCLYU3Sju+J/qSD6wJMSoH9cYV
a4AzcUyfbzOa8NElxOgpfCc43txE3McFHqwSOlWzQdcNn/eE8ebUbmbvHf9NDlOo
em6ucAY0YwKBgGwkRyzApJq5RwVrzcf1JjDV0h05klnZpcrleYqGGxB2Af1sreCc
AorK2xR0vmEhHd34e2oGaCRFoVolRED6k5sKgclVbD1GSsCdyOt9jU11ZXZLxUOX
IvBpOhwAOhhm0A+ok2KVUpeqeELUeeeBNdYHco7eZOoXaP0p85DAOjV5AoGAMq2E
OuDF88SK5DtpcRnZkjyFaLjtK9YsgcjCTQNfZVPJlJzcQgoqQy+i8+a2Xv2wR2ks
M132uqCnmEsydm0+B/1j27Ws3dXMGoph66fQmgc7V6ccAo2UXBATayv+GZaJtzi7
jZHEM3V3ma2EYVWoAdyKn/r6r2gvgZ+dfEx3ZqsCgYEAnD5iRF3dZn0GiFmS4KSe
F2vsYn1EzHg3XjJYQV6Lt2be55PT6tuILiChRYMb889AU2VOyOgZRtdHclLZy3wW
bDTlXa5vKWqpFK7gm38eX3AAsQofoRIERJZBExlo9PqdKHyE4SDj3gSZwhN6ZpU4
2wh7Utg1DvPmbCtEpChJYzA=
-----END PRIVATE KEY-----
~'
, '(-+((BEGIN|END) (PUBLIC|PRIVATE) KEY)-+\s?|\s)'
, '');
--
    l_in raw(2000);
    l_out raw(2000);
    l_plain     varchar2(4000);
    l_signature varchar2(4000);
    SIGN constant integer := &OP_SIGN;
    VERI constant integer := &OP_VERI;
    l_alg_db binary_integer;
    l_alg_java varchar2(80);
    l_verify boolean := false;
begin
    -- Java does not support ANSI X9.31
    if '&OP_ALG' = 'SIGN_SHA1_RSA' then
        l_alg_db   := dbms_crypto.sign_sha1_rsa;
```

```
    l_alg_java := 'SHA1withRSA';
elsif '&OP_ALG' = 'SIGN_SHA1_RSA_X931' then
    l_alg_db   := dbms_crypto.sign_sha1_rsa_x931;
    l_alg_java := 'SHA1withRSA'; -- will fail
elsif '&OP_ALG' = 'SIGN_SHA224_RSA' then
    l_alg_db   := dbms_crypto.sign_sha224_rsa;
    l_alg_java := 'SHA224withRSA';
elsif '&OP_ALG' = 'SIGN_SHA256_RSA' then
    l_alg_db   := dbms_crypto.sign_sha256_rsa;
    l_alg_java := 'SHA256withRSA';
elsif '&OP_ALG' = 'SIGN_SHA256_RSA_X931' then
    l_alg_db   := dbms_crypto.sign_sha256_rsa_x931;
    l_alg_java := 'SHA256withRSA'; -- will fail
elsif '&OP_ALG' = 'SIGN_SHA384_RSA' then
    l_alg_db   := dbms_crypto.sign_sha384_rsa;
    l_alg_java := 'SHA384withRSA';
elsif '&OP_ALG' = 'SIGN_SHA384_RSA_X931' then
    l_alg_db   := dbms_crypto.sign_sha384_rsa_x931;
    l_alg_java := 'SHA384withRSA'; -- will fail
elsif '&OP_ALG' = 'SIGN_SHA512_RSA' then
    l_alg_db   := dbms_crypto.sign_sha512_rsa;
    l_alg_java := 'SHA512withRSA';
elsif '&OP_ALG' = 'SIGN_SHA512_RSA_X931' then
    l_alg_db   := dbms_crypto.sign_sha512_rsa_x931;
    l_alg_java := 'SHA512withRSA'; -- will fail
else
    l_alg_db   := 0; -- will throw exception during verify.
    l_alg_java := 'NoAlg';
end if;
dbms_output.put_line('Alg Chosen for DB:   ' || l_alg_db);
dbms_output.put_line('Alg Chosen for Java: ' || l_alg_java);

l_plain := 'Text to be signed.';

-- sign
if SIGN = 0 then
    l_in := utl_i18n.string_to_raw(l_plain,'AL32UTF8');
    l_out := dbms_crypto.sign(
        src => l_in
        , prv_key => utl_i18n.string_to_raw(l_prvkey,'AL32UTF8')
        , pubkey_alg => dbms_crypto.key_type_rsa
        , sign_alg => l_alg_db
    );
    l_signature := utl_raw.cast_to_varchar2(utl_encode.base64_encode(l_out));
    l_signature := replace(l_signature, chr(13)||chr(10),'');
else
    l_signature := signatureutl_sign(l_plain, l_prvkey, l_alg_java);
```

```
    end if;
    dbms_output.put_line('SIGNATURE' || chr(13) || chr(10) || l_signature);
    -- verify
    if VERI = 0 then
        l_in := utl_i18n.string_to_raw(l_plain,'AL32UTF8');
        l_out := utl_encode.base64_decode(utl_raw.cast_to_raw(l_signature));
        l_verify := dbms_crypto.verify(
            src => l_in
            , sign => l_out
            , pub_key => utl_i18n.string_to_raw(l_pubkey,'AL32UTF8')
            , pubkey_alg => dbms_crypto.key_type_rsa
            , sign_alg => l_alg_db
        );
    else
        l_verify := signatureutl_verify(l_plain, l_pubkey, l_signature, l_alg_java);
    end if;
    if l_verify then
        dbms_output.put_line('True');
    else
        dbms_output.put_line('False');
    end if;
end;
/

exit;
```

sign-and-verify.sql hosted with ❤️ by GitHub view raw

Javaによる署名と検証の実装

```
import java.util.Base64;
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.PrivateKey;
import java.security.spec.X509EncodedKeySpec;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.Signature;

/*
 * Utility Class for Sign/Verify with RSA
 *
 * How to load Java into schema APEXDEV
 * loadjava -force -verbose -resolve -u apexdev/****@localhost/xepdb1 SignatureUtl.java
 *
 * Wrapper Function:
--
create or replace function signatureutl_verify
```

```
(
    text varchar2,
    key  varchar2,
    sig  varchar2,
    alg  varchar2
) return boolean
as
language java name 'SignatureUtl.verify(java.lang.String, java.lang.String, java.lang.String, j
/
create or replace function signatureutl_sign
(
    text varchar2,
    key  varchar2,
    alg  varchar2
) return varchar2
as
language java name 'SignatureUtl.sign(java.lang.String, java.lang.String, java.lang.String) ret
/
--
 */
public class SignatureUtl {
  /*
   * verify
   */
  public static boolean verify(String text, String key, String sig, String alg)
  throws Exception
  {
    byte[] data   = text.getBytes("UTF-8");          // plain text into bytes
    byte[] pkdata = Base64.getDecoder().decode(key); // PEM text key into DER
    byte[] dsig   = Base64.getDecoder().decode(sig); // Base64 signature into bytes
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    PublicKey  publicKey  = keyFactory.generatePublic(new X509EncodedKeySpec(pkdata));
    Signature signature = Signature.getInstance(alg);
    signature.initVerify(publicKey);
    signature.update(data);
    return signature.verify(dsig);
  }
  /*
   * sign - privare key in PKCS#8
   */
  public static String sign(String text, String key, String alg)
  throws Exception
  {
    byte[] data   = text.getBytes("UTF-8");          // plain text into bytes
    byte[] pkdata = Base64.getDecoder().decode(key);  // PEM text key into DER
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    PrivateKey privateKey = keyFactory.generatePrivate(new PKCS8EncodedKeySpec(pkdata));
```

```
    Signature signature = Signature.getInstance(alg);
    signature.initSign(privateKey);
    signature.update(data);
    byte[] bytes = signature.sign();
    return Base64.getEncoder().encodeToString(bytes);
  }
}
```

SignatureUtl.java hosted with ❤️ by GitHub view raw

暗号化と復号

DBMS_CRYPTO.PKENCRYPT/PKDECRYPTがサポートしているアルゴリズムは
PKENCRYPT_RSA_PKCS1_OAEPのみで詳しい仕様について記載がありません。

https://docs.oracle.com/en/database/oracle/oracle-database/21/arpls/DBMS_CRYPTO.html#GUID-C588E52C-CEEF-4DDE-9B4A-DBFCE6F1B5E3

テスト・スクリプトをJavaで記述し暗号化/復号して確認したところ、ハッシュ・アルゴリズムとしてSHA-256が使用されていることが確認できています。

以下のスクリプトでテストしています。

PL/SQLのテスト・スクリプト

```
/*
 * Steps to prepare RSA key pair using OpenSSL
 *
 * Generate key pair in PKCS#1
 * openssl genrsa -out private.pem 2048
 *
 * Get public key for l_pubkey
 * openssl rsa -in private.pem -outform PEM -pubout -out public.pem
 *
 * Get private key for l_prvkey in PKCS#8
 * openssl pkcs8 -topk8 -nocrypt -in private.pem -outform PEM -out pk8.pem
 */
set echo off
set serveroutput on
set pages 1000 lines 2000

accept OP_ENC number default 0 prompt 'Encrypt - DB 0, otherwise Java: '
accept OP_DEC number default 0 prompt 'Decrypt - DB 0, otherwise Java: '
prompt Candidates:
prompt SHA-1
prompt SHA-224
prompt SHA-256 - For Database
prompt SHA-384
```

```
prompt SHA-512
accept OP_ALG char   default 'SHA-256' prompt 'Java Padding Algorithm: '

declare
    l_pubkey varchar2(4000) := regexp_replace(
q'~
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxopp+WyhYyeeiDJXGTmj
JueYELRMT7S3KdDwY1JqXzFrR5aDgsxP81nXW/KuzwKWtaylbSa7q4HtI11mHnW0
/WTDiCQE8UvmUeXuOzp8X81HETIZY/bp/TC2/NrX5oU1ZHxMemDJOGG6HQYfSwQe
BbgAf0r4g+2kQ8j0AF+eUwAzjtruRfMj6eP3bqg2H4Wdf//j1asntdUkvOJZkk36
WLNXZnzXlfYkfQ2DkLzwbScO2vLY7vXfH13tPPUqn86UC3LTF58DKmQV3ZN0C9ra
ZLPHya0qD2w1/8sjuRx9hWe0MNeRrrr1qCo5AWiYG+O9I+zu0A9iSNuwqBa2Xwv6
lwIDAQAB
-----END PUBLIC KEY-----
~'
, '(-+((BEGIN|END) (PUBLIC|PRIVATE) KEY)-+\s?|\s)'
, '');
   l_prvkey varchar2(4000) := regexp_replace(
q'~
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDGimn5bKFjJ56I
MlcZOaMm55gQtExPtLcp0PBjUmpfMWtHloOCzE/zWddb8q7PApa1rKVtJrurge0j
XWYedbT9ZMOIJATxS+ZR5e47OnxfzUcRMhlj9un9MLb82tfmhTVkfEx6YMk4Ybod
Bh9LBB4FuAB/SviD7aRDyPQAX55TADOO2u5F8yPp4/duqDYfhZ1//+PVqye11SS8
4lmSTfpYs1dmfNeV9iR9DYOQvPBtJw7a8tju9d8fXe089SqfzpQLctMXnwMqZBXd
k3QL2tpks8fJrSoPbDX/yyO5HH2FZ7Qw15GuuvWoKjkBaJgb470j7O7QD2JI27Co
FrZfC/qXAgMBAAECggEAV0yrdnpWqHPWWIrSE4xtegDhs46j83RCCcgOu42UqlDf
nDXJk+zkxttdP9gBF5NK042bm9gpMyvJKwC5k3EsaTMyax1zcupt1tskmrSH1dQ8
iIof8f2zO5p9C8fqoTk3lfy14CXItNr77CEyB1lDHj6fMVm01sXasdedusBQrKum
FmQ/k47lxElc/ljOBv8737Hs/PFkZmBZuAAGO3FERevWv7pqy/wYPIh6Iutd/Bk5
Zc+lh/03Qv1lG7os7U4QOF7kte4gUMTy0H66l+kkUccydphUNb3NrFu3br0K+CTx
I3mzg319Xt+nsPdciK4VmEsRIjN2xEmgJ8G9ETeceQKBgQD0cI0jcI06zYdg9Ej/
3e/7ugbFxqz7EructzJPWwuW1WAltR7TdqRkoeCYtkOdUWp+CLxV+q+oiKiIqsgz
RVnzX/unjsALv3CHsfYoWsX7ZbAJwjCyiMW0R8moL4VP4P7EnnY8b0vad3qd4Fyj
Vhk+0uMK1VETcfrHLqS1xzM1PQKBgQDP7ikRTyu/GTBl56I3y3lfy43HdkMJjsIm
pT19+75M4noJTYE1XF5jjvKLwB7dA4/03lIpzRmCLYU3Sju+J/qSD6wJMSoH9cYV
a4AzcUyfbzOa8NElxOgpfCc43txE3McFHqwSOlWzQdcNn/eE8ebUbmbvHf9NDlOo
em6ucAY0YwKBgGwkRyzApJq5RwVrzcf1JjDV0h05klnZpcrleYqGGxB2Af1sreCc
AorK2xR0vmEhHd34e2oGaCRFoVolRED6k5sKgclVbD1GSsCdyOt9jU11ZXZLxUOX
IvBpOhwAOhhm0A+ok2KVUpeqeELUeeeBNdYHco7eZOoXaP0p85DAOjV5AoGAMq2E
OuDF88SK5DtpcRnZkjyFaLjtK9YsgcjCTQNfZVPJlJzcQgoqQy+i8+a2Xv2wR2ks
M132uqCnmEsydm0+B/1j27Ws3dXMGoph66fQmgc7V6ccAo2UXBATayv+GZaJtzi7
jZHEM3V3ma2EYVWoAdyKn/r6r2gvgZ+dfEx3ZqsCgYEAnD5iRF3dZn0GiFmS4KSe
F2vsYn1EzHg3XjJYQV6Lt2be55PT6tuILiChRYMb889AU2VOyOgZRtdHclLZy3wW
bDTlXa5vKWqpFK7gm38eX3AAsQofoRIERJZBExlo9PqdKHyE4SDj3gSZwhN6ZpU4
2wh7Utg1DvPmbCtEpChJYzA=
```

```
-----END PRIVATE KEY-----
~'
, '(-+((BEGIN|END) (PUBLIC|PRIVATE) KEY)-+\s?|\s)'
, '');
--
    l_in raw(2000);
    l_out raw(2000);
    l_plain  varchar2(4000);
    l_cipher varchar2(4000);
    ENC constant integer := &OP_ENC;
    DEC constant integer := &OP_DEC;
    l_alg varchar2(80)  := '&OP_ALG';
begin
    -- construct plain text to indicate the method chosen
    if ENC = 0 then
        l_plain := 'TEST STRING: PKENCRYPT => ';
    else
        l_plain := 'TEST STRING: Java ' || l_alg || ' => ';
    end if;
    if DEC = 0 then
        l_plain := l_plain || 'PKDECRYPT';
    else
        l_plain := l_plain || 'Java ' || l_alg;
    end if;
    -- encryption
    dbms_output.put_line('PLAIN TEXT' || chr(13) || chr(10) || l_plain);
    if ENC = 0 then
        l_in := utl_i18n.string_to_raw(l_plain,'AL32UTF8');
        l_out := dbms_crypto.pkencrypt(
            src => l_in
            , pub_key => utl_i18n.string_to_raw(l_pubkey,'AL32UTF8')
            , pubkey_alg => dbms_crypto.key_type_rsa
            , enc_alg => dbms_crypto.pkencrypt_rsa_pkcs1_oaep
        );
        l_cipher := utl_raw.cast_to_varchar2(utl_encode.base64_encode(l_out));
        l_cipher := replace(l_cipher, chr(13)||chr(10),'');
    else
        l_cipher := rsautl_encrypt(l_plain, l_pubkey, l_alg);
    end if;
    dbms_output.put_line('ENCRYPTED/BASE64' || chr(13) || chr(10) || l_cipher);
    -- decryption
    if DEC = 0 then
        l_in := utl_encode.base64_decode(utl_raw.cast_to_raw(l_cipher));
        l_out := dbms_crypto.pkdecrypt(
            src => l_in
            , prv_key => utl_i18n.string_to_raw(l_prvkey,'AL32UTF8')
            , pubkey_alg => dbms_crypto.key_type_rsa
```

```
            , enc_alg => dbms_crypto.pkencrypt_rsa_pkcs1_oaep
        );
        l_plain := utl_i18n.raw_to_char(l_out,'AL32UTF8');
    else
        l_plain := rsautl_decrypt(l_cipher, l_prvkey, l_alg);
    end if;
    dbms_output.put_line('DECRYPTED' || chr(13) || chr(10) || l_plain);
end;
/

exit;
```

test-rsa-enc-dec.sql hosted with ❤️ by GitHub view raw

Javaによる暗号化と復号の実装

```
import java.util.Base64;
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.PrivateKey;
import java.security.spec.X509EncodedKeySpec;
import java.security.spec.PKCS8EncodedKeySpec;
import javax.crypto.Cipher;
import javax.crypto.spec.OAEPParameterSpec;
import java.security.spec.MGF1ParameterSpec;
import javax.crypto.spec.PSource;

/*
 * Utility Class for Encrypt/Decrypt with RSA
 *
 * How to load Java into schema APEXDEV
 * loadjava -force -verbose -resolve -u apexdev/****@localhost/xepdb1 RSAUtl.java
 *
 * Wrapper Function:
--
create or replace function rsautl_encrypt
(
    text varchar2,
    key  varchar2
    alg varchar2    // Valid only SHA-256
) return varchar2
as
language java name 'RSAUtl.encrypt(java.lang.String, java.lang.String, java.lang.String) return
/
create or replace function rsautl_decrypt
(
    text varchar2,
```

```
    key  varchar2,
    alg varchar2  // Valid only SHA-256
) return varchar2
as
language java name 'RSAUtl.decrypt(java.lang.String, java.lang.String, java.lang.String) return
/
--
 */
public class RSAUtl {
  /*
   * Encryption
   */
  public static String encrypt(String text, String key, String alg)
  throws Exception
  {
    MGF1ParameterSpec mgf  = new MGF1ParameterSpec(alg);
    OAEPParameterSpec spec = new OAEPParameterSpec(alg,"MGF1",mgf,PSource.PSpecified.DEFAULT);
    byte[] data   = text.getBytes("UTF-8");          // plain text into bytes
    byte[] pkdata = Base64.getDecoder().decode(key); // PEM text key into DER
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    PublicKey  publicKey  = keyFactory.generatePublic(new X509EncodedKeySpec(pkdata));
    Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPPadding");
    cipher.init(Cipher.ENCRYPT_MODE, publicKey, spec);
    return Base64.getEncoder().encodeToString(cipher.doFinal(data));
  }
  /*
   * Decryption - PKCS#8
   */
  public static String decrypt(String text, String key, String alg)
  throws Exception
  {
    MGF1ParameterSpec mgf  = new MGF1ParameterSpec(alg);
    OAEPParameterSpec spec = new OAEPParameterSpec(alg,"MGF1",mgf,PSource.PSpecified.DEFAULT);
    byte[] data   = Base64.getDecoder().decode(text); // base64 cipher text into bytes
    byte[] pkdata = Base64.getDecoder().decode(key);  // PEM text key into DER
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    PrivateKey privateKey = keyFactory.generatePrivate(new PKCS8EncodedKeySpec(pkdata));
    Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPPadding");
    cipher.init(Cipher.DECRYPT_MODE, privateKey, spec);
    return new String(cipher.doFinal(data),"UTF-8");
  }
}
```

RSAUtl.java hosted with ❤ by GitHub　　view raw

検証コードを書いていて気がついた点を以下に記載します。Javaのコードですが、最初は以下のようにjavax.crypto.Cipherを初期化していました。

```
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
PublicKey  publicKey  = keyFactory.generatePublic(new X509EncodedKeySpec(pkdata));
Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-256AndMGF1Padding");
cipher.init(Cipher.ENCRYPT_MODE, publicKey);
```

この方法で初期化するとデータベースで暗号化したテキストをJavaで復号、またはその逆はできませんでした。javax.crypto.BadPaddingExceptionが発生します。

成功した初期化手順は以下になります。PSource.PSpecified.DEFAULTの指定が必要になります。

```
MGF1ParameterSpec mgf  = new MGF1ParameterSpec(alg);
OAEPParameterSpec spec = new OAEPParameterSpec(alg,"MGF1",mgf,PSource.PSpecified.DEFAULT);
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
PublicKey  publicKey  = keyFactory.generatePublic(new X509EncodedKeySpec(pkdata));
Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPPadding");
cipher.init(Cipher.ENCRYPT_MODE, publicKey, spec);
```

以上になります。

完

Yuji N. 時刻: 16:06

共有

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示