# 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。

2020年3月14日土曜日

## RS256を使ったJWTを作成する

以前にこちらの記事にて、APEX_JWTパッケージを使わずにJWTを生成する方法について紹介しました。APEX_JWTパッケージでは、JSON　Webアルゴリズムとして、HMAC SHA-256 ("HS256")のみがサポートされていて、RSASSA-PKCS1-v1_5 with SHA-256 ("RS256")には対応していません。RS256を指定したJWTを生成するために、APEX_JWTパッケージが持つ機能の一部であっても、活用する方法はありませんでした。

以下に、RS256をデジタル署名としたJWTを生成する方法を記載します。Oracle Databaseはいくつかの暗号処理が実装されていますが、RSA公開鍵暗号方式については未実装です。そのため、デジタル署名の生成については、データベースに組み込みのJavaにて実装します。

RSAの公開鍵/秘密鍵はこちらの記事で紹介したのと、同じ方法で生成します。

```
mkdir -p ~/.oci && openssl genrsa -out ~/.oci/poa_oci_api_key.pem 2048
```

実際のデータの部分だけを一行にして取り出せるように、こちらの記事で紹介したスクリプトも以下に紹介しておきます。

```
#!/bin/sh

while read l
do
  test ${l:0:1} != "-" && /bin/echo -n $l
done < ~/.oci/poa_oci_api_key.pem
echo
```

### デジタル署名のJavaの実装をデータベースに登録する

デジタル署名を行うJavaの実装を、ファイル名SHA256withRSA.javaとして用意しました。

```
import java.math.BigInteger;
import java.util.Base64;
import java.security.KeyFactory;
import java.security.Signature;
import java.security.PrivateKey;
import java.security.SignatureException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.RSAPrivateCrtKeySpec;

import sun.security.util.DerInputStream;
import sun.security.util.DerValue;

public class SHA256withRSA {
 /**
  * RSASSA-PKCS1-v1_5 with SHA-256によるデジタル署名の生成
  *
  * @param header    BASE64エンコード済みのJWTヘッダー
  * @param payload   BASE64エンコード済みのJWTペイロード
  * @param key       PKCS#1形式でのRSA秘密鍵(BEGIN/END行なし、改行なし)
  * @return デジタル署名 - BASE64エンコード済み
  */
 public static String sign(String header, String payload, String key)
 throws Exception
 {
  /* header and payload are both encoded in base64 */
  byte[] data = new String(header + "." + payload).getBytes("UTF-8");

  /* get PrivateKey instance from PKCS#1 */
  byte[] pkdata = Base64.getDecoder().decode(key);
  DerInputStream derReader = new DerInputStream(pkdata);
  DerValue[] seq = derReader.getSequence(0);
  // skip version seq[0];
  BigInteger modulus = seq[1].getBigInteger();
  BigInteger publicExp = seq[2].getBigInteger();
  BigInteger privateExp = seq[3].getBigInteger();
  BigInteger prime1 = seq[4].getBigInteger();
  BigInteger prime2 = seq[5].getBigInteger();
  BigInteger exp1 = seq[6].getBigInteger();
  BigInteger exp2 = seq[7].getBigInteger();
  BigInteger crtCoef = seq[8].getBigInteger();

  RSAPrivateCrtKeySpec keySpec =
   new RSAPrivateCrtKeySpec(modulus, publicExp, privateExp, prime1, prime2, exp1, exp2, crtCoef);
  KeyFactory keyFactory = KeyFactory.getInstance("RSA");
  PrivateKey privateKey = keyFactory.generatePrivate(keySpec);

  // creating the object of Signature
  Signature sr = Signature.getInstance("SHA256withRSA");
  sr.initSign(privateKey);
```

```
    sr.update(data);
    byte[] bytes = sr.sign();

    /* return signature in Base64 */
    return Base64.getEncoder().encodeToString(bytes);
  }
}
```

これをデータベースにロードします。ロードするスキーマはMYWORKSPACEとします。

```
loadjava -force -verbose -resolve -u myworkspace/**********@localhost/service.world SHA256withRSA.java
```

-verboseをオプションがついているので、以下のような出力が行われます。最後のErrorsが０でloadjavaの実行が終了したら、このJavaコードのデータベースへのロードが完了です。

```
arguments: '-u' 'myworkspace/***@localhost/service.world' '-force' '-verbose' '-resolve' 'SHA256withRSA.java' creating : source SHA256withRSA
loading  : source SHA256withRSA
resolving: source SHA256withRSA
Classes Loaded: 0
Resources Loaded: 0
Sources Loaded: 1
Published Interfaces: 0
Classes generated: 0
Classes skipped: 0
Synonyms Created: 0
Errors: 0
```

## Javaの実装をラップするPL/SQLファンクションを定義する

PL/SQLからファンクションとして呼び出せるよう、Wrapperとなるファンクションを設定します。

```
create or replace function sha256withrsa_sign
(
  header      varchar2,
  payload     varchar2,
  private_key varchar2
) return varchar2
as
language java name 'SHA256withRSA.sign (java.lang.String, java.lang.String, java.lang.String) return java.lang.String';
/
```

これで、SHA256WITHRSA_SIGNとしてPL/SQLのコード中からRSAによるデジタル署名を実行できるようになりました。

**sun.security.util**というパッケージに含まれるクラスは、Oracle Databaseのデフォルトでは、一般のスキーマからのアクセスに制限が設けられている模様です。ですので、sysやsystemといったDBA権限のあるユーザーにて、これらの実行をMYWORKSPACEにたいして許可する必要がありました。そのために使用したコマンドが以下になります。

```
dbms_java.grant_permission( 'MYWORKSPACE', 'SYS:java.lang.RuntimePermission', 'accessClassInPackage.sun.security.util', '' );
```

## JWTをPL/SQLコードで生成する

実装したデジタル署名アルゴリズムを指定したうえで、PL/SQLのコードを使ってJWTを生成します。このコード自体は、こちらの記事とほぼ同じものになります。

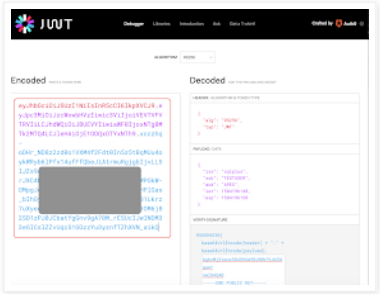
```
set lines 1000
set serveroutput on
declare
  l_now      timestamp;
  l_secret    varchar2(32767) := 'MIIEpAIBAAKCAQEA+ARMJiUjN+3kWFckXnxQkbHcbnKxoB46cHyuI+p7f7itiE4x8gJ6A9ML1alo6uCnHn8D0vaDJ/DVzL9whTS8zXJTB/WzGs35
  l_username  varchar2(32) := 'TESTUSER';
  l_jwt       varchar2(32767);
  l_jwt_token apex_jwt.t_token;
  l_jwt_t     apex_t_varchar2;

  l_header_json json_object_t;
  l_header_str  varchar2(200);
  l_header_base64 varchar2(400);
  l_payload_json json_object_t;
  l_payload_str  varchar2(200);
  l_payload_base64 varchar2(800);
  l_token varchar2(1000);
  l_hmac varchar2(1000);

  -- Unix時間の取得
  function unixtime(p_timestamp in timestamp)
  return pls_integer
  is
    l_date date;
    l_epoc number;
  begin
    l_date := sys_extract_utc(p_timestamp);
    l_epoc := l_date - date'1970-01-01';
    return l_epoc * 24 * 60 * 60;
  end unixtime;

  -- Base64のデコード
  function from_base64(t in varchar2) return varchar2 is
  begin
    return utl_raw.cast_to_varchar2(utl_encode.base64_decode(utl_raw.cast_to_raw(t)));
```

```
  end from_base64;

  -- Base64のエンコード
  function to_base64(t in varchar2) return varchar2 is
    l_base64 varchar2(32767);
  begin
    l_base64 := utl_raw.cast_to_varchar2(utl_encode.base64_encode(utl_raw.cast_to_raw(t)));
    l_base64 := replace(l_base64, chr(13)||chr(10), '');
    return l_base64;
  end to_base64;
begin
  -- 共通で使用する現在時刻
  l_now := current_timestamp;
  dbms_output.put_line('Current Timestamp = ' || l_now || ', unixtime = ' || unixtime(l_now));

  -- ヘッダーを手作業で生成する。
  dbms_output.put_line('Hand made ===================================');
  l_header_json := json_object_t();
  l_header_json.put('alg','RS256');
  l_header_json.put('typ','JWT');
  l_header_str := l_header_json.to_string();
  l_header_base64 := to_base64(l_header_str);
  dbms_output.put_line('Header    = ' || l_header_str);

  -- ペイロードを手作業で作成する。
  l_payload_json := json_object_t();
  l_payload_json.put('iss','sqlplus');
  l_payload_json.put('sub',l_username);
  l_payload_json.put('aud','APEX');
  l_payload_json.put('iat',unixtime(l_now));
  l_payload_json.put('exp',unixtime(l_now)+10);
  l_payload_str := l_payload_json.to_string();
  l_payload_base64 := to_base64(l_payload_str);
  dbms_output.put_line('Payload   = ' || l_payload_str);

  -- シグネチャを手作業で作成する。
  l_hmac := SHA256withRSA_sign(l_header_base64, l_payload_base64, l_secret);
  l_hmac := trim(translate(l_hmac, '+/=', '-_ '));
  dbms_output.put_line('JWT = ' || l_header_base64 || '.' || l_payload_base64 || '.' || l_hmac);
end;
/

exit;
```

上記のコードを実行すると以下のような出力がされます。

```
Current Timestamp = 17-MAR-20 03.48.50.702398 PM, unixtime = 1584427730
Hand made ===================================
Header   = {"alg":"RS256","typ":"JWT"}
Payload   = {"iss":"sqlplus","sub":"TESTUSER","aud":"APEX","iat":1584427730,"exp":1584427740}
JWT = eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzcWxwbHVzIiwic3ViIjoiVEVTVFVTRVIiLCJhdWQiOiJBUEVYIiwiaWF0IjoxNTg0NDI3NzMwLCJleHAiOjE1ODQ0Mjc3NDE
```

https://jwt.ioにはJWTのデバッガーがありますので、生成されたデータと鍵を入力し、JWTのデジタル署名の検証が可能です。




RSASSA-PKCS1-v1_5 with SHA-256によるデジタル署名の生成についてはこちらの記事、PKCS#1の鍵フォーマットから、秘密鍵を取り出すためにこちらの記事を参照させていただきました。

**完**

Yuji N. 時刻: 23:43

共有

‹ ホーム ›

ウェブ バージョンを表示

**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