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17日火曜日

RS256で作成したJWTを検証する

以前に書いたこちらの記事の継続です。

JWTは作成できたので、今度は検証する方法を実装してみます。以前の記事のJavaコードには署名(sign)しか実装がありませんので、検証(verify)を追加しました。

```
import java.math.BigInteger;
import java.util.Base64;
import java.security.KeyFactory;
import java.security.Signature;
import java.security.PublicKey;
import java.security.PrivateKey;
import java.security.SignatureException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.RSAPrivateCrtKeySpec;
import java.security.spec.X509EncodedKeySpec;

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 then sign */
    Signature sr = Signature.getInstance("SHA256withRSA");
    sr.initSign(privateKey);
    sr.update(data);
    byte[] bytes = sr.sign();

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

  /**
   * RSASSA-PKCS1-v1_5 with SHA-256によるデジタル署名の検証
   *
   * @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 sig       BASE64エンコードされた電子署名
   * @param key       RSA公開鍵(BEGIN/END行なし、改行なし)
   * @return 検証結果
   */
  public static boolean verify(String header, String payload, String sig, String key)
  throws Exception
  {
    /* header and payload are both encoded in base64 */
    byte[] data = new String(header + "." + payload).getBytes("UTF-8");
    /* get signature in binary representation. */
    byte[] dsig = Base64.getDecoder().decode(sig);
```

```
    /* get PublicKey instance from X.509 */
    byte[] pkdata = Base64.getDecoder().decode(key);
    X509EncodedKeySpec keySpec = new X509EncodedKeySpec(pkdata);
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    PublicKey publicKey = keyFactory.generatePublic(keySpec);

    /* creating the object of Signature then verify */
    Signature sr = Signature.getInstance("SHA256withRSA");
    sr.initVerify(publicKey);
    sr.update(data);
    return sr.verify(dsig);
  }
}
```

データベースにロードするのは、前回とまったく同じloadjavaコマンドを使用します。

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

追加したメソッドのラッパーとなるファンクションを定義します。

```
create or replace function sha256withrsa_verify
(
  header     varchar2,
  payload    varchar2,
  signature  varchar2,
  public_key  varchar2
) return boolean
as
language java name 'SHA256withRSA.verify (java.lang.String, java.lang.String, java.lang.String, java.lang.String) return boolean';
/
```

RSA公開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。まずはキー・ペアを生成し、

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

それから公開鍵を取り出します。

```
openssl rsa -pubout -in ~/.oci/poa_oci_api_key.pem -out ~/.oci/poa_oci_api_key_public.pem
```

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_public.pem
echo
```

これで、データベース側の準備はできたので、JWTの検証処理を行ってみます。検証には以下のPL/SQLコードを使用しました。

```
set lines 1000
set serveroutput on
declare
  l_public    varchar2(32767) := 'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMII***一部省略***dgaetswIDAQAB';
  l_jwt       varchar2(32767) := 'eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzcWxwbHVzIiwic3ViIjoiVEVTVFVTRVIiLCJhdWQiOiJBUEVYIiwiaWF0***一部省略***hLzvCH
  l_username  varchar2(32);
  l_header_json json_object_t;
  l_header_len  number;
  l_header_base64 varchar2(400);
  l_payload_len number;
  l_payload_json json_object_t;
  l_payload_base64 varchar2(800);
  l_hmac varchar2(1000);

  -- 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
  /*トークンの内容を印刷 */
  l_header_len     := instr(l_jwt, '.', 1);
  l_header_base64  := substr(l_jwt, 1, l_header_len - 1);
  l_header_json    := json_object_t.parse(from_base64(l_header_base64));
  dbms_output.put_line('Header = ' || l_header_json.to_string);
  l_jwt := substr(l_jwt, l_header_len + 1);
  l_payload_len    := instr(l_jwt, '.', 1);
  l_payload_base64 := substr(l_jwt, 1, l_payload_len - 1);
  l_payload_json   := json_object_t.parse(from_base64(l_payload_base64));
  dbms_output.put_line('Payload = ' || l_payload_json.to_string);
  l_username := l_payload_json.get_string('sub');
  dbms_output.put_line('Username = ' || l_username);
  l_hmac := substr(l_jwt, l_payload_len + 1);
  dbms_output.put_line('Signature = ' || l_hmac);
```

```
  /* シグネチャの検証をする。*/
  l_hmac := trim(translate(l_hmac, '-_ ', '+/='));
  if SHA256withRSA_verify(l_header_base64, l_payload_base64, l_hmac, l_public) then
    dbms_output.put_line('Signature verified successfully.');
  else
    dbms_output.put_line('Signature verify failed.');
  end if;
end;
/
exit;
```

l_publicには、RSA公開鍵を一行で指定します。l_jwtにはJWTをこれも一行で指定します。実行した結果は以下のようになります。

```
Header = {"alg":"RS256","typ":"JWT"}
Payload = {"iss":"sqlplus","sub":"TESTUSER","aud":"APEX","iat":1584430294,"exp":1584430304}
Username = TESTUSER
Signature = AJE1R2m_-8OTZLocdNaOfa5UF3EirLlsYkpD***一部省略***t6mSYFPshK3km_X9jNXQfHHP9As-2HnnrOSFMN9A
Signature verified successfully.
```

これで、Oracle APEXにて(正確にはOracle Databaseにて)RS256を使ったJWTの作成と検証の両方ができるようになっています。

完

Yuji N. 時刻: 15:57

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