Message Signature

JSON Web Signature

Overview

In accordance with the evolution of the API, PayNet has implemented a new authentication mechanism utilizing JWS.

JWS stands for "JSON Web Signature". It is a compact, URL-safe means of representing digitally signed JSON (JavaScript Object Notation) data. It provides a way to ensure that a message has not been tampered with, and that it comes from a trusted source.

A JWS has three main components:

- **Header:** It contains metadata about the JWS such as the algorithm used for signing and the type of token. The header is a JSON object that is Base64Url encoded.

- **Payload:** This contains the actual data that is being transmitted in the JWS. It can be any valid JSON object, and it too is Base64Url encoded.

- **Signature:** This is the result of applying the cryptographic signature algorithm to the encoded header and payload, using a secret key. The signature is also Base64Url encoded.

These three components are then concatenated with periods (.) to form the complete JWS. The resulting string can be transmitted over the wire and verified by the recipient to ensure its authenticity and integrity.

INFO

We are in the midst of updating all our APIs to support JWS. Please refer to the respective API product reference to find out which authentication method is being supported currently.

Generating JWS (API Request)

Here are the steps to generate a JWS token for an API request.

1. Generate the payload, minify and hashing using SHA-256

Provided that you have the following payload to transmit, it is imperative that you apply SHA-256 hashing algorithm.

Sample payload :

```
{
  "data": {
    "businessMessageId": "20230412BOEEMYK1000ORB00000001",
    "clientMessage": "Client hello"
  }
}
```

Sample payload after minification :

```
{"data":
{"businessMessageId":"20230412BOEEMYK1000ORB00000001","clientMessage":"Client
hello"}}
```

Sample generated hash SHA-256 :

```
8fc1f5ed05596aa2952e68ac221f31ee8a87641315c7b091f0bd41266d380739
```

GET special handling

INFO

For GET methods where no JSON request payload exists, users are expected to construct the generic body for JWS in the format shown below

```
{"data":{"businessMessageId":"<business_message_id_of_get_request>"}}
```

2. Generate JWS Header

Please find below a sample JWS header that the client is required to construct in a similar fashion.

```
{
  "alg": "RS512",
  "typ": "JWT",
  "kid": "<Cert Serial Number>"
}
```

Field	Definition
alg	The algorithm for generating the signature in this context supports RS512
typ	The type of content for the JWS payload in this context is set as default to JWT
kid	The Key ID (kid) in this context is a reference to the specific public key used to verify the JWS signature. Please put certificate serial number used to verify the signature

3. Generate JWS body

Please find below a sample JWS body that the client is required to construct in a similar fashion.

```
{
   "iss": "BOEEMYK1",
   "exp": 1681385787,
   "jti": "20230412BOEEMYK1000ORB00000001",
   "ds": "<Hash SHA-256 Value of Payload>"
}
```

Field	Definition
iss	The "iss" (issuer) claim identifies the principal that issued the JWT. Default to client BIC code.
exp	The "exp" (expiration time) claim identifies the expiration time on or after which the JWT MUST NOT be accepted for processing. The processing of the "exp" claim

Field	Definition
	requires that the current date/time MUST be before the expiration date/time listed in the "exp" claim. Default to 15 mins in epoch time
jti	Default to businessMessageId
ds	sha-256 value of request payload

Example :

```
{
    "iss": "BOEEMYK1",
    "exp": 1681385787,
    "jti": "20230412BOEEMYK1000ORB00000001",
    "ds": "8fc1f5ed05596aa2952e68ac221f31ee8a87641315c7b091f0bd41266d380739"
}
```

4. Generate signature based on JWS constructed using private key

```
WAjEQpSsafFfT9WsH4QRTVrvEeU3wD6Ou67WzCF1d7UN2AIJvAASSb0zsAhdRxmvSbdus6[illegible]ZB[illegible]V
2txGaO7lcN_ZgtBpus9p2QMO73ZgT3Bhg0t2w80L1NKhG7WHFWxy3h1q9aMAHH8AQLzXtVsHz_xWAw3L
3EVil04UirN-kpQbXKfx35CN9mgsuAuydDCzj6HgEMn3k-mNLhof9ZZaDSziLimETXPN0Y6hnN5h0-54
```

5. Generate JWS full token

`base64UrlEncode(jws_header) + "." + base64UrlEncode(jws_payload) + "." + base64UrlEncode(jws_signature)`

Example :

```
eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6IlJQUF9QVksyMDIxIn0.eyJpc3MiOi
2txGaO7lcN_ZgtBpus9p2QMO73ZgT3Bhg0t2w80L1NKhG7WHFWxy3h1q9aMAHH8AQLzXtVsHz_xWAw3L
```

6. Place JWS token into Authorization header during request

```
curl --location -g --request PUT 'https://api_domain' \
--header 'Authorization: Bearer
eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6IlJQUF9QVksyMDIxIn0.eyJpc3MiOiJCT0VFTVlLMSIsIm
2txGaO7lcN_ZgtBpus9p2QMO73ZgT3Bhg0t2w80L1NKhG7WHFWxy3h1q9aMAHH8AQLzXtVsHz_xWAw3L
--data-raw '{
    <Sample Body>
}'
```

Verifying JWS (API Response)

Here are the steps to verify a JWS token for an API response.

1. Fetch JWS token from Authorization header

```
curl --location -g --request PUT 'https://api_domain' \
--header 'Authorization: Bearer
eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6IlJQUF9QVksyMDIxIn0.eyJpc3MiOiJCT0VFTVlLMSIsIm
2txGaO7lcN_ZgtBpus9p2QMO73ZgT3Bhg0t2w80L1NKhG7WHFWxy3h1q9aMAHH8AQLzXtVsHz_xWAw3L
--data-raw '{
    <Sample Body>
}'
```

2. Decode JWS header, and fetch `alg` and `kid`

```
{
    "alg": "RS512",
    "typ": "JWT",
    "kid": "<Cert Serial Number>"
}
```

Extract the JWS header from the JWS token, which is the first part of the token separated by a period (".").The decoded JWS header will be a JSON object that contains information about the JWS token, including the algorithm used to generate the signature and the key identifier (kid) that represents the certificate serial number to identify which certificate needs to be used to verify the response.

3. Decode JWS payload, and verify the signature

Validate the JWS signature against the JWS payload by using the public key with the encryption algorithm, as specified in alg element.

4. Verify the JWS payload against the actual API payload

Sample payload :

```
{
  "data": {
    "businessMessageId": "20230412BOEEMYK10000RB00000001",
    "clientMessage": "Client hello"
  }
}
```

Sample payload after minification :

```
{"data":
{"businessMessageId":"20230412BOEEMYK1000ORB00000001","clientMessage":"Client
hello"}}
```

Sample generated hash SHA-256 :

```
8fc1f5ed05596aa2952e68ac221f31ee8a87641315c7b091f0bd41266d380739
```

The client needs to compare the generated SHA-256 hash with those inside the JWS payload 'ds' tag.

Sample code

Dependencies and Libraries

INFO
Please ensure that your project includes the required libraries mentioned below before running the provided sample codes.

Java Python PHP GO

```
...
<dependencies>
    <dependency>
```

```xml
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-api</artifactId>
        <version>0.11.5</version>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-impl</artifactId>
        <version>0.11.5</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-jackson</artifactId>
        <version>0.11.5</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.15.0</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.15.0</version>
    </dependency>
    <dependency>
        <groupId>org.bouncycastle</groupId>
        <artifactId>bcprov-jdk15on</artifactId>
        <version>1.70</version>
    </dependency>
```

```xml
        <dependency>
            <groupId>org.bouncycastle</groupId>
            <artifactId>bcpkix-jdk15on</artifactId>
            <version>1.70</version>
        </dependency>
    </dependencies>
    ...
```

Business Message Id generation

Java

```java
// Import class
import my.paynet.sdk.duitnow.utils.DuitNowUtils;
import my.paynet.sdk.duitnow.constant.ChannelCode;
import my.paynet.sdk.duitnow.constant.OriginatorCode;
import my.paynet.sdk.duitnow.constant.TransactionCode;

public class Transaction {

    public void generateBusinessMessageId() {
      TransactionCode transactionCode =
TransactionCode.TRANSACTION_STATUS_INQUIRY;
      OriginatorCode originator = OriginatorCode.RETAIL_PAYMENTS_PLATFORM;
      ChannelCode channel = ChannelCode.RETAIL_INTERNET_BANKING;
      // call sdk method to generate
      String businessMessageId = DuitNowUtils.generateBusinessMessageId(bic,
transactionCode, originator, channel);
      ...
    }
}
```

JWS generation

Java Python PHP GO

```
// Import class
import my.paynet.sdk.duitnow.utils.DuitNowUtils;
import my.paynet.sdk.duitnow.constant.ChannelCode;
import my.paynet.sdk.duitnow.constant.OriginatorCode;
import my.paynet.sdk.duitnow.constant.TransactionCode;

public class Transaction {

    public void generateJWS() {
        TransactionCode transactionCode =
TransactionCode.TRANSACTION_STATUS_INQUIRY;
        OriginatorCode originator = OriginatorCode.RETAIL_PAYMENTS_PLATFORM;
        ChannelCode channel = ChannelCode.RETAIL_INTERNET_BANKING;
        // call sdk method to generate
        String businessMessageId =
DuitNowUtils.generateBusinessMessageId(bic, transactionCode, originator,
channel);
        ...
        String certSerialNumber = "12345"; // Replace with your certificate's
serial number
        Object payload = null; // Replace with your actual payload object
        PaynetSigner signer = new DuitNowV2Signer();
        String jws = signer.generateJws(privateKey, businessMessageId,
certSerialNumber, payload);

        ...
    }
}
```

JWS verification

Java Python PHP GO

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.PublicKey;
import java.security.Security;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.util.Base64;
```

```java
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.stream.Collectors;

import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.util.encoders.Hex;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

/**
 * @author Umair
 *
 */

public class JWSGenerator {

  private static final String KEY_ALGORITHM = "RSA";
  private static final String CLASSPATH_COLON = "classpath:";
  private static final String JWT = "JWT";

  private PrivateKey privateKey;
  private PublicKey publicKey;
  private String certSerialNumber;
  private static final String ALG = "alg";
  private static final String TYP = "typ";
  private static final String KID = "kid";
  private static final String ISS = "iss";
  private static final String EXP = "exp";
  private static final String JTI = "jti";
  private static final String DS = "ds";

  public static void main(String[] args) {

    // replace this with customer specific value
    String certificatePath = "<path to certificate file>";
    String jsonRequestPayload = "<sample JSON>";
    String jwsToken = "<jws token from response>";

    X509Certificate certificate = extractX509Certificate(certificatePath);
    PublicKey publicKey = certificate.getPublicKey();

    System.out.println(validate(jwsToken, jsonRequestPayload));
  }

  // load certificate for JWS verification
  private static X509Certificate extractX509Certificate(String path) throws
IOException, CertificateException {
    if (path == null || path.isBlank()) {
      return null;
    }
    InputStream inputStream = null;
    try {
      if (path.startsWith(CLASSPATH_COLON)) {
        String fileName = path.replace(CLASSPATH_COLON, "");
      inputStream = getClass().getClassLoader().getResourceAsStream(fileName);
    } else {
      inputStream = new FileInputStream(path);
    }
    CertificateFactory factory = CertificateFactory.getInstance(X_509);
    return (X509Certificate) factory.generateCertificate(inputStream);
  } finally {
    if (inputStream != null) {
      inputStream.close();
    }
  }
}

// minify JSON payload and hashed using SHA-256
private static String minifyDs(String payload) {
  try {
    ObjectMapper objectMapper = new ObjectMapper();
    String minify = objectMapper.readValue(payload, JsonNode.class).toString()
    System.out.printf("Minify Ds value...%s\n", minify);
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hash = digest.digest(minify.getBytes(StandardCharsets.UTF_8));
    return new String(Hex.encode(hash));
  } catch (JsonProcessingException | NoSuchAlgorithmException e) {
    e.printStackTrace();
    return null;
  }
}


// Verify JWS token
public static boolean validate(String token, String payload) {
    try {
      if (token == null || token.isEmpty()) {
        return false;
      }

      if (token.startsWith("Bearer")) {
        token = token.replace("Bearer ", "");
      }
      String minifyPayload = minifyDs(payload);

      Claims claims =
Jwts.parserBuilder().setSigningKey(publicKey).build().parseClaimsJws(token).getB

      return minifyPayload.equals(claims.get(DS));
    } catch (Exception e) {
      e.printStackTrace();
      return false;
    }
  }

}
```

How to Generate Key Pair

Using OpenSSL

Step 1: Generate private key and CSR (Certificate Signing Request)

```
openssl req \
       -newkey rsa:2048 -nodes -keyout example.key \
       -out example.csr
```

Step 2: Generate self-signed certificate from generated CSR

INFO

For production usage, this certificate must be created by valid Certificate Authority (CA). Self-signed certificate only valid for sandbox usage.

```
openssl x509 \
       -signkey example.key \
       -in example.csr \
       -req -days 365 -out example.cer
```

Sample Codes

Message Signing

Java Python PHP C#

```
// Import class
import my.paynet.sdk.duitnow.utils.DuitNowUtils;
import my.paynet.sdk.duitnow.constant.ChannelCode;
import my.paynet.sdk.duitnow.constant.OriginatorCode;
import my.paynet.sdk.duitnow.constant.TransactionCode;

public class Transaction {

    public void signMessage() {
        ...
        TransactionCode transactionCode =
TransactionCode.TRANSACTION_STATUS_INQUIRY;
        OriginatorCode originator = OriginatorCode.RETAIL_PAYMENTS_PLATFORM;
        ChannelCode channel = ChannelCode.RETAIL_INTERNET_BANKING;

        // Replace your private key here
        PrivateKey privateKey = getPrivateKey(privateKeyString);

        String businessMessageId =
DuitNowUtils.generateBusinessMessageId(bic, transactionCode, originator,
channel);
        String endToEndId = "20231220PICAMYK1010ORB29894000";
        // Replace message to sign
        String message = DuitNowUtils.getMessageToSign(transactionCode,
businessMessageId, endToEndId);

        PaynetSigner signer = new DuitNowV2Signer();
        // SDK generate signature
        String signature = signer.generateSignature(privateKey, message);
        ...
    }
}
```

Message Verification

Java PHP C#

```java
// Import class
import my.paynet.sdk.duitnow.utils.DuitNowUtils;
import my.paynet.sdk.duitnow.constant.TransactionCode;

public class Transaction {

    public void messageVerification() {
        // Replace your private key here
        PrivateKey privateKey = getPrivateKey(privateKeyString);
        String endToEndId = "20231220PICAMYK1010ORB29894000";
        PaynetSigner signer = new DuitNowV2Signer();

        // Replace your certificate here
        X509Certificate certificate = ...
        String signature =  signer.generateSignBodyAndSignature(certificate,
TransactionCode.TRANSACTION_STATUS_INQUIRY, businessMessageId, endToEndId);
        // Message from response
        String message = ...

        boolean validate =
signer.verify(getPublicCert(publicCertString),message,signature);
        ...
    }
}
```