

Security & Encryption

Signature Generation

INFO

You can also find our message signature SDK or sample apps in **resources** section.

PayNet TSP API uses a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) for authentication. To authenticate a request, serialize the json data of the request body to form a string and append with timestamp. Then use the API secret key to calculate the HMAC of that string. Finally, add this signature as a parameter of the request by using the syntax described in this section.

```
Authorization: version:apiKey:timestamp:signature
```



Pseudo Code

Following is pseudo grammar that illustrates the construction of the Authorization request header.

```
version = "v1";

timestamp = Unix Epoch timestamp in milliseconds;

stringToSign = serialize( request body json data ) + timestamp; // e.g.
{"field\":"value\","...}1595476169859
```




```
signature = Base64( HMAC-SHA256( secretKey, UTF-8-Encoding-Of( stringToSign )
) );

authorization = version + ":" + apiKey + ":" + timestamp + ":" + signature;
```

Sample Code

Java



```
public class TSPSignature {

    private static final String CIPHER = "AES/CBC/PKCS5Padding";
    private static final String AES = "AES";
    private static final String HMAC = "HmacSHA256";

    public static String genSignature(String key, String stringToSign) throws
Exception {
        Mac sha256_HMAC = Mac.getInstance(HMAC);
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes("UTF-8"),
HMAC);
        sha256_HMAC.init(secret_key);
        return
Base64.encodeBase64String(sha256_HMAC.doFinal(stringToSign.getBytes("UTF-
8"))));
    }
}
```

```
}
```

Data Encryption

As a general practice, PCI DSS and other security guidelines must be applied when handling PAN data. Some fields that contain sensitive data in PayNet TSP API are encrypted. Those fields are prefixed with 'enc' such as enc_pan_data and enc_token_data in the API request or response.

The serialized json data will be encrypted / decrypted using AES-256 (AES/GCM/NoPadding) f with a merchant specific secret key and a random IV (Initialization vector). Hence only the merchant who possesses the api key and secret key can access the sensitive data.

INFO

V2 are introduced to address the issue with a weak cipher algorithm that occurred in V1.

The random IV is prepended to the encrypted data like so:

Data Encryption Version v2


```
byte[] IV = new byte[12];  
new SecureRandom().nextBytes(IV);
```



Pseudo Code

Following is pseudo grammar for data encryption:

Version v2



```
// Create Cipher Instance and GCM SecretKeySpec
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
cipher.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(secretKey.getBytes(),
"AES"), new GCMParameterSpec(128, IV));


// Perform Encryption
byte[] cipherText = cipher.doFinal(pText.getBytes(StandardCharsets.UTF_8));

// Prepend Cipher Text with IV
byte[] cipherTextWithIv = ByteBuffer.allocate(IV.length +
cipherText.length).put(IV).put(cipherText).array();

// Encode to Base64
java.util.Base64.getEncoder().encodeToString(cipherTextWithIv);
```

Following is pseudo grammar for data decryption:

Version v2



```
// Extract IV bytes and cipher bytes from encryptedText
// Extract and decode bytes from base 64 encryptedText
ByteBuffer bb =
ByteBuffer.wrap(java.util.Base64.getDecoder().decode(cText.getBytes(UTF_8)));

// Extract IV
byte[] iv = new byte[12];
bb.get(iv);
```

```
// Extract Cipher Text
byte[] cipherText = new byte[bb.remaining()];
bb.get(cipherText);

// Get Cipher Instance
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");

// Initialize Cipher for DECRYPT_MODE
cipher.init(Cipher.DECRYPT_MODE , new SecretKeySpec(password.getBytes(),
"AES") , new GCMParameterSpec(128, iv));

// Convert to UTF8 string
new String(cipher.doFinal(cipherText), UTF_8);
```

Sample Code Version 2

Java

```
public static String encrypt(String secretKey, String stringToEncrypt)
    throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
    InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException,
    IOException {
    // Create SecretKeySpec
    SecretKeySpec secretKeySpec = new SecretKeySpec(secretKey.getBytes(), "AES");
    // Create IvSpec
    byte[] ivSpec = new byte[12];
    new SecureRandom().nextBytes(ivSpec);
```

```

        // Create Cipher Instance and Initialize Cipher for ENCRYPT_MODE
        Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, new GCMParameterSpec(128, ivSpec));
        // Perform Encryption
        byte[] cipherbytes = cipher.doFinal(stringToEncrypt.getBytes(StandardCharsets.UTF_8));
        // Prepend IV bytes to cipher bytes
        byte[] cipherTextWithIv = ByteBuffer.allocate(ivSpec.length + cipherbytes.length)
            .put(ivSpec)
            .put(cipherbytes)
            .array();
        // Encode to Base64
        return java.util.Base64.getEncoder().encodeToString(cipherTextWithIv);
    }

    public static String decrypt(String secretKey, String encryptedText)
        throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
        InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException,
        IOException {
        try {
            // Create SecretKeySpec
            SecretKeySpec keySpec = new SecretKeySpec(secretKey.getBytes(), AES);
            // Extract IV bytes and cipher bytes from encryptedText
            ByteBuffer bb =
                ByteBuffer.wrap(java.util.Base64.getDecoder().decode(encryptedText.getBytes(StandardCharsets.UTF_8)));
            byte[] iv = new byte[12];
            bb.get(iv);

            byte[] cipherBytes = new byte[bb.remaining()];
            bb.get(cipherBytes);

```

```
        // Get Cipher Instance and Initialize Cipher for DECRYPT_MODE
        Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
        cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, new GCMParameterSpec(128, iv));

        // Convert to UTF8 string
        return new String(cipher.doFinal(cipherBytes), StandardCharsets.UTF_8);
    } catch (Exception ex) {
        logger.error(ex.getMessage());
    }

    return null;
}
```