SPONSORED MERCHANT API WITH AMERICAN EXPRESS

Java Code Example

The following document shows the process of consuming the Sponsored Merchant API (SMAPI) from American Express using Java as the language for programming.

Prerequisites to connecting:

- Access granted to the Amex for Developer's Portal by an Amex Enablement Manager
- Received and installed the Amex Server TLS certificates on Client Server's Truststore
- Shared valid Client Certificate with Amex for QA Environment
- Amex has installed the Client Certificate and provided back the Client ID and Client Secret keys.

Client Certificate Requirements

- Certificate must be signed by a valid Certificate Authority
- Certificate cannot be Self-Signed
- The validity must be 1 or 2 years, and must still have at least 9 months before expiry
- The Certificate file must contain the full chain (Root, Intermediate, Leaf)

Step 1 :: Create Authorization Header

Create a hash of the request payload

- We utilized base64, Mac, and SecretKeySpec
    - https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html
    - https://docs.oracle.com/javase/8/docs/api/javax/crypto/Mac.html
    - https://docs.oracle.com/javase/8/docs/api/javax/crypto/spec/SecretKeySpec.html
- The Client Secret key is used in generation of the BodyHash and Mac
- Note: Amex accepts only the Sha256 algorithm for both BodyHash and Mac

```
// create bodyhash by hashing payload
SecretKeySpec signingKey = new SecretKeySpec(clientSecret.getBytes(),
HMAC_SHA256_ALGORITHM);
Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
mac.init(signingKey);
byte[] bodyBytes = mac.doFinal(payload.getBytes());
String bodyhashString = Base64.getEncoder().encodeToString(bodyBytes);
```

Compile the components that make up the mac signature

- THE ORDER OF COMPONENTS IS CRITICAL
- TIMESTAMP, NONCE, REQUEST METHOD, RESOURCE PATH, HOST, PORT, BODYHASH
- THE COMPONENTS ARE COMBINED INTO A STRING WHICH IS DELIMITED BY A NEWLINE CHARACTER \N.

```
// The order of the MAC components is critical
// Timestamp + \n + nonce + \n + httpMethod + \n + path + \n + host + \n + port + \n +
hash + \n

String macInput = ts + "\n" + nonce + "\n" + httpMethod + "\n" + resourcePath + "\n" +
host
        + "\n" + port + "\n" + bodyhashString + "\n";
```

Create a hash of the mac components to get the mac signature

- CREATED USING THE SAME METHOD AS BODYHASH ABOVE

```
// create mac signature by hashing baseString
byte[] macBytes = mac.doFinal(macInput.getBytes());
String macString = Base64.getEncoder().encodeToString(macBytes);
```

Compile the Authorization Header

The order of components in the Authorization Header is critical

- MAC ID = YOUR APPLICATION'S CLIENT ID. YOU CAN FIND THE CLIENT ID VALUE IN THE DASHBOARD AFTER LOGGING IN.
- TS = TIMESTAMP WHICH YOU GENERATE. THE FORMAT IS UNIX EPOCH TIME (MS).
- NONCE = UNIQUE IDENTIFIER STRING. THE VALUE OF NONCE MUST BE UNIQUE FOR EACH REQUEST
- BODYHASH = HASH OF THE MESSAGE BODY USING THE HMAC SHA256 ALGORITHM
- MAC = THE MAC IS GENERATED USING THE HMAC SHA256 ALGORITHM.

```
// build authorization header string
String authorizationHeader = "MAC id=\"" + clientId + "\",ts=\"" + ts + "\",nonce=\""
+ nonce +
        "\",bodyhash=\"" + bodyhashString + "\",mac=\"" + macString + "\"";
```

Step 2 :: Setup Additional API Request Components

Step 1: Create the HTTP Headers

- The headers for the SMAPI include:
  - Content-Type: application/json
  - X-AMEX-API-KEY: client ID
  - Authorization: Authorization Header

```
//create the headers object after hmac is generated
HttpHeaders headers = new HttpHeaders();
headers.add(HttpHeaders.CONTENT_TYPE, "application/json");
headers.add("X-AMEX-API-KEY", clientId);
headers.add("Authorization", authHmac);
```

Step 2: Create the remaining HTTP components

- Create the SSL Context
  - https://docs.oracle.com/en/java/javase/11/docs/api/java.base/javax/net/ssl/SSLContext.html
  - https://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/conn/ssl/SSLContexts.html

```
// Create SSL Context
final KeyStore keyStore = KeyStore.getInstance("PKCS12");
final String keyPassPhrase = "PRIVATE KEY PASSPHRASE";
String pathToCertificate = "PATH TO CERTIFICATE";
keyStore.load(new FileInputStream(pathToCertificate), keyPassPhrase.toCharArray());
SSLContext sslContext = SSLContexts.custom().loadKeyMaterial(keyStore,
keyPassPhrase.toCharArray()).build();
```

Creating the Connection Manager

- We utilized PoolingConnectionManager and SSL ConnectionSocketFactory
  - https://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/impl/conn/PoolingClientConnectionManager.html
  - https://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/conn/ssl/SSLConnectionSocketFactory.html

```
// Create connection manager
final Registry<ConnectionSocketFactory> socketFactoryRegistry = RegistryBuilder
        .<ConnectionSocketFactory>create().register("https", new
SSLConnectionSocketFactory(sslContext))
        .build();
final PoolingHttpClientConnectionManager poolingConnectionManager = new
        PoolingHttpClientConnectionManager(socketFactoryRegistry);
```

Step 3: Create Rest Template for making the HTTP Request

- Create the Rest Template using the above connection manager. HTTP requests can be made using the rest template that will pass the Client Certificate.

```
// Create Rest Template
final CloseableHttpClient httpClientBuilder = HttpClientBuilder.create()
        .setConnectionManager(poolingConnectionManager).build();
final HttpComponentsClientHttpRequestFactory requestFactory = new
        HttpComponentsClientHttpRequestFactory();
requestFactory.setHttpClient(httpClientBuilder);
RestTemplate restTemplate = new RestTemplate(requestFactory);
```

Step 3 :: Call API and capture response

- URL = The endpoint of the API
- HttpMethod.Post = “POST”
- httpEntity = payload and headers compiled earlier
- String.class = Class for response type (String in this example)

```
HttpEntity<String> httpEntity = new HttpEntity<>(payload, headers);
ResponseEntity<String> apiResponse = restTemplate.exchange(url, HttpMethod.POST,
httpEntity, String.class);
```

For debugging, you can print the response message to view the API response

```
System.out.println(apiResponse.toString());
```

Appendix:

For additional information and resources, check the below documentation

.NET Sample Code:
https://github.com/americanexpress/amex-api-dotnet-client-core

Amex for Developer’s Portal:
https://developer.americanexpress.com/

HMAC Authentication Information:
https://developer.americanexpress.com/documentation#client-authentication-hmac

Full JAVA Code Payload Sample

Sponsored Merchant Client

```java
package com.company;

import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;

public class SponsoredMerchantClient {

    private static final String clientId = "YOUR CLIENT ID";
    private static final String clientSecret = "YOUR CLIENT SECRET";
    private static final String host = "HOST URL";
    private static final String resourcePath = "ENDPOINT URL";
    private static final String httpMethod = "POST";
    private static final String payload = "YOUR PAYLOAD HERE";
    private static final int port = 443;
    private static final String url = "https://" + host + resourcePath;

    public String sendSponsoredMerchantData() throws Exception {

        final String authorizationMac = HeadersBuilder.generateHmac(clientId,
clientSecret, resourcePath, host, port, httpMethod, payload);
        final HttpHeaders httpHeaders = HeadersBuilder.generateHeaders(clientId,
authorizationMac);
        final ResponseEntity<String> httpRequest = HTTPBuilder.createRestTemplate(url,
payload, httpHeaders);
        System.out.println(httpRequest.toString());
        return httpRequest.toString();

    }
}
```

Headers Builder

```
package com.company;

import org.springframework.http.HttpHeaders;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.util.UUID;

public class HeadersBuilder {
    private static final String HMAC_SHA256_ALGORITHM = "HmacSHA256";

    public static HttpHeaders generateHeaders(String clientId, String authHmac) {

        //create the headers object after hmac is generated
        HttpHeaders headers = new HttpHeaders();
        headers.add(HttpHeaders.CONTENT_TYPE, "application/json");
        headers.add("X-AMEX-API-KEY", clientId);
        headers.add("Authorization", authHmac);

        return headers;
    }


    public static String generateHmac(String clientId, String clientSecret, String
resourcePath,
                                      String host, int port, String httpMethod, String
payload) throws Exception {

        // get time in milliseconds
        String ts = String.valueOf(System.currentTimeMillis());

        // nonce must be unique for each request
        String nonce = UUID.randomUUID().toString();

        // create bodyhash by hashing payload
        SecretKeySpec signingKey = new SecretKeySpec(clientSecret.getBytes(),
HMAC_SHA256_ALGORITHM);
        Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
        mac.init(signingKey);
        byte[] bodyBytes = mac.doFinal(payload.getBytes());
        String bodyhashString = Base64.getEncoder().encodeToString(bodyBytes);


        // The order of the MAC components is critical
        // Timestamp + \n + nonce + \n + httpMethod + \n + path + \n + host + \n +
port + \n + hash + \n

        String macInput = ts + "\n" + nonce + "\n" + httpMethod + "\n" + resourcePath
+ "\n" + host
                + "\n" + port + "\n" + bodyhashString + "\n";

        // create mac signature by hashing baseString
        byte[] macBytes = mac.doFinal(macInput.getBytes());
        String macString = Base64.getEncoder().encodeToString(macBytes);

        // build and return authorization header string
        return "MAC id=\"" + clientId + "\",ts=\"" + ts + "\",nonce=\"" + nonce +
                "\",bodyhash=\"" + bodyhashString + "\",mac=\"" + macString + "\"";
    }
}
```

HTTP Builder

```
package com.company;

import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContexts;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
import org.springframework.web.client.RestTemplate;

import javax.net.ssl.SSLContext;
import java.io.FileInputStream;
import java.security.KeyStore;

public class HTTPBuilder {

    static ResponseEntity<String> createRestTemplate(String url, String payload,
HttpHeaders headers) throws Exception {

        // Create SSL Context
        final KeyStore keyStore = KeyStore.getInstance("PKCS12");
        final String keyPassPhrase = "PRIVATE KEY PASSPHRASE";
        String pathToCertificate = "PATH TO CERTIFICATE";
        keyStore.load(new FileInputStream(pathToCertificate),
keyPassPhrase.toCharArray());
        SSLContext sslContext = SSLContexts.custom().loadKeyMaterial(keyStore,
keyPassPhrase.toCharArray()).build();

        // Create connection manager
        final Registry<ConnectionSocketFactory> socketFactoryRegistry =
RegistryBuilder
                .<ConnectionSocketFactory>create().register("https", new
SSLConnectionSocketFactory(sslContext))
                .build();
        final PoolingHttpClientConnectionManager poolingConnectionManager = new
                PoolingHttpClientConnectionManager(socketFactoryRegistry);

        // Create Rest Template
        final CloseableHttpClient httpClientBuilder = HttpClientBuilder.create()
                .setConnectionManager(poolingConnectionManager).build();
        final HttpComponentsClientHttpRequestFactory requestFactory = new
                HttpComponentsClientHttpRequestFactory();
        requestFactory.setHttpClient(httpClientBuilder);
        RestTemplate restTemplate = new RestTemplate(requestFactory);


        HttpEntity<String> httpEntity = new HttpEntity<>(payload, headers);
        ResponseEntity<String> apiResponse = restTemplate.exchange(url,
HttpMethod.POST, httpEntity, String.class);

        System.out.println(apiResponse.toString());

        return restTemplate.exchange(url, HttpMethod.POST, httpEntity, String.class);
    }
}
```

Sample Payload (Format JSON)

```json
{
  "se_setup_request_count": 1,
  "message_id": "egr2bt362",
  "se_setup_requests": [
    {
      "record_number": "0000036500",
      "participant_se": "1021311634",
      "submitter_id": "1030026553",
      "se_detail_status_code": "36500",
      "se_status_code_change_date": "2015/12/25",
      "language_preference_code": "EN",
      "japan_credit_bureau_indicator": "0000036500",
      "marketing_indicator": "Y",
      "ownership_type_indicator": "D",
      "seller_transacting_indicator": "Y",
      "client_defined_code": "36500",
      "seller": {
        "seller_id": "GSMF093019APIX1006",
        "seller_url": "www.gsmfautomationtool.com/acquisition",
        "seller_status": "Success",
        "seller_mcc": "5999",
        "seller_legal_name": "John Doe",
        "seller_dba_name": "John Doe",
        "seller_business_registration_number": "0000036500",
        "seller_business_phone_number": "9914023611",
        "seller_email_address": "john.doe@example.com",
        "seller_currency_code": "USD",
        "seller_start_date": "2015/12/25",
        "seller_term_date": "2015/12/26",
        "seller_charge_volume": "36500",
        "seller_transaction_count": "425",
        "seller_chargeback_count": "425",
        "seller_chargeback_amount": "425",
        "seller_street_address": {
          "address_line_1": "100 Elm Street",
          "address_line_2": "Oak Avenue",
          "address_line_3": "Maple Court",
          "address_line_4": "Third Floor",
          "address_line_5": "Suite A",
          "city_name": "New York",
          "region_code": "NY",
          "postal_code": "85032",
          "country_code": "US"
        }
      },
      "significant_owners": {
        "first_owner": {
          "first_name": "FOFIRSTNM001",
          "last_name": "Smith",
          "identification_number": "0000036500",
          "date_of_birth": "2015/12/27",
          "street_address": {
            "address_line_1": "100 Elm Street",
            "address_line_2": "Oak Avenue",
            "address_line_3": "Maple Court",
            "address_line_4": "Third Floor",
            "address_line_5": "Suite A",
            "city_name": "New York",
            "region_code": "New York",
            "postal_code": "85032",
            "country_code": "US"
          }
        },        "second_owner": {
          "first_name": "Adam",
          "last_name": "Smith",
          "identification_number": "0000036500",
          "date_of_birth": "2015/12/28",
          "street_address": {
            "address_line_1": "100 Elm Street",
            "address_line_2": "Oak Avenue",
            "address_line_3": "Maple Court",
            "address_line_4": "Third Floor",
            "address_line_5": "Suite A",
            "city_name": "New York",
            "region_code": "New York",
            "postal_code": "85032",
            "country_code": "US"
          }
        },
        "third_owner": {
          "first_name": "Adam",
          "last_name": "Smith",
          "identification_number": "0000036500",
          "date_of_birth": "2015/12/29",
          "street_address": {
            "address_line_1": "100 Elm Street",
            "address_line_2": "Oak Avenue",
            "address_line_3": "Maple Court",
            "address_line_4": "Third Floor",
            "address_line_5": "Suite A",
            "city_name": "New York",
            "region_code": "New York",
            "postal_code": "85032",
            "country_code": "US"
          }
        },
        "fourth_owner": {
          "first_name": "Adam",
          "last_name": "Smith",
          "identification_number": "0000036500",
          "date_of_birth": "2015/12/30",
          "street_address": {
            "address_line_1": "100 Elm Street",
            "address_line_2": "Oak Avenue",
            "address_line_3": "Maple Court",
            "address_line_4": "Third Floor",
            "address_line_5": "Suite A",
            "city_name": "New York",
            "region_code": "New York",
            "postal_code": "85032",
            "country_code": "US"
          }
        }
      },
      "authorized_signer": {
        "first_name": "Adam",
        "last_name": "Smith",
        "identification_number": "0000036500",
        "date_of_birth": "2015/12/31",
        "street_address": {
          "address_line_1": "100 Elm Street",
          "address_line_2": "Oak Avenue",
          "address_line_3": "Maple Court",
          "address_line_4": "Third Floor",
          "address_line_5": "Suite A",
          "city_name": "New York",
          "region_code": "New York",
          "postal_code": "85032",
          "country_code": "US"
        },
        "title": "MR."
      },
      "sale": {
        "channel_indicator_code": "DS",
        "channel_name": "CN",
        "represent_id": "36500",
        "iso_register_number": "0000036500"
      }
    }
  ]
}
```