Signing requests with Signature Version 6

Signing Requests with Signature Version 6

0. Secret Key

We are using private key (symmetric key) encryption to generate signature to authenticate requests received from client and to sign response sent to the client.

Secret key can be accessed here: https://amazonpay.amazon.in/integration-details using the credentials for the merchant account.

Each key has a public access key id associated and a secret value, the access key id should be sent in HTTP Authorization header to let us know which key is being used.

The secret value of a key should never be stored as plain text, should be stored in a secure storage in encrypted form. Secret value should be retrieved and decrypted at the runtime and removed from the system after using it.

If the secret value is compromised, Amazon should be notified immediately, to disable the use of compromised key and provide a new key.

1. Create a Canonical request

- To begin the signing process, create a string that includes information from your request in a standardized (canonical) format. This ensures that when we receive the request, we can calculate the same signature that you calculated.
- Follow the steps here to create a canonical version of the request. Otherwise, your version and the version calculated by us won't match and the request will be denied.
- The following example shows the pseudocode to create a canonical request.

Example canonical request pseudocode:

```
CanonicalRequest =
HTTPRequestMethod + '\n' +
hostname/uri + '\n' +
formattedQueryParameters +'\n'+
formattedHeaders + '\n' +
formattedRequestParameters
```

formattedQueryParameters / formattedHeaders / formattedRequestParameters are the list of key- value pairs of sorted parameters joined by '&'. Each of the key-value string is rfc3886 encoded.

Example canonical request for POST / PUT Method:

```
POST
amazonpay-sandbox.amazon.in/v1/offline/payments/charge


x-amz-algorithm=AWS4-HMAC-SHA384&x-amz-client-id=A2XMNOQAN8MC64&x-amz-
date=20200906T043202Z&x-amz-expires=500&x-amz-source=Browser&x-amz-user-
agent=Postman&x-amz-user-ip=52.95.75.13
amount=.1&attributableProgram=S2SPay&chargeId=api_testing_262&currencyCode=I
NR&customerIdType=Barcode&customerIdValue=4025914314671133&intent=AuthorizeA
ndCapture&merchantId=A2XMNOQAN8MC64&storeDetail=%7BstoreIdType%3DMERCHANT_ST
ORE_ID%2C%20storeId%3DTest_Store_ID_1%7D
```

NOTE : In POST / PUT API since there is no query parameters so an empty string should passed.

Example canonical request for GET Method :

```
GET
amazonpay.amazon.in/v1/offline/payments/refund
merchantId=A2XMNOQAN8MC64&txnId=Refundtest5459-k&txnIdType=MerchantTxnId
x-amz-algorithm=AWS4-HMAC-SHA384&x-amz-client-id=A2XMNOQAN8MC64&x-amz-
date=20200906T055702Z&x-amz-expires=500&x-amz-source=Browser&x-amz-user-
agent=Postman&x-amz-user-ip=52.95.75.13


```

NOTE : In GET API since there is no body parameters so an empty string should be passed after writing the header parameters and adding the new line characters.

It is mandatory to verify the signature coming in response to the APIs before accepting the response. Below you can find sample canonical response formats to verify signatures at your end.

Example canonical Response for POST Method :

```
POST
amazonpay.amazon.in/v1/offline/payments/refund

x-amz-algorithm=AWS4-HMAC-SHA384&x-amz-date=20200906T071710Z&x-amz-request-
id=ab6e5e05-1f15-48a1-ae39-84fd9ae62a17
amazonRefundId=S04-8640119-6506863-R007626&amount=0.10&createTime=2020-09-
06T05%3A34%3A35.129Z&currencyCode=INR&refundId=Refundtest5459-
k&refundedFee=0.00&status=Approved&updateTime=2020-09-06T05%3A35%3A05.488Z
```

Example canonical Response for GET Method :

```
GET
amazonpay.amazon.in/v1/offline/payments/refund

x-amz-algorithm=AWS4-HMAC-SHA384&x-amz-date=20200906T072009Z&x-amz-request-
id=33c6c2f3-7de0-4e31-bb5e-7e637da8a04d
amazonRefundId=S04-8640119-6506863-R007626&amount=0.10&createTime=2020-09-
06T05%3A34%3A35.129Z&currencyCode=INR&refundId=Refundtest5459-
k&refundedFee=0.00&status=Approved&updateTime=2020-09-06T05%3A35%3A05.488Z
```

NOTE : Response is returned in the body hence canonical request for both POST and GET method follow the same pattern for signature verification.

2. Create a String to Sign

- The 'string to sign' includes meta information about your request and about the canonical request you created in step1. You will use string to sign in the next step along with the secret-key to generate the signature.
- To create the string to sign, concatenate the algorithm, date and time, credential scope,and digest of canonical request, as shown in the pseudocode :

```
StringToSign   =
Algorithm + \n +
RequestDateTime + \n +
CredentialScope + \n +
HashedCanonicalRequest
```

Detailed Steps

1. Start with Algorithm designation, followed by a newline character. This value is the hashing algorithm used to calculate the digest of canonical request. We are using SHA384, algorithm designation is `AWS4-HMAC-SHA256`

   ```
   AWS4-HMAC-SHA384\n
   ```

2. Append the request date value, followed by a newline character. The date is specified with ISO8601 basic format in ***UTC time zone***. The format is: `YYYYMMDD'T'HHMMSS'Z'`

   If you are using JAVA and SimpleDateFormat then please use date format as `yyyyMMdd'T'HHMMSS'Z'`

   ```
   20181130T120049Z\n
   ```

3. Append the credential scope value, followed by a newline character.

   Credential scope is a string that includes the date `YYYYMMDD`, the region you are targeting `eu-west-1`, the service you are requesting `AmazonPay` and a termination string `aws4_request`, separated by `/`

   ```
   20181130/eu-west-1/AmazonPay/aws4_request\n
   ```

4. Append the hash of the canonical request that you created in step1. This value is not followed by a newline character. This hashed canonical request must be SHA384 encoded.

   ```
   316097191c79d640a0f0b434e043072c45ff0a51ab27972252e1d9ec47fcaa29f7b70ca010
   587390ed108513fed247f4
   ```

Sample string to sign:

```
AWS4-HMAC-SHA384
20181130T120049Z
20181130/eu-west-1/AmazonPay/aws4_request
316097191c79d640a0f0b434e043072c45ff0a51ab27972252e1d9ec47fcaa29f7b70ca010587
390ed108513f
```

3. Calculate the Signature

Before you calculate a signature, to derive a signing key from your secret key.

Because the derived signing key is specific to date, service and region, it offers a greater degree of protection.

You don't just use your secret-key to sign the request, you use the signing key and the string to sign that you created in step2 as the inputs to a keyed hash function. The hex-encoded result from the keyed hash function is the signature.

To calculate signature

1. Derive your signing key. To do this, use your secret key to create a series of hash-based message authentication codes (HMACs). This is shown in the following pseudocode, where HMAC(key, data) represents an HMAC-SHA384 function that returns output in binary format. The result of each hash function becomes input for the next one.

```
kSecret = your secret access key
kDate = HMAC("AWS4" + kSecret, Date)
kRegion = HMAC(kDate, Region)
kService = HMAC(kRegion, Service)
kSigning = HMAC(kService, "aws4_request")
```

- Note that the date used in the hashing process is in the format YYYYMMDD (for example, 20150830), and does not include the time. This should be same as the date used in previous steps.

- Make sure you specify the HMAC parameters in the correct order for the programming language you are using. This example shows the key as the first parameter and the data (message) as the second parameter, but the function that you use might specify the key and data in a different order.

- Use the digest (binary format) for the key derivation. Most languages have functions to compute either a binary format hash, commonly called a digest, or a hex-encoded hash, called a hexdigest. The key derivation requires that you use a binary-formatted digest.

Example of signing key :

```
c4afb1cc5771d871763a393e44b703571b55cc28424d1a5e86da6ed3c154a4b9
```

2. Calculate the signature. To do this, use the signing key that you derived and the string to sign as inputs to the keyed hash function.

   After you calculate the signature, encode it using base64url encoding.

   Pseudocode to generate the signature:

   ```
   signature = HexEncode(HMAC(derived signing key, string to sign)
   ```

4. Sample codes to generate / Verify signatures

Java - https://amazonpay.s3-us-west-2.amazonaws.com/Signature/JAVA_GenerateSignature.zip

DotNet - https://amazonpay.s3-us-west-2.amazonaws.com/Signature/DotNet_GenrateSignature.zip

5. References

- How to derive a Signing key in several languages: https://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html
- AWS signature v4 process: https://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html AWS
- signature v4 process in python: https://docs.aws.amazon.com/general/latest/gr/sigv4-signed-request-examples.html