API Request Signing Guide

Issue 01

Date 2021-12-22

HUAWEI

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 Overview

This document describes how to call cloud service APIs that have been registered with API Gateway, through AK/SK authentication. It explains the signing process and implementation logic, and provides signature SDKs and invocation examples of different programming languages, such as Java, Go, Python, and C.

NOTE

1. For the authentication of certain cloud service APIs that are not registered with API Gateway, see the *API Reference* of the corresponding service.

   The *API Reference* contains a section named "Calling APIs" that describes API authentication methods.
2. The SDK of each programming language is packaged in the sample code and can be obtained separately. You can integrate the SDK into your application by referring to the API calling example.
3. If you cannot find any signing example of the programming language you use in this document, please sign requests by referring to **AK/SK Authentication**.
4. Alternatively, you can call APIs using a token. For details about the use of a token, see the *API Reference* of the relevant service.
5. AK/SK authentication supports API requests with a body less than or equal to 12 MB. For API requests with a larger body, token authentication is recommended.
6. For the APIs provided by a cloud service, see the *API Reference* of the cloud service.
7. The local time on the client must be synchronized with the clock server to avoid a large offset in the value of the **X-Sdk-Date** request header.

   API Gateway checks the time format and compares the time in the header with the time when API Gateway receives the request. If the time difference exceeds 15 minutes, API Gateway will reject the request.

2 AK/SK Signing and Authentication Algorithm

2.1 AK/SK Authentication Process

The AK/SK-based authentication process at the client is as follows:

1. **Construct a standard request**.

   Assemble the request content according to the rules of API Gateway, ensuring that the client signature is consistent with that in the backend request.
2. Create a **to-be-signed string** using the standard request and other related information.
3. **Calculate a signature** using the AK/SK and to-be-signed string.
4. **Add the generated signature to an HTTP request as a header** or query string.

The following describes the process in detail.

NOTE

You can follow the instructions provided in this chapter to sign API requests.

You can also call APIs by using the signing SDKs and sample code of common languages described in **Signing SDKs and Demo**.

2.2 Constructing a Standard Request

To access an API through AK/SK authentication, create a standard request, and then sign the request. The client must follow the same request specifications as API Gateway so that each HTTP request can obtain the same signing result from the frontend and backend to complete identity authentication.

The pseudocode of standard HTTP requests is as follows:

```
CanonicalRequest =
    HTTPRequestMethod + '\n' +
    CanonicalURI + '\n' +
    CanonicalQueryString + '\n' +
    CanonicalHeaders + '\n' +
```



```
    SignedHeaders + '\n' +
    HexEncode(Hash(RequestPayload))
```

The following procedure uses the Virtual Private Cloud (VPC) query API as an example to describe how to construct a standard request.

Original request:

```
GET https://service.region.example.com/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?
limit=2&marker=13551d6b-755d-4757-b956-536f674975c0 HTTP/1.1
Host: service.region.example.com
X-Sdk-Date: 20191115T033655Z
```

Step 1 Specify an HTTP request method (**HTTPRequestMethod**) and end with a carriage return line feed (CRLF).

HTTP request methods include GET, PUT, POST, and so on. For example:

```
GET
```

Step 2 Add a standard URI (**CanonicalURI**) and end with a CRLF.

Description

Path of the requested resource, which is the URI code of the absolute path.

Format

According to RFC 3986, each part of a standard URI except the redundant and relative paths must be URI-encoded. If a URI does not end with a slash (/), add a slash at its end.

Example

See the URI of each API in the *API Reference* of the corresponding cloud service. For example, the standard URI code of the VPC query API (**/v1/***{project_id}***/vpcs**) is as follows:

```
GET
/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs/
```

NOTE

During signature calculation, the URI must end with a slash (/). When a request is sent, the URI does not need to end with a slash (/).

Step 3 Add a standard query string (**CanonicalQueryString**) and end with a CRLF.

Description

Query strings. If no query strings are configured, an empty string is used.

Format

Pay attention to the following to ensure standard query strings:

- Perform URI encoding on each parameter and value according to the following rules:
  - Do not perform URI encoding on any non-reserved characters defined in RFC 3986, including A–Z, a–z, 0–9, hyphen (-), underscore (_), period (.), and tilde (~).
  - Use **%XY** to perform percent encoding on all non-reserved characters. **X** and **Y** indicate hexadecimal characters (0–9 and A–F). For example, the

space character must be encoded as **%20**, and an extended UTF-8 character must be encoded in the "%XY%ZA%BC" format.

- Add "*URI-encoded parameter name*=*URI-encoded parameter value*" to each parameter. If no value is specified, use an empty string instead. The equal sign (=) is required.

  For example, in the following string that contains two parameters, the value of parameter **parm2** is null.

  ```
  parm1=value1&parm2=
  ```

- Sort the parameters in alphabetically ascending order. For example, a parameter starting with uppercase letter **F** precedes another parameter starting with lowercase letter **b**.
- Construct a standard query string from the first parameter after sorting.

Example

The URI of the VPC query API contains two optional parameters: **limit** and **marker**. **limit** indicates the number of records displayed on each page, and **marker** indicates the start VPC ID for pagination query.

```
GET
/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs/
limit=2&marker=13551d6b-755d-4757-b956-536f674975c0
```

Step 4 Add standard headers (**CanonicalHeaders**) and end with a CRLF.

Description

List of standard request headers, including all HTTP message headers in the to-be-signed request. The X-Sdk-Date header must be included to verify the signing time, which is in the UTC time format *YYYYMMDDTHHMMSSZ* as specified in ISO 8601.

CAUTION

The local time on the client must be synchronized with the clock server to avoid a large offset in the value of the **X-Sdk-Date** request header.

API Gateway checks the time format and compares the time with the time when API Gateway receives the request. If the time difference exceeds 15 minutes, API Gateway will reject the request.

Format

CanonicalHeaders consists of multiple message headers, for example, **CanonicalHeadersEntry0 + CanonicalHeadersEntry1 + ...**. Each message header (**CanonicalHeadersEntry**) is in the format of **Lowercase(HeaderName) + ':' + Trimall(HeaderValue) + '\n'**.

NOTE

- **Lowercase** is a function for converting all letters into lowercase letters.
- **Trimall** is a function for deleting the spaces before and after a value.
- The last message header carries a CALF. Therefore, an empty line appears because the **CanonicalHeaders** field also contains a CALF according to the specifications.

Example

Requests for calling the VPC query API need to contain the **X-Sdk-Date**, **Host** (cloud service endpoint), and **Content-Type** headers.

```
GET
/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs/
limit=2&marker=13551d6b-755d-4757-b956-536f674975c0
content-type:application/json
host:service.region.example.com
x-sdk-date:20191115T033655Z
```

NOTICE

Standard message headers must meet the following requirements:

- All letters in a header are converted to lowercase letters, and all spaces before and after the header are deleted.
- All headers are sorted in alphabetically ascending order.

For example, the original headers are as follows:

```
Host: service.region.example.com\n
Content-Type: application/json;charset=utf8\n
My-header1:    a   b   c  \n
X-Sdk-Date:20190318T094751Z\n
My-Header2:    "x   y   \n
```

The message header names are converted into lowercase letters, the message headers are sorted in alphabetical order, and spaces before and after the header values are deleted. The standardized message headers are as follows:

```
content-type:application/json;charset=utf8\n
host:service.region.example.com\n
my-header1:a   b   c\n
my-header2:"x   y\n
x-sdk-date:20190318T094751Z\n
```

Step 5 Add message headers (**SignedHeaders**) for request signing, and end with a CALF.

Description

List of message headers used for request signing. This step is to determine which headers are used for signing the request and which headers can be ignored during request verification. The **X-Sdk-date** header must be included.

Format

SignedHeaders = Lowercase(HeaderName0) + ';' + Lowercase(HeaderName1) + ";" + ...

Letters in the message headers are converted to lowercase letters. All headers are sorted alphabetically and separated with commas.

Lowercase is a function for converting all letters into lowercase letters.

Example

In the following request, the **Content-Type**, **Host**, and **X-Sdk-Date** headers are used for request signing.

```
GET
/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs/
limit=2&marker=13551d6b-755d-4757-b956-536f674975c0
content-type:application/json
host:service.region.example.com
x-sdk-date:20191115T033655Z
```



```
content-type;host;x-sdk-date
```

The signed headers are as follows:

```
SignedHeaders=content-type;host;x-sdk-date
```

For details about how to add headers to a request, see **Adding the Signature to the Request Header**.

Step 6 Use a hash function, such as SHA-256, to create a hash value based on the body (**RequestPayload**) of the HTTP or HTTPS request.

Description

Request message body. The message body needs two layers of conversion (**HexEncode(Hash(*RequestPayload*))**). **Hash** is a function for generating message digest. Currently, SHA-256 is supported. **HexEncode** is the Base16 encoding function for returning a digest consisting of lowercase letters. For example, **HexEncode("m")** returns **6d** instead of **6D**. Each byte you enter is expressed as two hexadecimal characters.

NOTE

If **RequestPayload** is null, the null value is used for calculating a hash value.

Example

This example uses GET as an example, and the request body is empty. After hash processing, the request body (empty string) is as follows:

```
GET
/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs/
limit=2&marker=13551d6b-755d-4757-b956-536f674975c0
content-type:application/json
host:service.region.example.com
x-sdk-date:20191115T033655Z

content-type;host;x-sdk-date
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

A standard request is constructed.

Step 7 Perform hash processing on the standard request in the same way as that on the **RequestPayload**. After hash processing, the standard request is expressed with lowercase hexadecimal strings.

Algorithm pseudocode:
Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))

Example of the standard request after hash processing:

```
b25362e603ee30f4f25e7858e8a7160fd36e803bb2dfe206278659d71a9bcd7a
```

----End

2.3 Creating a To-Be-Signed String

After a standard HTTP request is constructed and the request hash value is obtained, create a to-be-signed string by combining them with the signature algorithm and signing time.


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

Parameters in the pseudocode are described as follows:

- **Algorithm**

  Signature algorithm. For SHA-256, the value is SDK-HMAC-SHA256.

- **RequestDateTime**

  Request timestamp, which is the same as **X-Sdk-Date** in the request header. The format is *YYYYMMDDTHHMMSSZ*.

- **HashedCanonicalRequest**

  Hash value generated using the SHA-256 algorithm based on the standard request constructed in **Constructing a Standard Request**.

In this example, the following to-be-signed string is obtained:

```
SDK-HMAC-SHA256
20191115T033655Z
b25362e603ee30f4f25e7858e8a7160fd36e803bb2dfe206278659d71a9bcd7a
```

2.4 Calculating the Signature

Use the SK and to-be-signed string as the input of the encryption hash function, and convert the calculated binary signature into a hexadecimal expression.

The pseudocode is as follows:

```
signature = HexEncode(HMAC(Secret Access Key, string to sign))
```

HMAC indicates hash calculation, and **HexEncode** indicates hexadecimal conversion. **Table 2-1** describes the parameters in the pseudocode.

Table 2-1 Parameter description

Parameter	Description
Secret Access Key	Signature key
string to sign	Character string to be signed

If the SK is **MFyfvK41ba2giqM7Uio6PznpdUKGpownRZlmVmHc**, the calculated signature is as follows:

```
7be6668032f70418fcc22abc52071e57aff61b84a1d2381bb430d6870f4f6ebe
```

2.5 Adding the Signature to the Request Header

Add the signature to the **Authorization** HTTP header. The **Authorization** header is used for identity authentication and not included in the **SignedHeaders**.

The pseudocode is as follows:


```
Pseudocode for Authorization header creation:
Authorization: algorithm Access=Access key, SignedHeaders=SignedHeaders, Signature=signature
```

There is no comma but a space before the algorithm and **Access**. **SignedHeaders** and **Signature** must be separated with commas.

The signed headers are as follows:

```
SDK-HMAC-SHA256 Access=QTWAOYTTINDUT2QVKYUC, SignedHeaders=content-type;host;x-sdk-date,
Signature=7be6668032f70418fcc22abc52071e57aff61b84a1d2381bb430d6870f4f6ebe
```

The signed headers are added to the HTTP request for identity authentication. If the identity authentication is successful, the request is sent to the corresponding cloud service for processing.

The complete request that contains the signature information is as follows:

```
GET /v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?limit=2&; marker=13551d6b-755d-4757-
b956-536f674975c0 HTTP/1.1
Host: service.region.example.com
Content-Type: application/json
x-sdk-date: 20191115T033655Z
Authorization: SDK-HMAC-SHA256 Access=QTWAOYTTINDUT2QVKYUC, SignedHeaders=content-
type;host;x-sdk-date, Signature=7be6668032f70418fcc22abc52071e57aff61b84a1d2381bb430d6870f4f6ebe
```

Example request for calling an API with a curl command:

```
curl -X GET "https://service.region.example.com/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?
limit=2&marker=13551d6b-755d-4757-b956-536f674975c0" -H "content-type: application/json" -H "X-Sdk-
Date: 20191115T033655Z" -H "host: service.region.example.com" -H "Authorization: SDK-HMAC-SHA256
Access=QTWAOYTTINDUT2QVKYUC, SignedHeaders=content-type;host;x-sdk-date,
Signature=7be6668032f70418fcc22abc52071e57aff61b84a1d2381bb430d6870f4f6ebe" -d $''
```


3 AK/SK Signing and Authentication Guide

3.1 AK/SK Signing and Authentication Process

The AK/SK signing and authentication process is as follows:

1. API calling information is collected.
   The information to be collected includes:
   - Endpoint and URI that will constitute the request URL
   - AK/SK used for signing and authentication
   - Project ID and subproject ID
   - Account name and account ID

   Table 3-1 Required information to collect

Item	Description
Endpoint	Endpoint of a cloud service in a region. For details on how to obtain an endpoint, see **Obtaining an Endpoint**.
URI	API request path and parameters. Obtain the request path and parameters from the *API Reference* of the cloud service.
AK/SK	Access key ID (AK) and secret access key (SK), which are used to sign API requests. For details on how to obtain the AK/SK, see **Obtaining an AK/SK**.
Project_Id	Project ID, which needs to be configured for the URI of most APIs to identify different projects. For details on how to obtain a project ID, see **Obtaining a Project ID**.

Item	Description
X-Project-Id	Subproject ID, which is used in multi-project scenarios. To access resources in a subproject through AK/SK-based authentication, the **X-Project-Id** field must be added to the request header. For details on how to obtain a subproject ID, see **Obtaining a Project ID**.
X-Domain-Id	Account ID, which is used to: • Obtain a token for token authentication. • Call APIs of global services, such as Content Delivery Network (CDN), through AK/SK authentication. For details on how to obtain the account ID, see **Obtaining the Account Name and Account ID**.

2. APIs are called.

   This document provides signature SDKs and API calling examples in multiple languages, such as Java, Go, Python, and C. You can find the language you need in **Signing SDKs and Demo** and integrate the SDK into your application by referring to the examples and API calling description.

3.2 Obtaining an Endpoint

An endpoint is the access domain name of a cloud service in a region. Each service has different domain names in different regions.

For details, see **Regions and Endpoints**.

NOTE

For all example request URLs in this document, the endpoint **service.region.example.com** is used as an example.

3.3 Obtaining an AK/SK

If an AK/SK has already been generated, skip this step. Find the downloaded AK/SK file, which is usually named **credentials.csv**.

As shown in the following figure, the file contains the username, AK, and SK

Figure 3-1 Content of the credential.csv file

	A	B	C
1	User Name	Access Key Id	Secret Access Key
2	hu[redacted]dg	QTWA[redacted]UT2QVKYUC	MFyfvK41ba2[redacted]npdUKGpownRZlmVmHc

Procedure

Step 1 **Register an account and log in to the management console.**

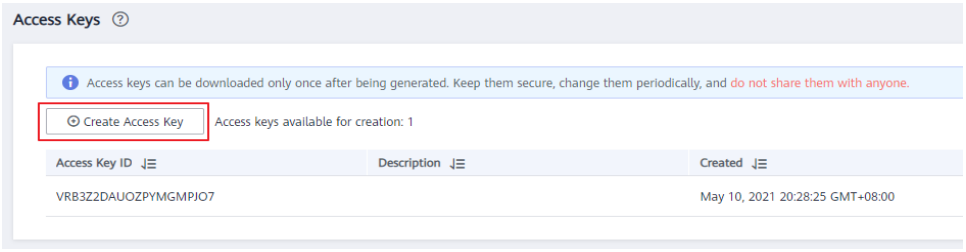
Step 2 Hover the mouse pointer over the username and choose **My Credentials** from the drop-down list.

Step 3 In the navigation pane on the left, choose **Access Keys**.

Step 4 Click **Create Access Key**.

Step 5 Enter a description, and click **OK** to download the access key. Keep the access key secure.

Figure 3-2 Obtaining an access key



----End

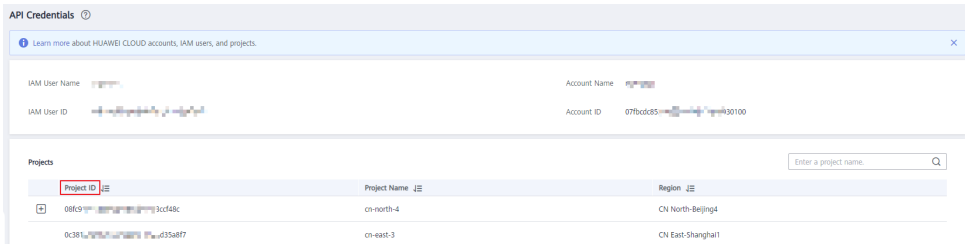
3.4 Obtaining a Project ID

A project ID is required in the URLs of some APIs when the APIs are called. It is also required when you obtain a token. Perform the following steps to obtain a project ID:

1. **Register an account and log in to the management console.**
2. Hover the mouse pointer over the username in the upper right corner, choose **My Credentials** from the drop-down list, and then view the project ID.

   Projects physically isolate cloud server resources by region, and **multiple projects can be created** in the same region to implement more fine-grained isolation. As shown in the following figure, find the region where your server locates, and obtain the corresponding project ID in the **Project ID** column.

Figure 3-3 Viewing the project ID



NOTE

To view the subproject ID, click the project to expand the subproject list.

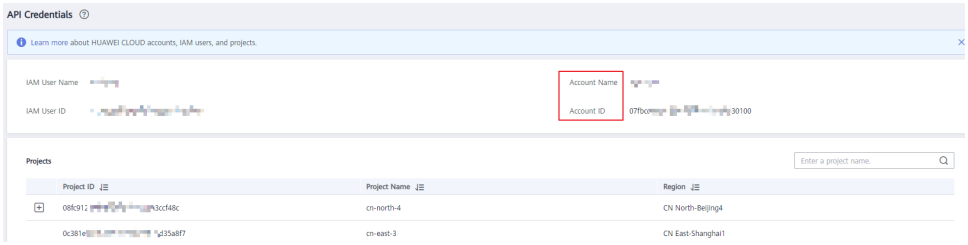
3.5 Obtaining the Account Name and Account ID

The account name and account ID are required for some URLs when an API is called. To obtain the account name and account ID, perform the following operations:

1. **Register an account and log in to the management console.**
2. Hover the mouse pointer over the username and choose **My Credentials** from the drop-down list.

   View the account name and account ID.

   Figure 3-4 Viewing the account name and account ID



3.6 Signing SDKs and Demo

3.6.1 Java

This section uses Eclipse as an example to describe how to integrate the Java SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

NOTE

The signing SDK is used for signing requests and not used for cloud service access. For the cloud service SDK, see **SDK Development Guide**.

Preparing the Environment

- Download the Eclipse installation file or package from the **Eclipse official website**, and install Eclipse or decompress the package for use.
- Download Java Development Kit 1.8.111 or a later version from the **Oracle official website**.
- Obtain the Maven repository address **https://repo.huaweicloud.com/apache/maven/maven-3/**.

Obtaining the SDK

Download the SDK at **https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/APIGW-java-sdk.zip**.

The following table shows the directory structure of the package.

Name	Description
libs\java-sdk-core-x.x.x.jar	Signing SDK and dependencies
libs\commons-codec-x.x.jar	
libs\commons-logging-x.x.jar	
libs\httpclient-x.x.x.jar	
libs\httpcore-x.x.x.jar	
src\com\apig\sdk\demo\Main.java	Sample code for signing requests
.classpath	Sample project files
.project	

To build a project with Maven, download the **java-sdk-core-*x.x.x*.jar** file in the SDK from **https://mirrors.huaweicloud.com/repository/maven/huaweicloudsdk/**.

The following are the Maven configuration items for adding the **java-sdk-core** dependency:

```
<dependency>
    <groupId>com.huawei.apigateway</groupId>
    <artifactId>java-sdk-core</artifactId>
    <version>3.0.12</version>
</dependency>
```


NOTE

If you build a project with Maven, modify the **settings.xml** file by adding the following content:

1. Add the following content to the **profiles** section:

```
<profile>
    <id>MyProfile</id>
    <repositories>
        <repository>
            <id>HuaweiCloudSDK</id>
            <url>https://mirrors.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>HuaweiCloudSDK</id>
            <url>https://mirrors.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
    </pluginRepositories>
</profile>
```

2. Add the following content to the **mirrors** section:

```
<mirror>
    <id>huaweicloud</id>
    <mirrorOf>*</mirrorOf>
    <url>https://mirrors.huaweicloud.com/repository/maven/</url>
</mirror>
```

3. Add the **activeProfiles** tag to activate the configurations.

```
<activeProfiles>
    <activeProfile>MyProfile</activeProfile>
</activeProfiles>
```

Importing the Sample Project

Step 1 Start Eclipse and choose **File** > **Import**. In the **Import** dialog box, choose **General** > **Existing Projects into Workspace**, and select the **APIGW-java-sdk-***x.x.x* folder.

Figure 3-5 Import

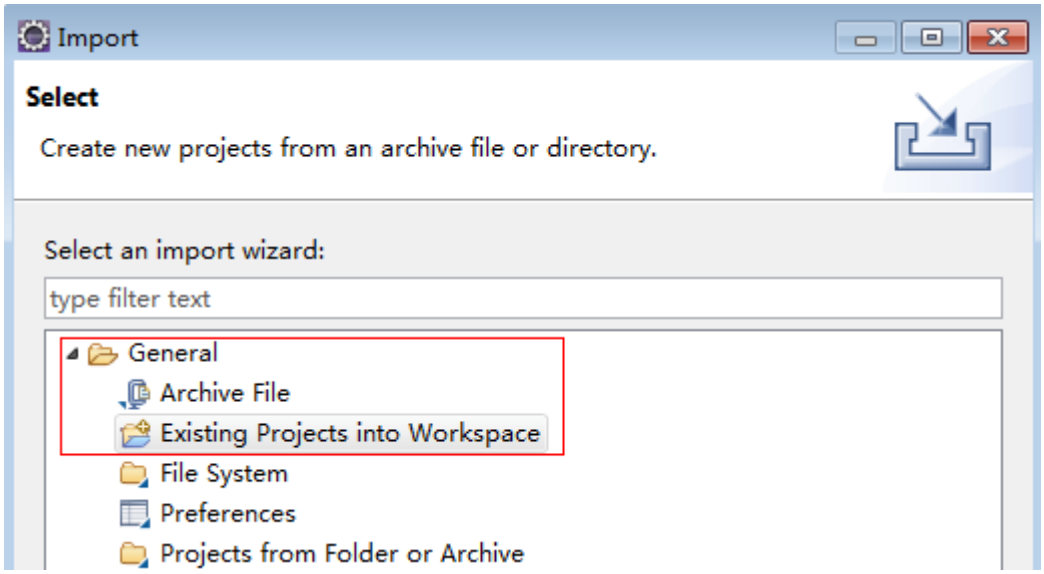
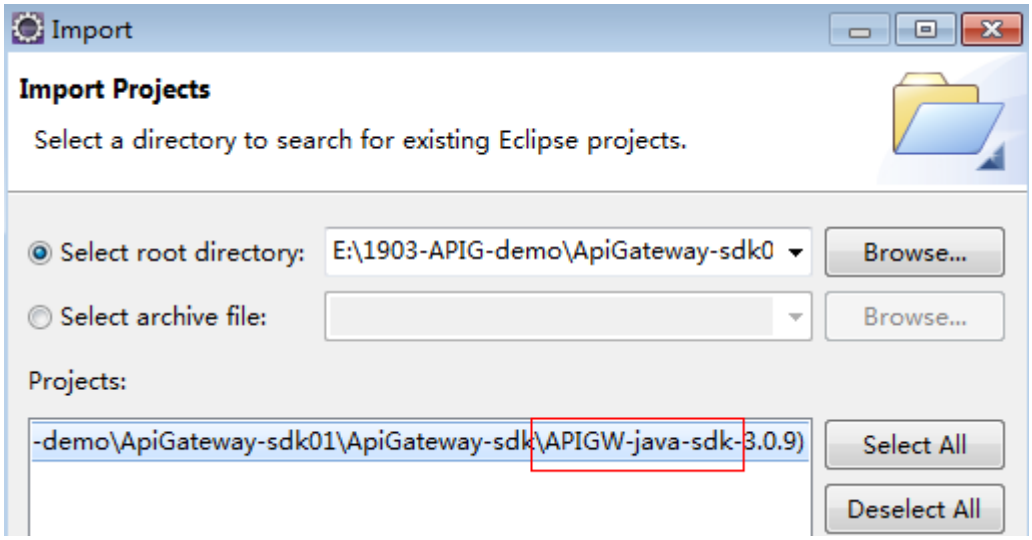
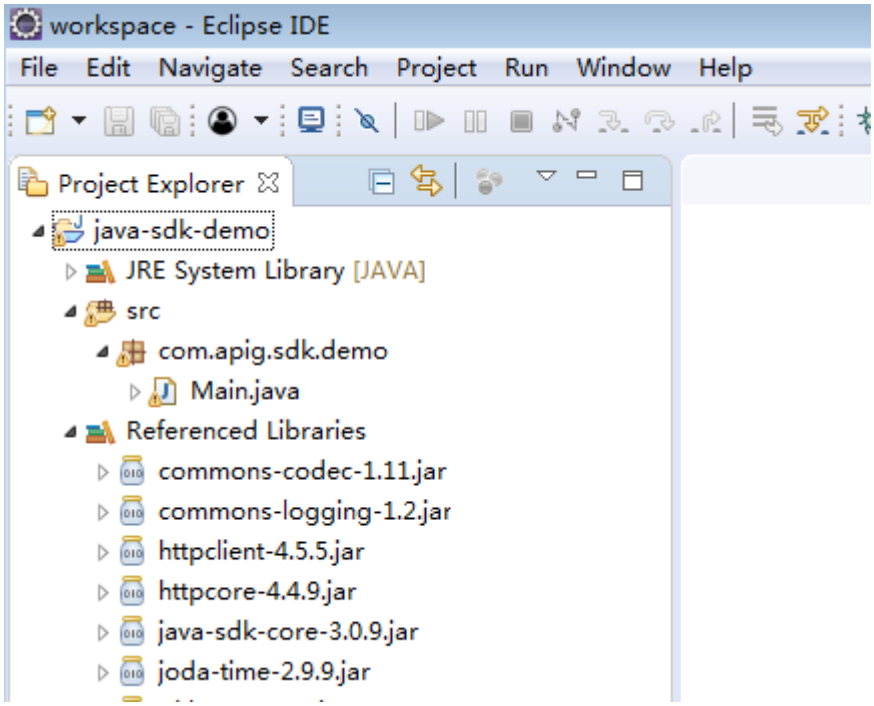


Figure 3-6 Import Projects



Step 2 Click **Finish**. The following figure shows the imported sample project.

Figure 3-7 Sample project for signing requests



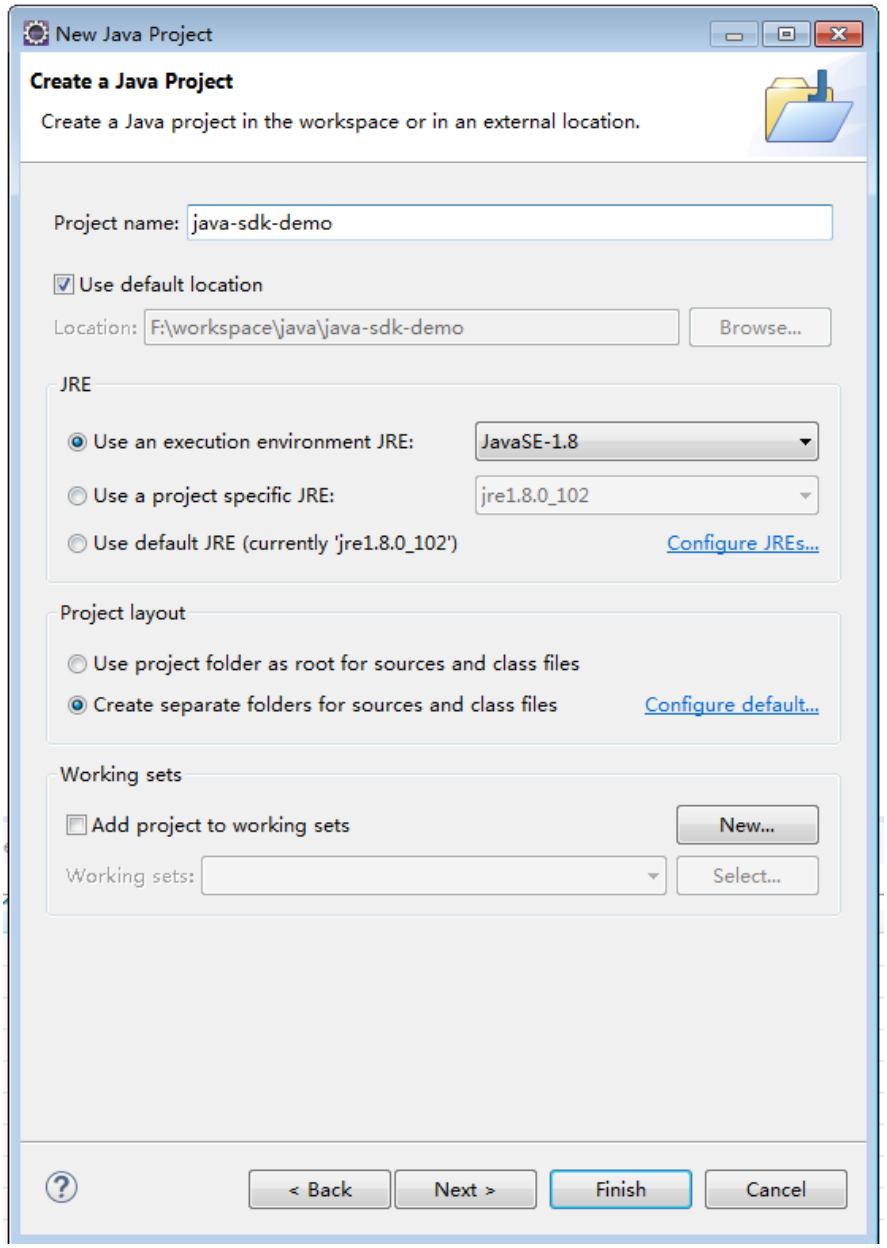
----End

NOTE

If Eclipse is installed, the JDK environment has been configured. Therefore, no more descriptions about JDK environment are provided.

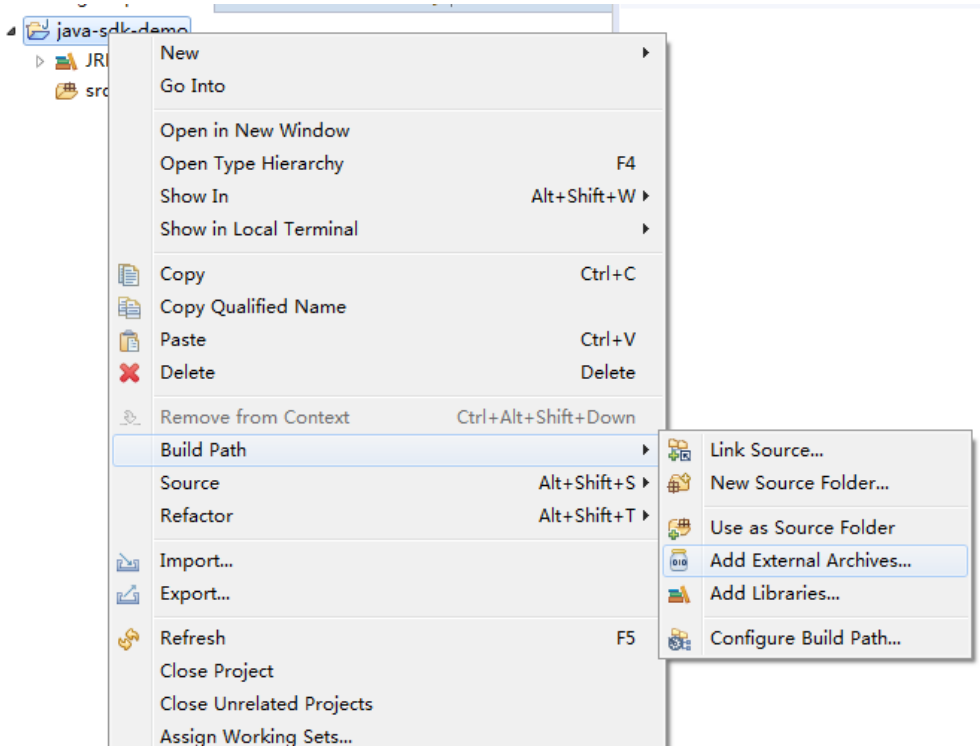
Creating a Project with the Signing SDK

Step 1 Start Eclipse and create a Java project. Specify a project name, for example, **java-sdk-demo**. Retain the default values for other parameters and click **Finish**.

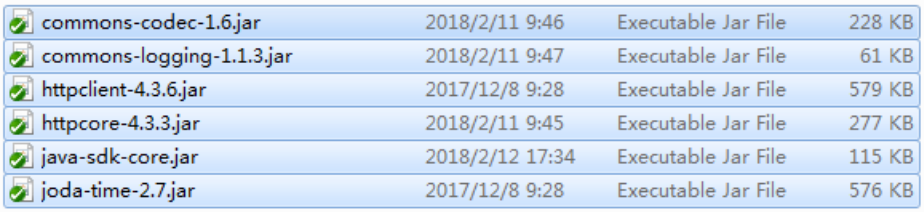


Step 2 Import the .jar files in the Java SDK.

1. Right-click the created project **java-sdk-demo** and choose **Build Path** > **Add External Archives**.



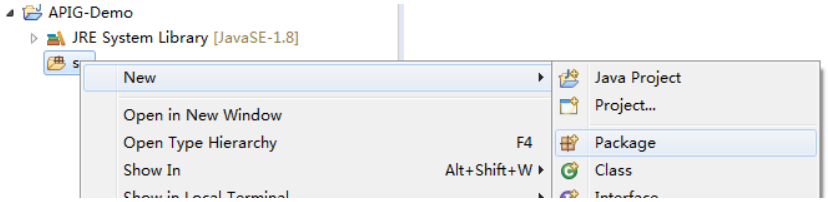
2. Select all .jar files in the **java\libs** directory.



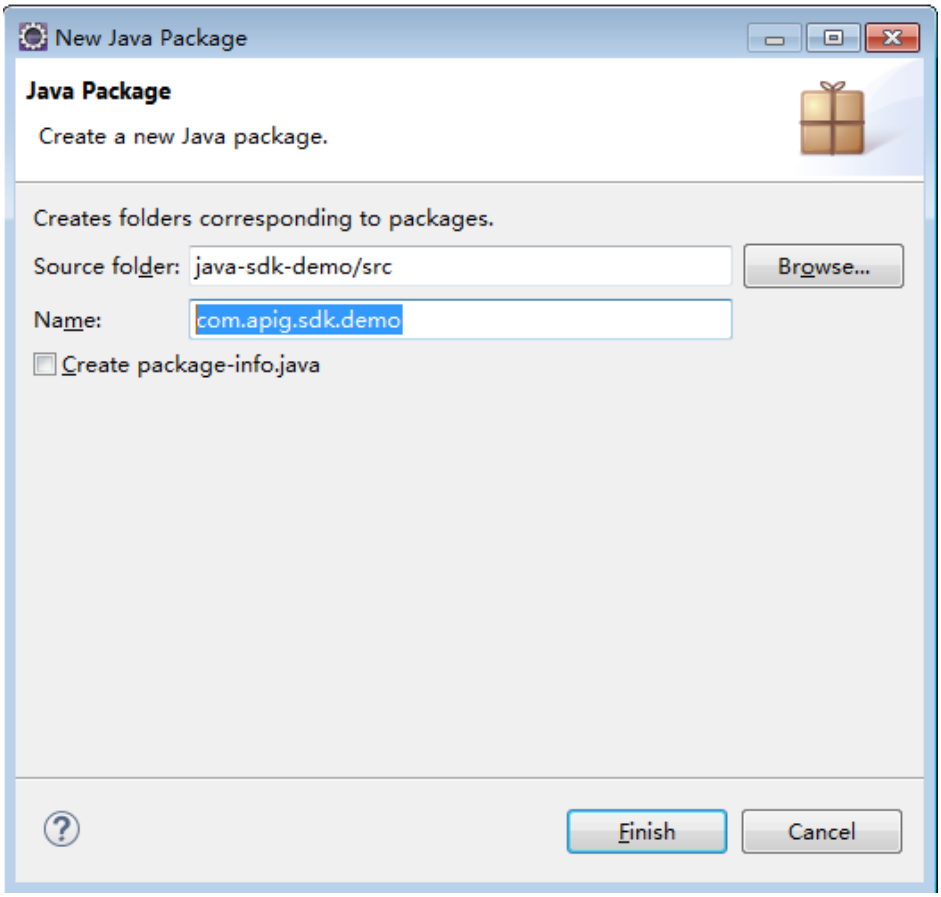
3. Click **Open**.

Step 3 Create a package and a class named **Main**.

1. Right-click **src** and choose **New** > **Package**.



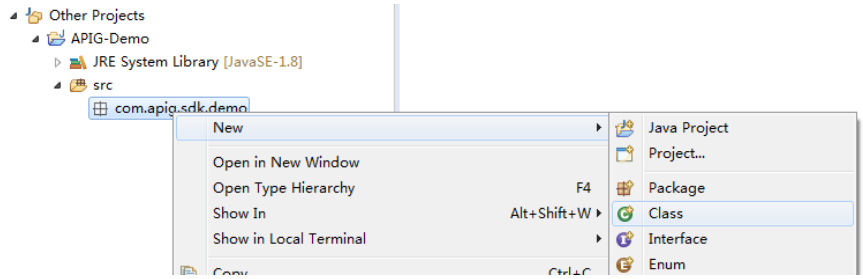
2. Enter **com.apig.sdk.demo** for **Name**.



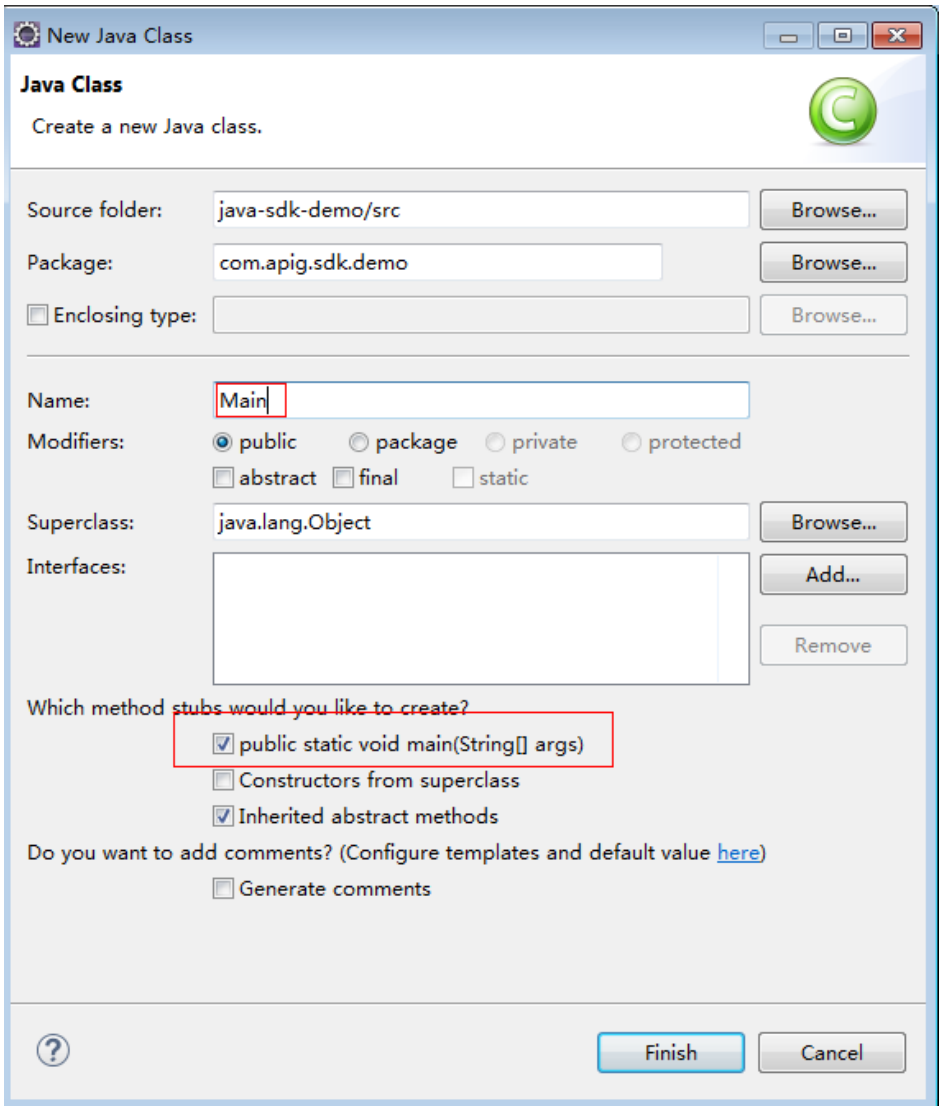
3. Click **Finish**.

   The package is created.

4. Right-click **com.apig.sdk.demo** and choose **New** > **Class**.



5. Enter **Main** for **Name** and select **public static void main(String[] args)**.



6. Click **Finish**.

   The **Main** file is created.

Step 4 The project is created.

Before using **Main.java**, enter the required code according to **Calling APIs**.

----End

Calling APIs

The sample project can be invoked after you change the **environment information**. The following is a procedure for invoking the SDK in an application to sign requests.

Step 1 Add the following references to **Main.java**:

```
import java.io.IOException;
import javax.net.ssl.SSLContext;

import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
```



```
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.conn.ssl.AllowAllHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.conn.ssl.SSLContexts;
import org.apache.http.conn.ssl.TrustSelfSignedStrategy;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
```

Step 2 Create a request and set required parameters.

Sample code and annotations:

```
Request request = new Request();
    try {
        //Set the AK/SK to sign and authenticate the request.
        request.setKey("QTWAOYTTINDUT2QVKYUC");
        request.setSecret("MFyfvK41ba2giqM7Uio6PznpdUKGpownRZlmVmHc");

        //The following example shows how to set the request URL and parameters to query a VPC list.

        //Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
        request.setMethod("GET");

        //Set a request URL in the format of https://{Endpoint}/{URI}.
         request.setUrl("https://{service}.region.example.com/v1/{project_id}/vpcs");
        //Set parameters for the request URL.
        request.addQueryStringParam("limit", "2");

        //Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id
for invoking a project-level service.
        request.addHeader("X-Project-Id", "xxx");

        //Add a body if you have specified the PUT or POST method. Special characters, such as the double
quotation mark ("), contained in the body must be escaped.
        //request.setBody("demo");
       //setBody can only be a string.

    } catch (Exception e) {
        e.printStackTrace();
        return;
    }
```

Step 3 Sign the request, access the API, and print the result.

The sample code is as follows:

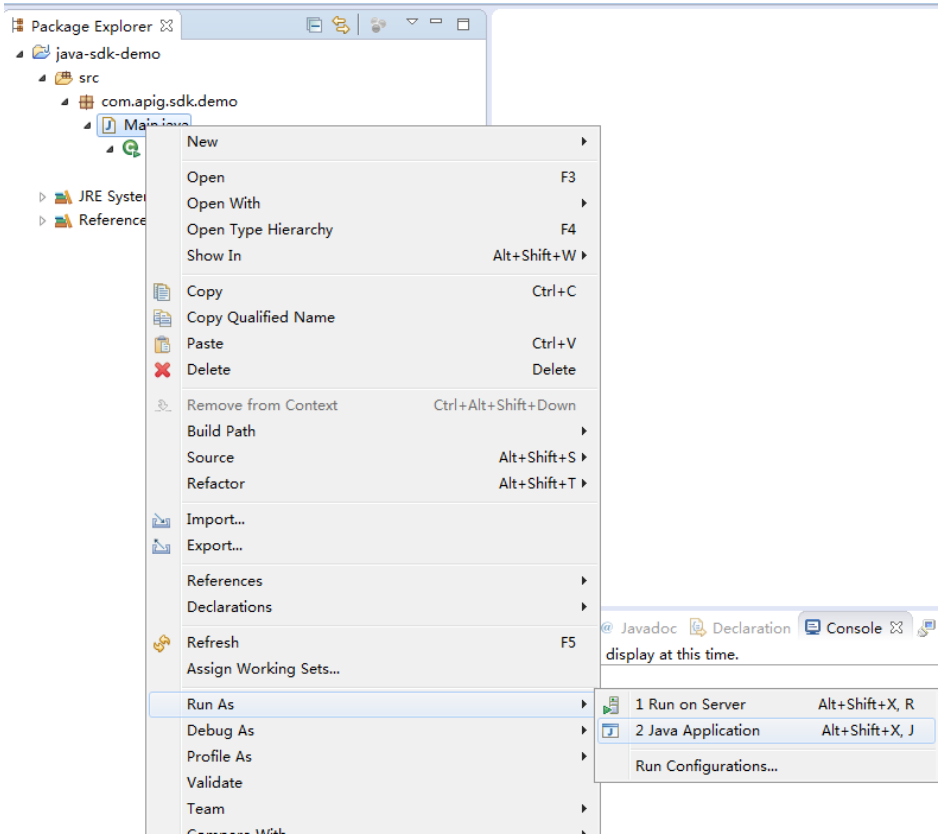
```
    CloseableHttpClient client = null;
    try
    {
        HttpRequestBase signedRequest = Client.sign(request);

        client = HttpClients.custom().build();
        HttpResponse response = client.execute(signedRequest);
        System.out.println(response.getStatusLine().toString());
        Header[] resHeaders = response.getAllHeaders();
        for (Header h : resHeaders)
        {
            System.out.println(h.getName() + ":" + h.getValue());
        }
        HttpEntity resEntity = response.getEntity();
        if (resEntity != null)
        {
            System.out.println(System.getProperty("line.separator") + EntityUtils.toString(resEntity, "UTF-8"));
        }

    } catch (Exception e)
    {
```



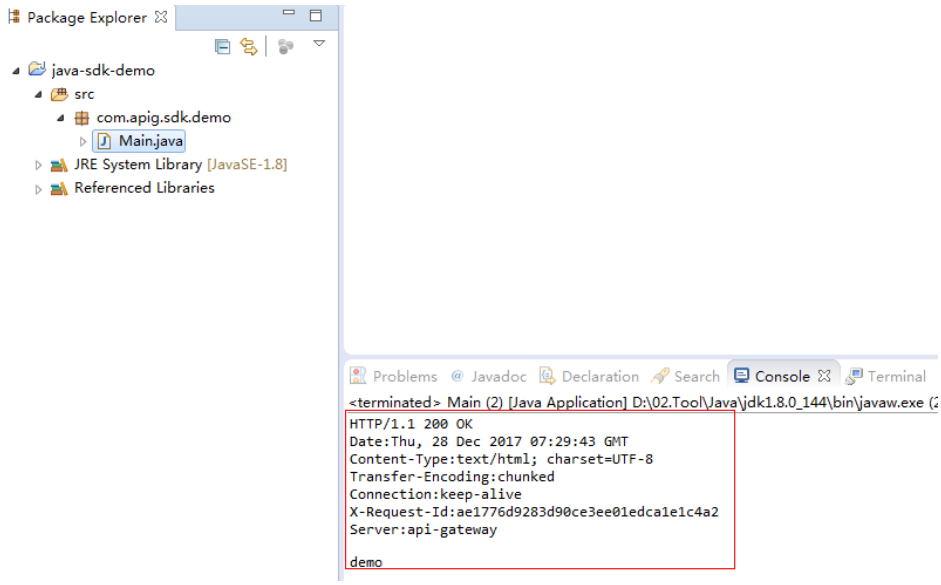
```
        e.printStackTrace();
    } finally
    {
        try
        {
            if (client != null)
            {
                client.close();
            }
        } catch (IOException e)
        {
            e.printStackTrace();
        }
    }
```

Step 4 Right-click **Main.java** and choose **Run As** > **Java Application**.

Run the project test code.



Step 5 On the **Console** tab page, view the running result.



----End

3.6.2 Go

This section uses IntelliJ IDEA as an example to describe how to integrate the Go SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

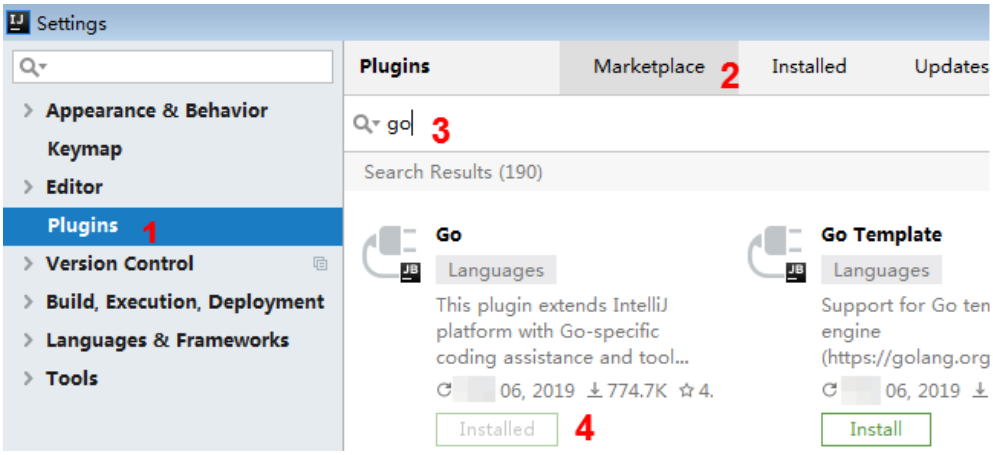
NOTE

The signing SDK is used for signing requests and not used for cloud service access. For the cloud service SDK, see **SDK Development Guide**.

Preparing the IDEA Development Environment

- Download IntelliJ IDEA from the **IntelliJ IDEA official website** and install it.
- Download the Go installation package from the **Go official website** and install it.
- To install the Go plug-in on IDEA, choose **File** > **Settings**.

Figure 3-8 Installing the Go plug-in on IDEA



Obtaining the SDK

Download the SDK at **https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/APIGW-go-sdk.zip**.

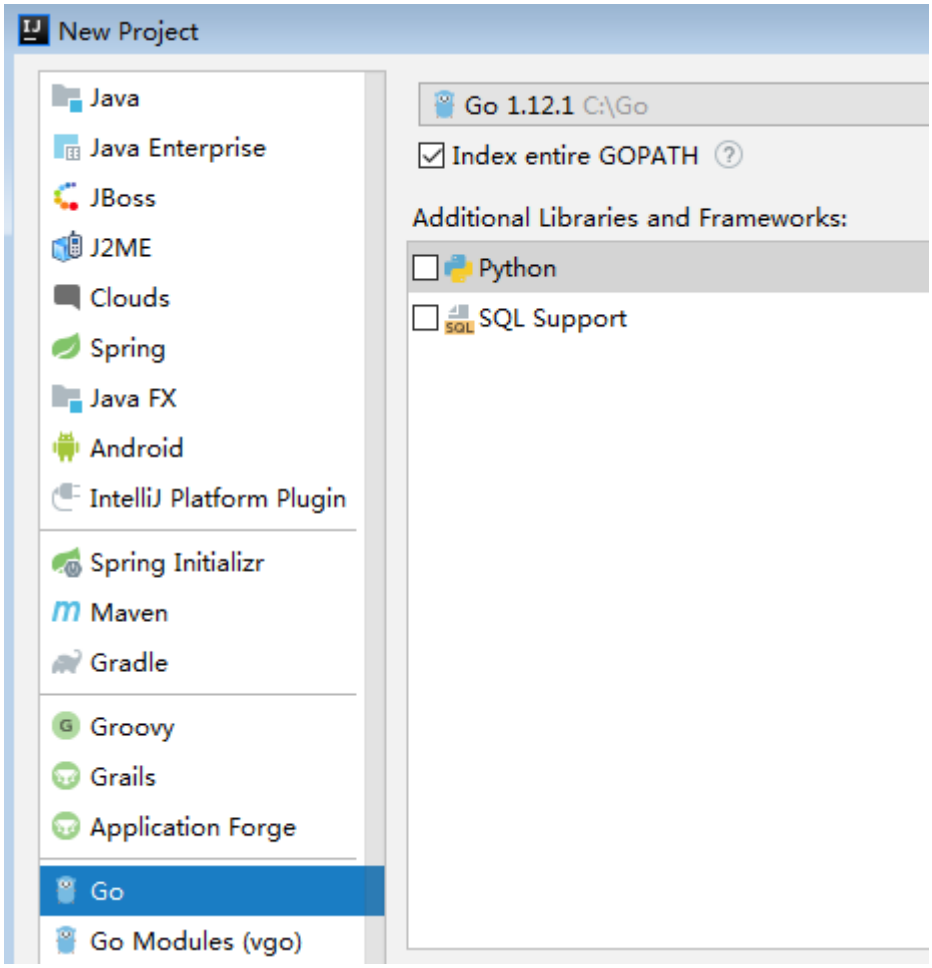
Develop your application using the SDK and sample code.

Name	Description
core\escape.go	Used for escaping special characters.
core\signer.go	Signing SDK
demo.go	Sample code

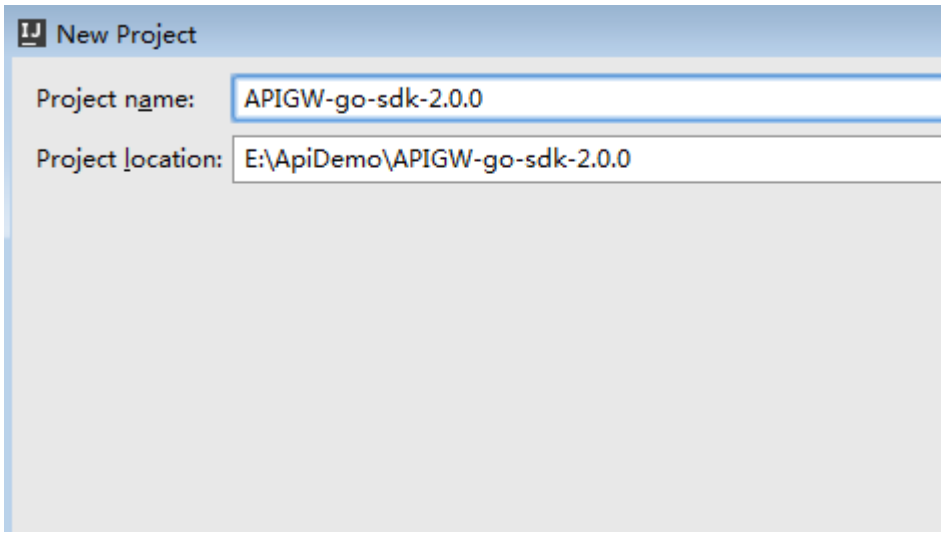
Importing the Sample Project on IDEA

Step 1 Start IDEA and choose **File** > **New** > **Project**.

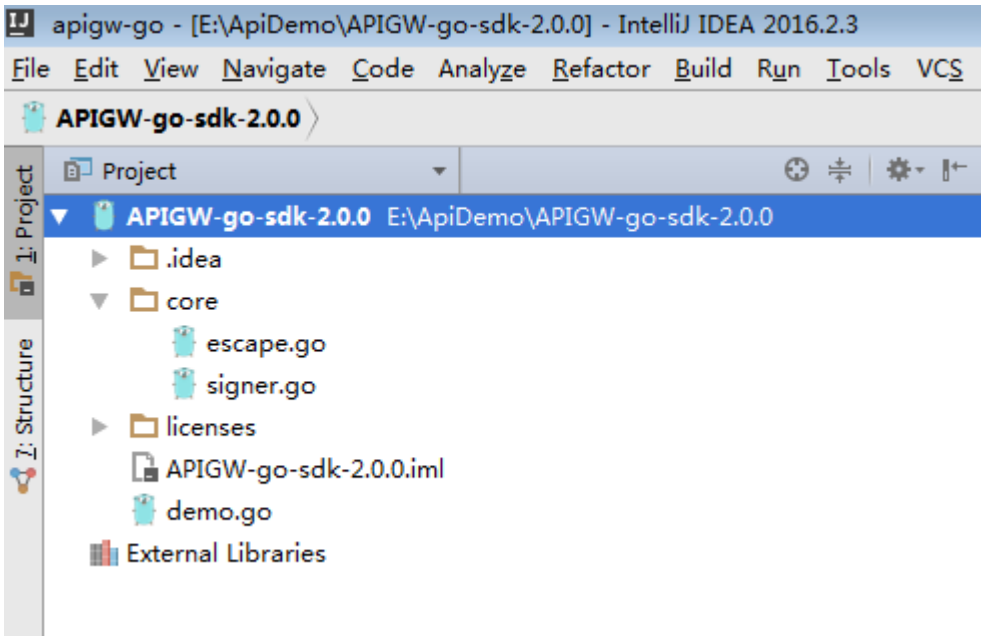
On the displayed **New Project** page, choose **Go** and click **Next**.



Step 2 Click **...**, select the directory where the SDK is decompressed, and click **Finish**.



Step 3 View the directory structure shown in the following figure.



----End

Request Signing and API Calling

Step 1 Import the Go SDK (signer.go) to the project.

```
import "./core"
```

Step 2 Generate a new signer and enter the AK and SK.

```
s := core.Signer{
        Key: "QTWAOY********KYUC",
        Secret: "MFyfvK41ba2giqM7**********KGpownRZlmVmHc",
}
```

Step 3 Generate a new request, and specify the domain name, method, request URI, and body.

```
//The following example shows how to set the request URL and parameters to query a VPC list.
//Add a body if you have specified the PUT or POST method. Special characters, such as the double
```



```
quotation mark ("), contained in the body must be escaped.
r, _ := http.NewRequest("GET", "https://service.region.example.com/v1/{project_id}/vpcs?a=1",
ioutil.NopCloser(bytes.NewBuffer([]byte(""))))
```

Step 4 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in multi-project scenarios or the **X-Domain-Id** header for a global service.

```
/Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for
invoking a project-level service.
r.Header.Add("X-Project-Id", "xxx")
```

Step 5 Execute the following function to add the **X-Sdk-Date** and **Authorization** headers for signing:

```
s.Sign(r)
```

Step 6 Access the API and view the access result.

```
resp, err := http.DefaultClient.Do(r)
body, err := ioutil.ReadAll(resp.Body)
```

----End

3.6.3 Python

This section uses IntelliJ IDEA as an example to describe how to integrate the Python SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

NOTE

The signing SDK is used for signing requests and not used for cloud service access. For the cloud service SDK, see **SDK Development Guide**.

Preparing the Environment

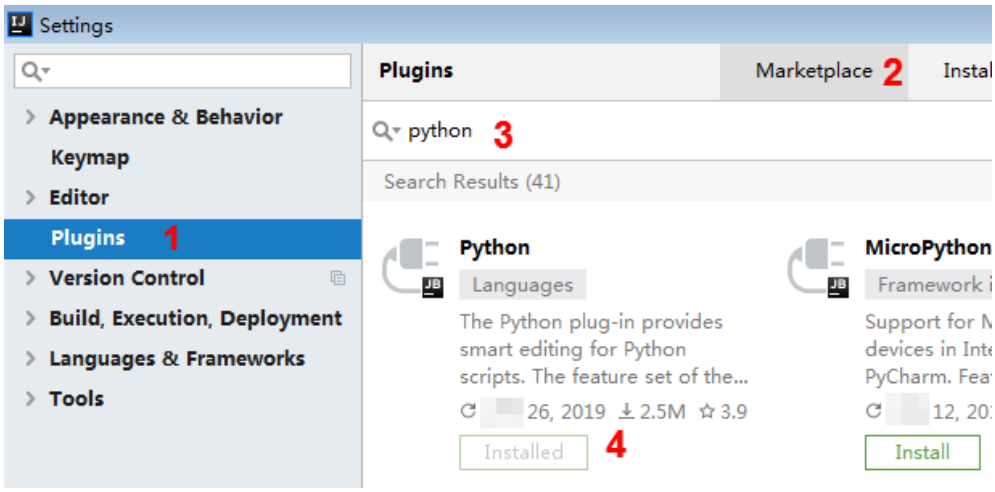
- Download IntelliJ IDEA from the **IntelliJ IDEA official website** and install it.
- Download the Python installation package (version 2.7.9 or later, or 3.x) from the **Python official website** and install it.

  After Python is installed, run the **pip** command to install the **requests** library.

  ```
  pip install requests
  ```

  NOTE

  If a certificate error occurs during the installation, download the **get-pip.py** file to upgrade the pip environment, and try again.
- Install the Python plug-in on IDEA.



Obtaining the SDK

Download the SDK at **https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/APIGW-python-sdk.zip**.

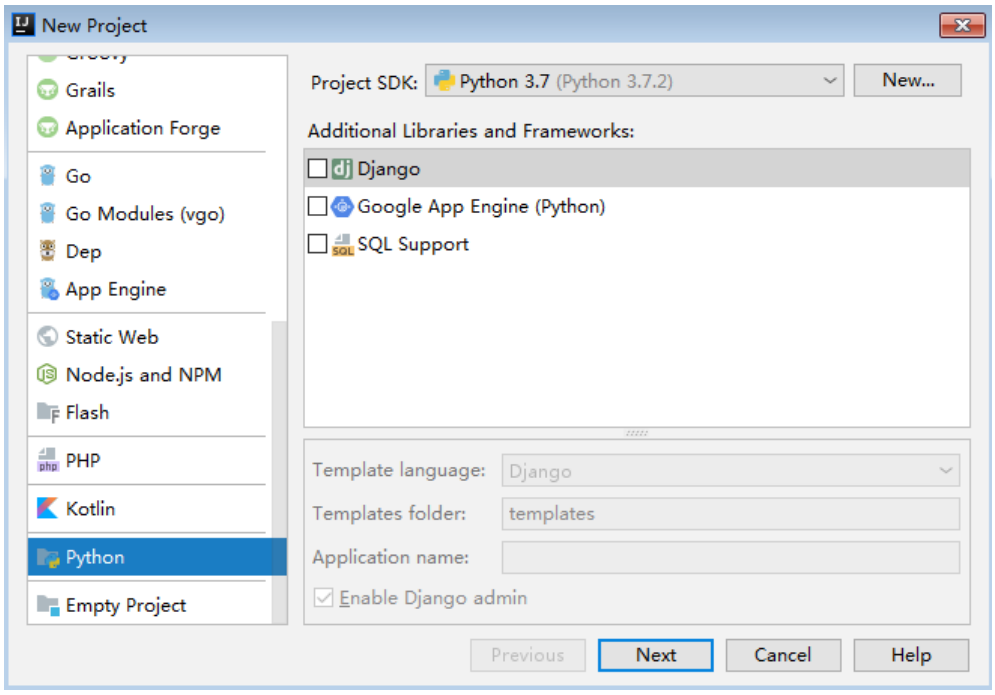
The following table shows the directory structure of the downloaded package.

Name	Description
apig_sdk__init__.py	SDK code
apig_sdk\signer.py	
main.py	Sample code
licenses\license-requests	Third-party license

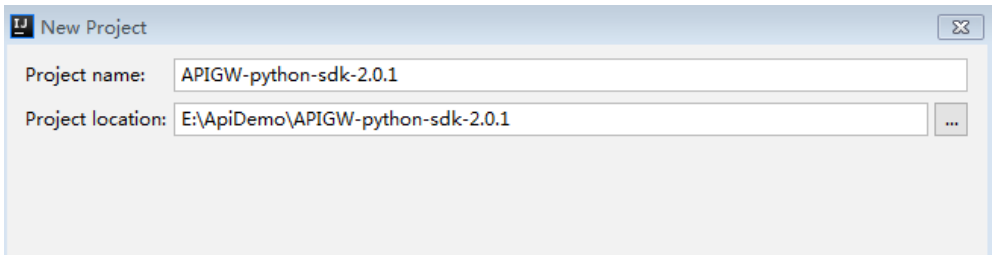
Importing the Sample Project

Step 1 Start IDEA and choose **File** > **New** > **Project**.

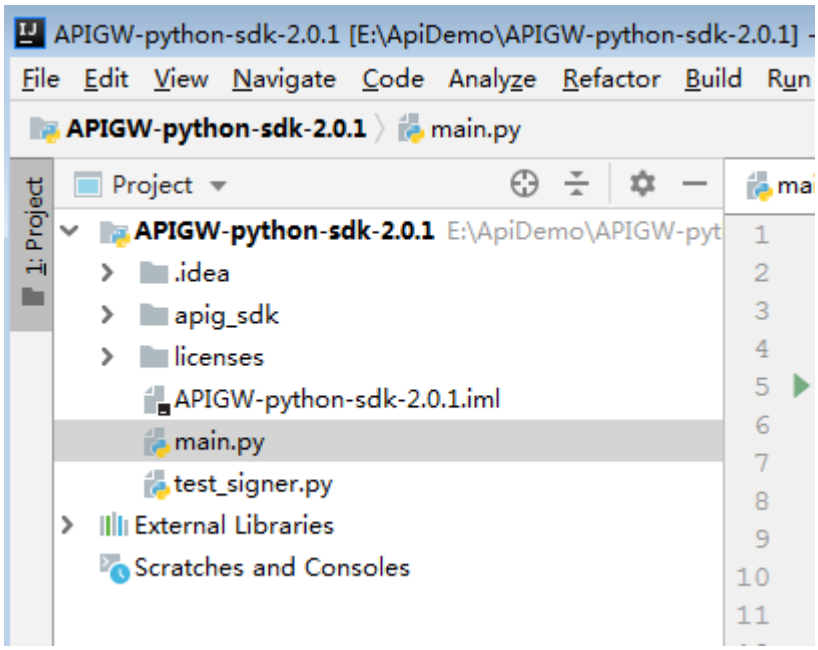
On the displayed **New Project** page, choose **Python** and click **Next**.



Step 2 Click **Next**. Click **...**, select the directory where the SDK is decompressed, and click **Finish**.



Step 3 View the directory structure shown in the following figure.



----End

Request Signing and API Calling

Step 1 Run the **pip** command to install the **requests** library.

```
pip install requests
```

Step 2 Import **apig_sdk** to the project.

```
from apig_sdk import signer
import requests
```

Step 3 Generate a new signer and enter the AK and SK.

```
sig = signer.Signer()
# Set the AK/SK to sign and authenticate the request.
sig.Key = "QTWAOY********VKYUC"
sig.Secret = "MFyfvK41ba2giqM7**********KGpownRZlmVmHc"
```

Step 4 Generate a new request, and specify the domain name, method, request URI, and body.

The following is an example request for querying VPCs. The HTTP method is GET, the domain name (endpoint) is **service.region.example.com**, and the API URI is **/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?limit=1**.

For details, see section "Querying VPCs" in the *VPC API Reference*.

```
# The following example shows how to set the request URL and parameters to query a VPC list.
r = signer.HttpRequest("GET", "https://{service}.region.example.com/
v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?limit=1")
# r.body = "{\"a\":1}"
```

Step 5 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in multi-project scenarios or the **X-Domain-Id** header for a global service. Separate multiple headers with commas.

```
r.headers = {"X-Project-Id": "xxx"}
```

Step 6 Execute the following function to add the **X-Sdk-Date** and **Authorization** headers for signing:

```
sig.Sign(r)
```


NOTE

- **X-Sdk-Date** is a request header parameter required for signing requests.
- The SDK automatically completes signing requests, and you do not need to know which header parameters are involved in the signing process.

Step 7 Access the API and view the access result.

```
resp = requests.request(r.method, r.scheme + "://" + r.host + r.uri, headers=r.headers, data=r.body)
print(resp.status_code, resp.reason)
print(resp.content)
```

----End

3.6.4 C#

This section uses Visual Studio as an example to describe how to integrate the C# SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

NOTE

The signing SDK is used for signing requests and not used for cloud service access. For the cloud service SDK, see **SDK Development Guide**.

Preparing the Environment

Download Visual Studio from the **Visual Studio official website** and install it.

Obtaining the SDK

Download the SDK at **https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/APIGW-csharp-sdk.zip**.

The following table shows the directory structure of the downloaded package.

Name	Description
apigateway-signature\Signer.cs	SDK code
apigateway-signature\HttpEncoder.cs	
sdk-request\Program.cs	Sample code for signing requests
csharp.sln	Project file
licenses\license-referencesource	Third-party license

Opening the Sample Project

Double-click **csharp.sln** in the SDK package to open the project. **apigateway-signature** is a shared library that implements the signature algorithm. It can be used in .Net Framework and .Net Core projects. The **sdk-request** project is used as an example.

Request Signing and API Calling

Step 1 Import the SDK to the project.

```
using APIGATEWAY_SDK;
```

Step 2 Generate a new signer and enter the AK and SK.

```
Signer signer = new Signer();
signer.Key = "QTWAOYT**********VKYUC";
signer.Secret = "MFyfvK41ba2giqM7**********KGpownRZlmVmHc";
```

Step 3 Generate a new request, and specify the domain name, method, request URI, and body.

```
//The following example shows how to set the request URL and parameters to query a VPC list.
HttpRequest r = new HttpRequest("GET", new Uri("https://{service}.region.example.com/
v1/77b6a44cba5**********9a8ff44fd/vpcs?limit=1"));
//Add a body if you have specified the PUT or POST method. Special characters, such as the double
quotation mark ("), contained in the body must be escaped.
r.body = "";
```

Step 4 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in multi-project scenarios or the **X-Domain-Id** header for a global service.

```
//Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for
invoking a project-level service.
r.headers.Add("X-Project-Id", "xxx");
```

Step 5 Execute the following function to generate **HttpWebRequest**, and add the **X-Sdk-Date** and **Authorization** headers for signing the request:

If you use HttpClient, you can obtain header information from the request. For details about headers, see **AK/SK Signing and Authentication Algorithm**.

```
HttpWebRequest req = signer.Sign(r);
```

Step 6 Access the API and view the access result.

```
var writer = new StreamWriter(req.GetRequestStream());
writer.Write(r.body);
writer.Flush();
HttpWebResponse resp = (HttpWebResponse)req.GetResponse();
var reader = newStreamReader(resp.GetResponseStream());
Console.WriteLine(reader.ReadToEnd());
```

----End

3.6.5 JavaScript

This section uses IntelliJ IDEA as an example to describe how to integrate the JavaScript SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

The descriptions in this section are provided based on the Node.js environment.

NOTE

The signing SDK is used for signing requests and not used for cloud service access. For the cloud service SDK, see **SDK Development Guide**.

Preparing the Environment

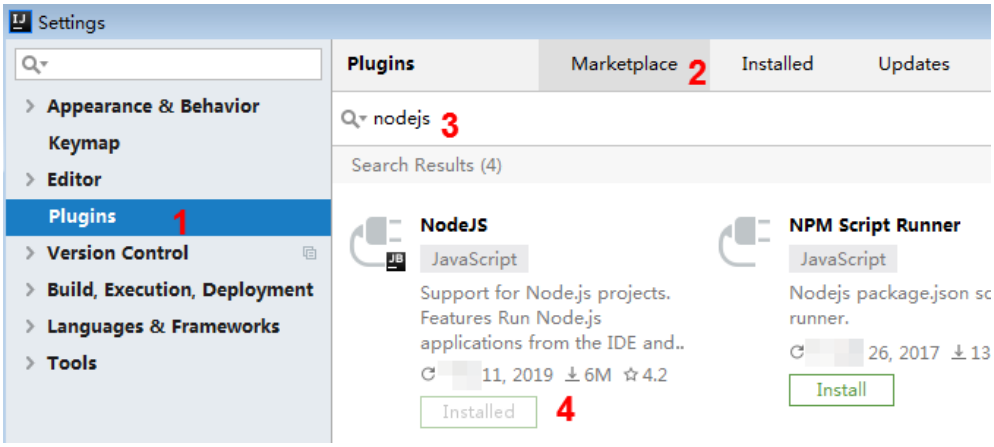
- Download IntelliJ IDEA from the **IntelliJ IDEA official website** and install it.

- Download the Node.js installation package from the **Node.js official website** and install it.

  After Node.js is installed, run the **npm** command to install the **moment** and **moment-timezone** modules.

  ```
  npm install moment --save
  npm install moment-timezone --save
  ```

- Install the Node.js plug-in on IDEA.



Obtaining the SDK

Download the SDK at **https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/APIGW-javascript-sdk.zip**.

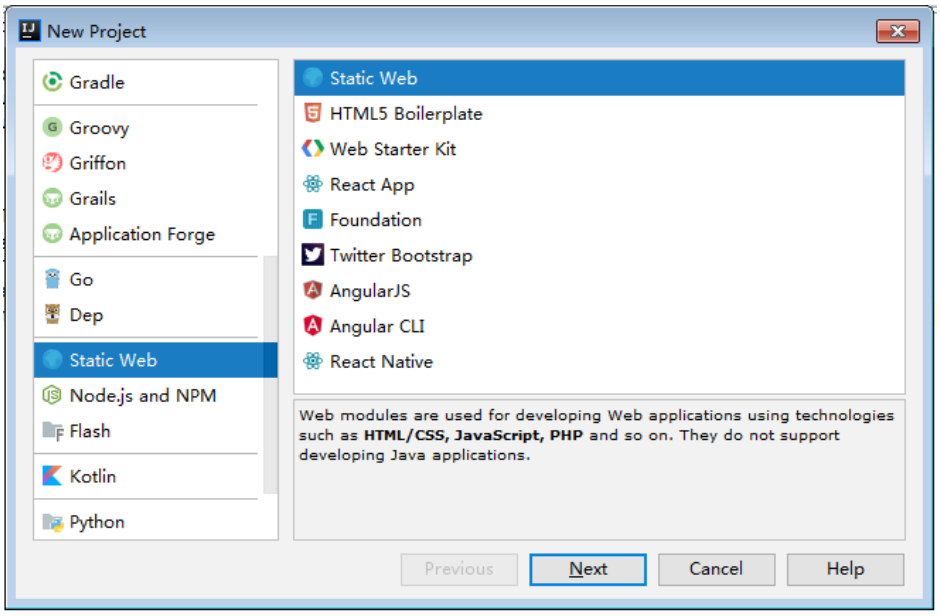
Decompress the downloaded package to the current folder. The following table shows the directory structure.

Name	Description
signer.js	SDK code
node_demo.js	Node.js sample code
test.js	Test case
licenses\license-crypto-js	Third-party licenses
licenses\license-node	

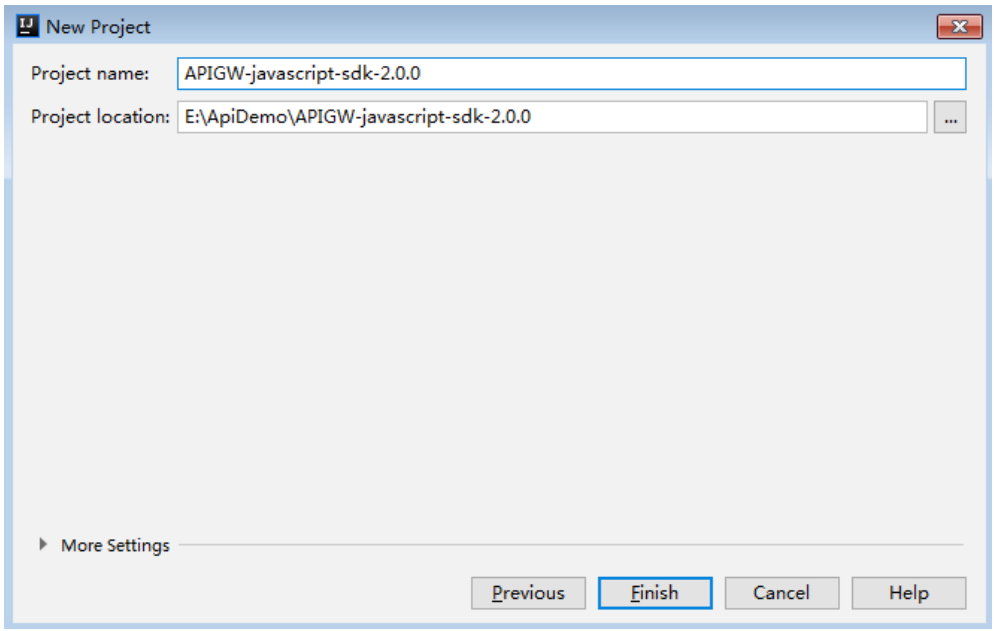
Creating a Project

Step 1 Start IDEA and choose **File** > **New** > **Project**.

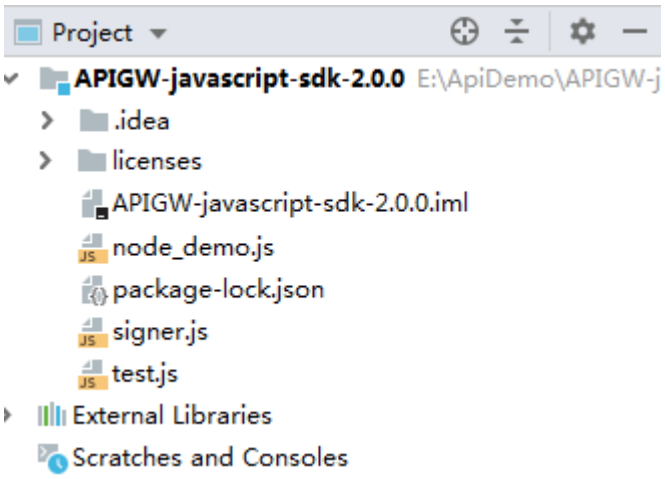
In the **New Project** dialog box, choose **Static Web** and click **Next**.



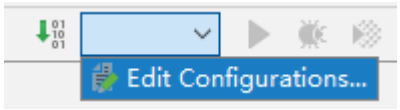
Step 2 Click **...**, select the directory where the SDK is decompressed, and click **Finish**.



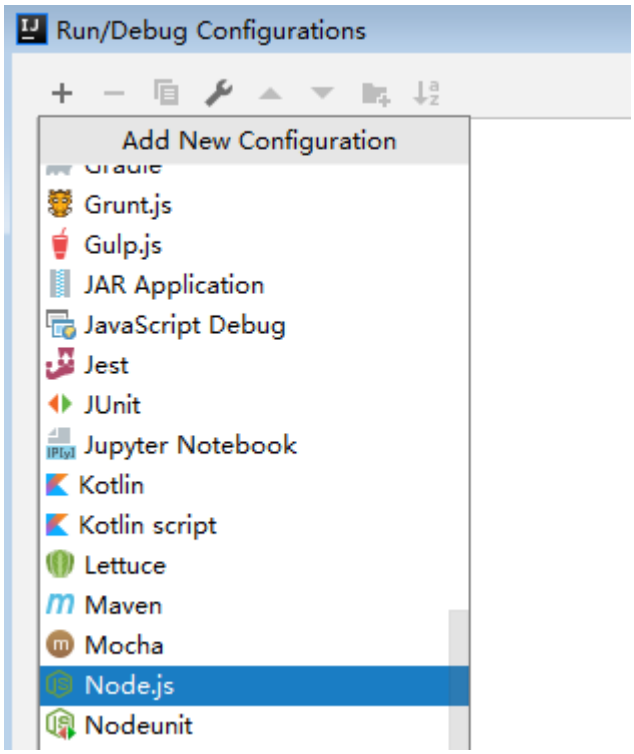
Step 3 View the directory structure shown in the following figure.



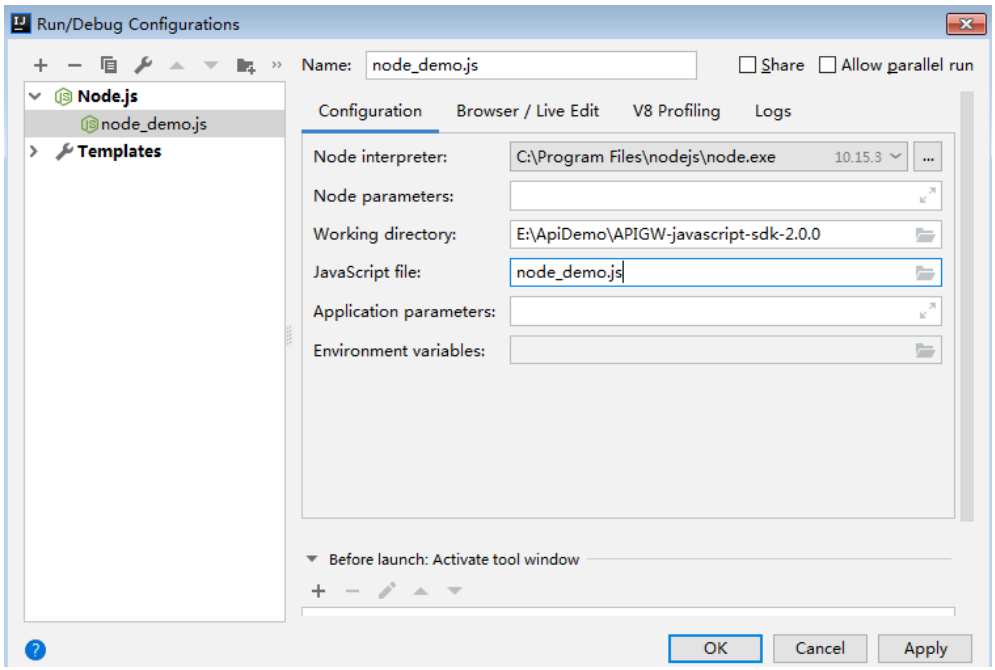
Step 4 In the upper right corner of the IDEA window, click **Edit Configurations** or **Add Configurations**.



Step 5 Click **+** and select **Node.js**.



Step 6 Set **JavaScript file** to **node_demo.js** and click **OK**.



----End

Calling APIs (Node.js)

Step 1 Run the **npm** command to install the **moment** and **moment-timezone** modules.

```
npm install moment --save
npm install moment-timezone --save
```

Step 2 Import **signer.js** to your project.

```
var signer = require('./signer')
var http = require('http')
```

Step 3 Generate a new signer and enter the key and secret.

```
var sig = new signer.Signer()
sig.Key = "QTWAOYT******QVKYUC"
sig.Secret = "MFyfvK41ba2giqM7**********KGpownRZlmVmHc"
```

Step 4 Generate a new request, and specify the domain name, method, request URI, and body.

```
//The following example shows how to set the request URL and parameters to query a VPC list.
var r = new signer.HttpRequest("GET", "service.region.example.com/
v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?limie=1");

//Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
r.body = '';
```

Step 5 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in multi-project scenarios or the **X-Domain-Id** header for a global service.

```
//Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for invoking a project-level service.
r.headers = {"X-Project-Id", "xxx"};
```

Step 6 Execute the following function to generate HTTPS request parameters, and add the **X-Sdk-Date** and **Authorization** headers for signing the request:

```
var opt = sig.Sign(r)
```


Step 7 Access the API and view the access result.

```
var req = https.request(opt, function(res){
        console.log(res.statusCode)
        res.on("data",    function(chunk){
        console.log(chunk.toString())
    })
})
req.on("error",function(err){
    console.log(err.message)
})
req.write(r.body)
req.end()
```

----End

3.6.6 PHP

This section uses IntelliJ IDEA as an example to describe how to integrate the PHP SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

NOTE

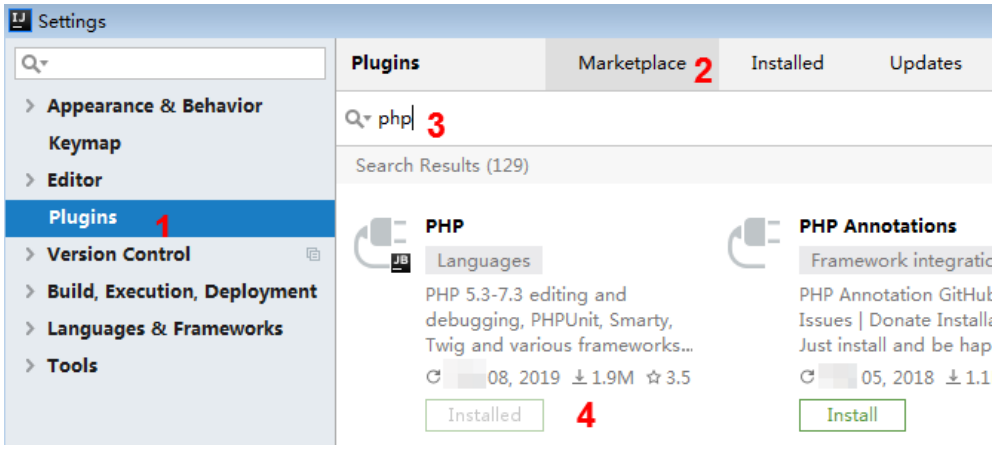
The signing SDK is used for signing requests and not used for cloud service access. For the cloud service SDK, see **SDK Development Guide**.

Preparing the Environment

- Download IntelliJ IDEA from the **IntelliJ IDEA official website** and install it.
- Download the PHP installation package from the **PHP official website** and install it.
- Copy the **php.ini-production** file from the PHP installation directory to the **C:\windows** directory, rename the file as **php.ini**, and then add the following lines to the file:

```
extension_dir = "{PHP installation directory}\ext"
extension=openssl
extension=curl
```

- Install the PHP plug-in on IDEA.



Obtaining the SDK

Download the SDK at **https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/APIGW-php-sdk.zip**.

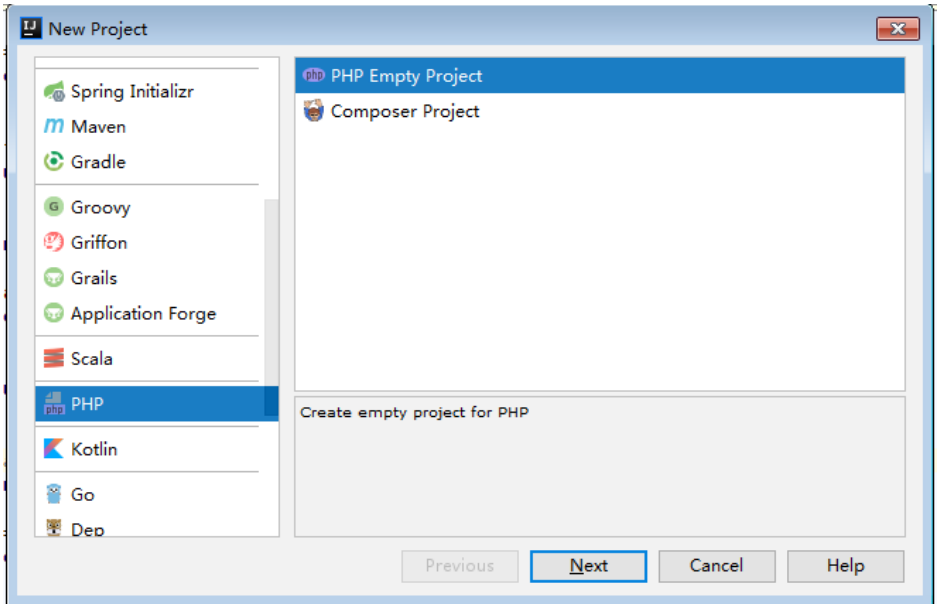
Decompress the downloaded package to the current folder. The following table shows the directory structure.

Name	Description
signer.php	SDK code
index.php	Sample code

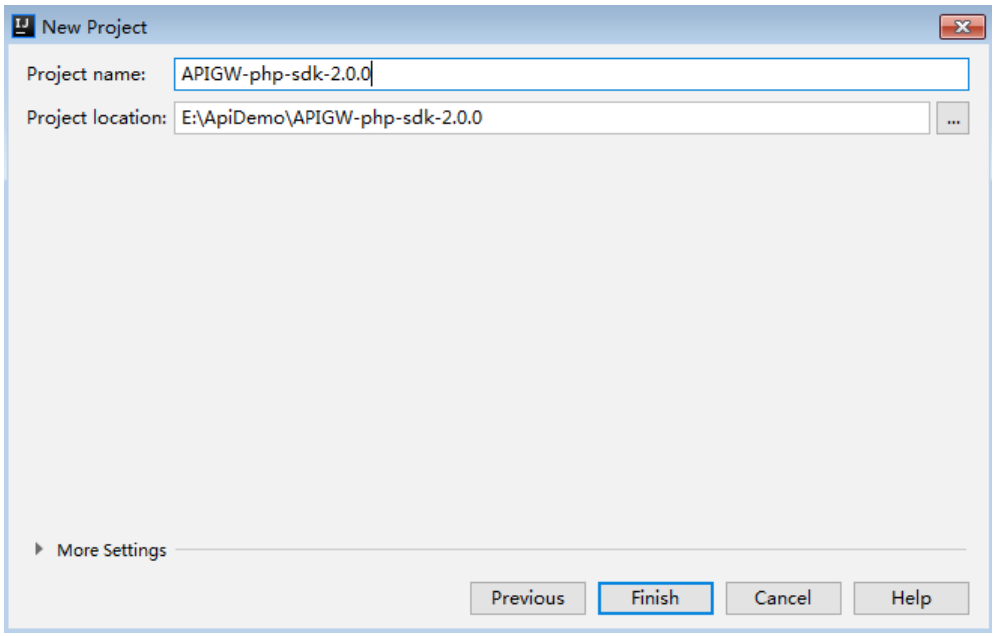
Creating a Project

Step 1 Start IDEA and choose **File** > **New** > **Project**.

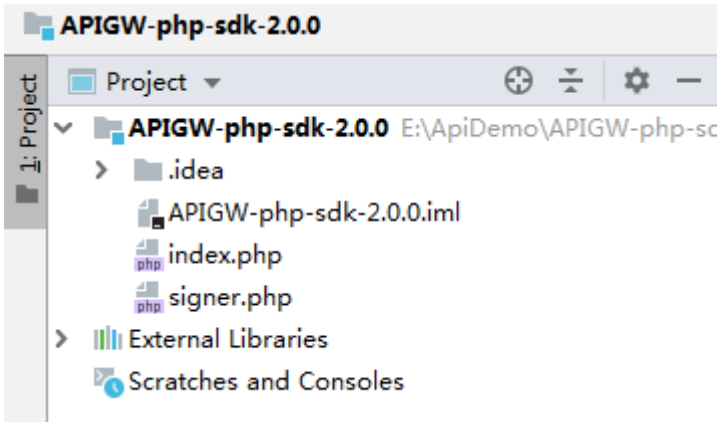
On the displayed **New Project** page, choose **PHP** and click **Next**.



Step 2 Click **...**, select the directory where the SDK is decompressed, and click **Finish**.



Step 3 View the directory structure shown in the following figure.



----End

Request Signing and API Calling

Step 1 Import the PHP SDK to your code.

```
require 'signer.php';
```

Step 2 Generate a new signer and enter the AK and SK.

```
$signer = new Signer();
$signer->Key = 'QTWAOY*********QVKYUC';
$signer->Secret = "MFyfvK41ba2giqM7**********KGpownRZlmVmHc";
```

Step 3 Generate a new request, and specify the domain name, method, request URI, and body.

```
//The following example shows how to set the request URL and parameters to query a VPC list.
$req = new Request('GET', 'https://service.region.example.com/v1/{project_id}/vpcs?limit=1');
//Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
$req->body = '';
```


Step 4 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in multi-project scenarios or the **X-Domain-Id** header for a global service.

```
//Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for invoking a project-level service.
$req->headers = array(
   'X-Project-Id' => 'xxx',
);
```

Step 5 Execute the following function to generate a **$curl** context variable.

```
$curl = $signer->Sign($req);
```

Step 6 Access the API and view the access result.

```
$response = curl_exec($curl);
echo curl_getinfo($curl, CURLINFO_HTTP_CODE);
echo $response;
curl_close($curl);
```

----End

3.6.7 C++

Preparing the Environment

This section uses Linux Ubuntu as an example. Before calling APIs, install the required SSL tools.

1. Install the OpenSSL library.
   ```
   apt-get install libssl-dev
   ```
2. Install the curl library.
   ```
   apt-get install libcurl4-openssl-dev
   ```

Obtaining the SDK

NOTE

The signing SDK is used for signing requests and not used for cloud service access. For the cloud service SDK, see **SDK Development Guide**.

Download the SDK at **https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/APIGW-cpp-sdk.zip**.

Decompress the downloaded package to the current folder. The following table shows the directory structure.

Name	Description
hasher.cpp	SDK code
hasher.h	
header.h	
RequestParams.cpp	
RequestParams.h	
signer.cpp	

Name	Description
signer.h	
constants.h	
Makefile	**Makefile** file
main.cpp	Sample code

Request Signing and API Calling

Step 1 Add the following references to **main.cpp**:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curl/curl.h>
#include "signer.h"
```

Step 2 Generate a new signer and enter the AK and SK.

```
//Set the AK/SK to sign and authenticate the request.
Signer signer("QTWAOYT********VKYUC", "MFyfvK41ba2giqM7**********KGpownRZlmVmHc");
```

Step 3 Generate a new **RequestParams** request, and specify the method, domain name, request URI, query strings, and request body.

```
//Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
//Set a request URL.
//Set parameters for the request URL.
//Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
RequestParams* request = new RequestParams("GET", "service.region.example.com", "/v1/{project_id}/vpcs",
    "limit=2", "");
```

Step 4 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in multi-project scenarios or the **X-Domain-Id** header for a global service.

```
//Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for invoking a project-level service.
request->addHeader("X-Project-Id", "xxx");
```

Step 5 Execute the following function to add the generated headers as request variables.

```
signer.createSignature(request);
```

Step 6 Use the curl library to access the API and view the access result.

```
static size_t
WriteMemoryCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
  size_t realsize = size * nmemb;
  struct MemoryStruct *mem = (struct MemoryStruct *)userp;

  mem->memory = (char*)realloc(mem->memory, mem->size + realsize + 1);
  if (mem->memory == NULL) {
    /* out of memory! */
    printf("not enough memory (realloc returned NULL)\n");
    return 0;
  }

  memcpy(&(mem->memory[mem->size]), contents, realsize);
  mem->size += realsize;
  mem->memory[mem->size] = 0;
```



```
    return realsize;
}

//send http request using curl library
int perform_request(RequestParams* request)
{
    CURL *curl;
    CURLcode res;
    struct MemoryStruct resp_header;
    resp_header.memory = (char*)malloc(1);
    resp_header.size = 0;
    struct MemoryStruct resp_body;
    resp_body.memory = (char*)malloc(1);
    resp_body.size = 0;

    curl_global_init(CURL_GLOBAL_ALL);
    curl = curl_easy_init();

    curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, request->getMethod().c_str());
    std::string url = "http://" + request->getHost() + request->getUri() + "?" + request->getQueryParams();
    curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
    struct curl_slist *chunk = NULL;
    std::set<Header>::iterator it;
    for (auto header : *request->getHeaders()) {
        std::string headerEntry = header.getKey() + ": " + header.getValue();
        printf("%s\n", headerEntry.c_str());
        chunk = curl_slist_append(chunk, headerEntry.c_str());
    }
    printf("--------------\n");
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, chunk);
    curl_easy_setopt(curl, CURLOPT_COPYPOSTFIELDS, request->getPayload().c_str());
    curl_easy_setopt(curl, CURLOPT_NOBODY, 0L);
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteMemoryCallback);
    curl_easy_setopt(curl, CURLOPT_HEADERDATA, (void *)&resp_header);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, (void *)&resp_body);
    //curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
    res = curl_easy_perform(curl);
    if (res != CURLE_OK) {
        fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));
    }
    else {
        long status;
        curl_easy_getinfo(curl, CURLINFO_HTTP_CODE, &status);
        printf("status %d\n", status);
        printf(resp_header.memory);
        printf(resp_body.memory);
    }
    free(resp_header.memory);
    free(resp_body.memory);
    curl_easy_cleanup(curl);

    curl_global_cleanup();

    return 0;
}
```

Step 7 Run the **make** command to obtain a **main** executable file, execute the file, and then view the execution result.

----End

3.6.8 C

Preparing the Environment

This section uses Linux Ubuntu as an example. Before calling APIs, install the required SSL tools.

1. Install the OpenSSL library.
   ```
   apt-get install libssl-dev
   ```
2. Install the curl library.
   ```
   apt-get install libcurl4-openssl-dev
   ```

Obtaining the SDK

NOTE

The signing SDK is used for signing requests and not used for cloud service access. For the cloud service SDK, see **SDK Development Guide**.

Download the SDK at **https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/APIGW-c-sdk.zip**.

Decompress the downloaded package to the current folder. The following table shows the directory structure.

Name	Description
signer_common.c	SDK code
signer_common.h	
signer.c	
signer.h	
Makefile	**Makefile** file
main.c	Sample code

Request Signing and API Calling

Step 1 Add the following references to **main.c**:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curl/curl.h>
#include "signer.h"
```

Step 2 Generate a sig_params_t variable, and enter the AK and SK.

```
sig_params_t params;
sig_params_init(&params);
sig_str_t ak = sig_str("QTWAOY******QVKYUC");
sig_str_t sk = sig_str("MFyfvK41ba2giqM7**********KGpownRZlmVmHc");
params.key = ak;
params.secret = sk;
```

Step 3 Specify the method, domain name, request URI, query strings, and request body.

```
sig_str_t host = sig_str("service.region.example.com");
sig_str_t method = sig_str("GET");
sig_str_t uri = sig_str("/v1/{project_id}/vpcs");
sig_str_t query_str = sig_str("limit=2");
sig_str_t payload = sig_str("");
params.host = host;
params.method = method;
params.uri = uri;
params.query_str = query_str;
params.payload = payload;
```


Step 4 Add header parameters or other headers required for other purposes. For example, add the **X-Project-Id** header in multi-project scenarios or the **X-Domain-Id** header for a global service.

```
//Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for invoking a project-level service.
sig_headers_add(&params.headers, "X-Project-Id", "xxx");
```

Step 5 Execute the following function to add the generated headers as request variables.

```
sig_sign(&params);
```

Step 6 Use the curl library to access the API and view the access result.

```
static size_t
WriteMemoryCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
    size_t realsize = size * nmemb;
    struct MemoryStruct *mem = (struct MemoryStruct *)userp;

    mem->memory = (char*)realloc(mem->memory, mem->size + realsize + 1);
    if (mem->memory == NULL) {
        /* out of memory! */
        printf("not enough memory (realloc returned NULL)\n");
        return 0;
    }

    memcpy(&(mem->memory[mem->size]), contents, realsize);
    mem->size += realsize;
    mem->memory[mem->size] = 0;

    return realsize;
}

//send http request using curl library
int perform_request(RequestParams* request)
{
    CURL *curl;
    CURLcode res;
    struct MemoryStruct resp_header;
    resp_header.memory = malloc(1);
    resp_header.size = 0;
    struct MemoryStruct resp_body;
    resp_body.memory = malloc(1);
    resp_body.size = 0;

    curl_global_init(CURL_GLOBAL_ALL);
    curl = curl_easy_init();

    curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, params.method.data);
    char url[1024];
    sig_snprintf(url, 1024, "http://%V%V?%V", &params.host, &params.uri, &params.query_str);
    curl_easy_setopt(curl, CURLOPT_URL, url);
    struct curl_slist *chunk = NULL;
    for (int i = 0; i < params.headers.len; i++) {
        char header[1024];
        sig_snprintf(header, 1024, "%V: %V", &params.headers.data[i].name, &params.headers.data[i].value);
        printf("%s\n", header);
        chunk = curl_slist_append(chunk, header);
    }
    printf("-------------\n");
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, chunk);
    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, params.payload.data);
    curl_easy_setopt(curl, CURLOPT_NOBODY, 0L);
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteMemoryCallback);
    curl_easy_setopt(curl, CURLOPT_HEADERDATA, (void *)&resp_header);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, (void *)&resp_body);
    //curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
    res = curl_easy_perform(curl);
    if (res != CURLE_OK) {
        fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));
```



```
    }
    else {
        long status;
        curl_easy_getinfo(curl, CURLINFO_HTTP_CODE, &status);
        printf("status %d\n", status);
        printf(resp_header.memory);
        printf(resp_body.memory);
    }
    free(resp_header.memory);
    free(resp_body.memory);
    curl_easy_cleanup(curl);

    curl_global_cleanup();

    //free signature params
    sig_params_free(&params);
    return 0;
}
```

Step 7 Run the **make** command to obtain a **main** executable file, execute the file, and then view the execution result.

----End

3.6.9 Android

This section uses Android Studio as an example to describe how to integrate the Android SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

Preparing the Environment

Download Android Studio at the **Android Studio official website** and install it.

Obtaining the SDK

NOTE

The signing SDK is used for signing requests and not used for cloud service access. For the cloud service SDK, see **SDK Development Guide**.

Download the SDK at **https://obs.cn-north-1.myhuaweicloud.com/apig-sdk/APIGW-android-sdk.zip**.

The following table shows the directory structure of the downloaded package.

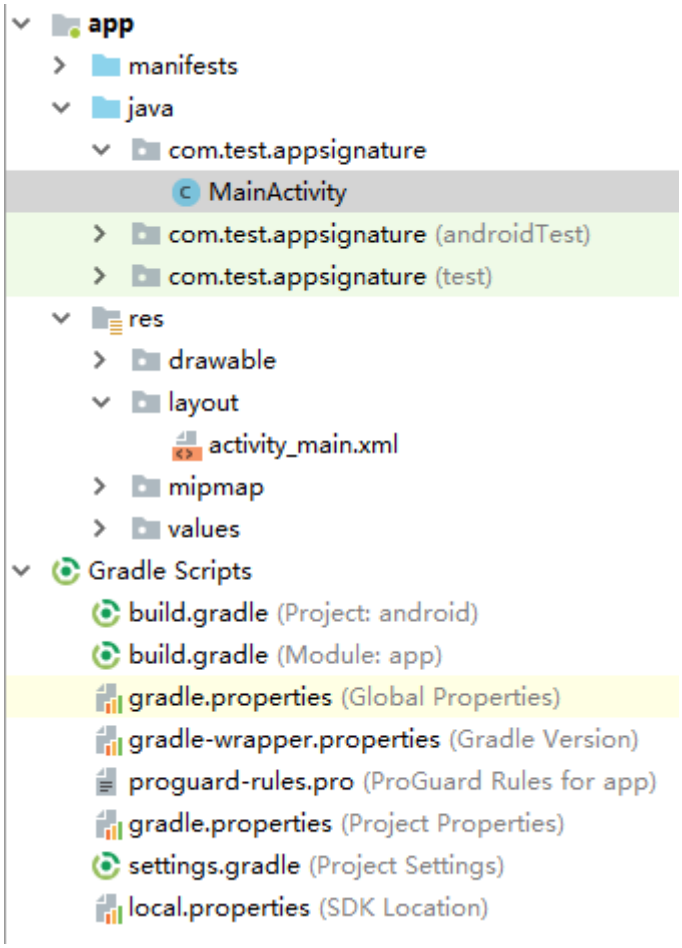
Name	Description
app\	Android project code
build.gradle	Gradle configuration files
gradle.properties	
settings.gradle	

Opening the Sample Project

Step 1 Start Android Studio and choose **File** > **Open**.

Select the directory where the SDK is decompressed.

Step 2 View the directory structure of the project shown in the following figure.



----End

Request Signing and API Calling

Step 1 Add required JAR files to the **app/libs** directory of the Android project. The following JAR files must be included:

- java-sdk-core-x.x.x.jar
- commons-logging-1.2.jar
- joda-time-2.9.9.jar

Step 2 Add dependencies of the **okhttp** library to the **build.gradle** file.

Add **implementation 'com.squareup.okhttp3:okhttp:3.11.0'** in the **dependencies** field of the **build.gradle** file.

```
dependencies {
    ...
    ...
    implementation 'com.squareup.okhttp3:okhttp:3.11.0'
}
```


Step 3 Create a request, enter the AK and SK, and specify the domain name, method, request URI, and body.

```
Request request = new Request();
try {
        request.setKey("QTWAOYTT*******KYUC");
        request.setSecrect("MFyfvK41ba2giqM7**********KGpownRZlmVmHc");
        request.setMethod("GET");
        request.setUrl("https://service.region.example.com3/v1/{project_id}/vpcs");
        request.addQueryStringParam("name", "value");
        request.addHeader("Content-Type", "text/plain");
        //request.setBody("demo");
} catch (Exception e) {
    e.printStackTrace();
    return;
}
```

Step 4 Sign the request to generate an **okhttp3.Request** object for API access.

```
okhttp3.Request signedRequest = Client.signOkhttp(request);
OkHttpClient client = new OkHttpClient.Builder().build();
Response response = client.newCall(signedRequest).execute();
```

----End

4 Error Codes

If an error code starting with **APIGW** is displayed when you call an API, seek solutions in the following table.

Table 4-1 Error codes

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0101	The API does not exist or has not been published in the environment.	404	The API does not exist or has not been published in the environment.	Check whether the domain name, method, and path are consistent with those of the registered API. Check whether the API has been published. If it has been published in a non-production environment, check whether the X-Stage header in the request is the environment name.
APIGW.0101	The API does not exist	404	The request method does not exist.	Check whether the request method is the same as the method specified for the API.
APIGW.0103	The backend does not exist.	404	The backend service is not found.	Contact technical support.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0104	The plug-ins do not exist.	400	No plugin configurations are found.	Contact technical support.
APIGW.0105	The backend configurations do not exist.	400	No backend configurations are found.	Contact technical support.
APIGW.0106	Orchestration error.	400	Orchestration error.	Check whether the frontend and backend parameters are properly set for the API.
APIGW.0201	API request error.	400	Invalid request parameters.	Set valid request parameters.
APIGW.0201	Request entity too large.	413	The request body exceeds 12 MB.	Reduce the size of the request body.
APIGW.0201	Request URI too large.	414	The request URI is too large.	Reduce the size of the request URI.
APIGW.0201	Request headers too large.	494	The request headers are too large.	Reduce the size of the request headers.
APIGW.0201	Backend unavailable.	502	The backend service is currently unavailable.	Check whether the backend address configured for the API is accessible.
APIGW.0201	Backend timeout.	504	The backend service timed out.	Increase the timeout duration of the backend service or shorten the processing time.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0301	Incorrect IAM authentication information.	401	The IAM authentication information is incorrect.	Check whether the token information (such as the project ID and name) is correct and whether the time difference between the client and server is less than 15 minutes.
APIGW.0302	The IAM user is not authorized to access the API.	403	The IAM user is not allowed to access the API.	Check whether access of the user has been restricted by the blacklist or whitelist.
APIGW.0303	Incorrect app authentication information.	401	The app authentication information is incorrect.	Check whether the request method, path, query parameters, and request body are consistent with those used for signing; check whether the date and time on the client are correct.
APIGW.0304	The app is not authorized to access the API.	403	The app is not allowed to access the API.	Check whether the app has been authorized to access the API.
APIGW.0305	Incorrect authentication information.	401	The authentication information is incorrect.	Check whether the authentication information is correct.
APIGW.0306	API access denied.	403	Access to the API is not allowed.	Check whether you have been authorized to access the API.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0307	The token must be updated.	401	The token needs to be updated.	Obtain the token from IAM again. The token may be invalid because APIs of different regions are called. You are advised to check the API URL.
APIGW.0308	The throttling threshold has been reached.	429	The request throttling threshold is reached.	Try again after the throttling resumes. By default, each API can be accessed for a maximum of 200 times per second. The rate limits of cloud service APIs cannot be adjusted. Please try again after the throttling resumes. To adjust the rate limit of an API you have created in API Gateway, contact technical support by submitting a service ticket.
APIGW.0310	The project is unavailable.	403	The project is currently unavailable.	Select another project and try again.
APIGW.0311	Incorrect debugging authentication information.	401	The debugging authentication information is incorrect.	Contact technical support.
APIGW.0401	Unknown client IP address.	403	The client IP address cannot be identified.	Contact technical support.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0402	The IP address is not authorized to access the API.	403	The IP address is not allowed to access the API.	Check whether access of the IP address has been restricted by the blacklist or whitelist.
APIGW.0404	Access to the backend IP address has been denied.	403	The backend IP address cannot be accessed.	Check whether the backend IP address or the IP address corresponding to the backend domain name is accessible.
APIGW.0501	The app quota has been used up.	405	The app quota has been reached.	Increase the app quota.
APIGW.0502	The app has been frozen.	405	The app has been frozen.	Check whether your account balance is sufficient.
APIGW.0601	Internal server error.	500	Internal error.	Contact technical support.
APIGW.0602	Bad request.	400	Invalid request.	Check whether the request is valid.
APIGW.0605	Domain name resolution failed.	500	Domain name resolution failed.	Check whether the domain name is correct and has been bound to a correct backend address.
APIGW.0606	Failed to load the API configurations.	500	API configurations are not loaded.	Contact technical support.
APIGW.0607	The following protocol is supported: {xxx}	400	The protocol is not supported. Only *xxx* is allowed. *xxx* is subject to the actual response.	Use HTTP or HTTPS to access the API.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0608	Failed to obtain the admin token.	500	The administrator account information cannot be obtained.	Contact technical support.
APIGW.0609	The VPC backend does not exist.	500	The VPC backend service cannot be found.	Contact technical support.
APIGW.0610	No backend available.	502	No backend services are available.	Check whether all backends are available.
APIGW.0611	The backend port does not exist.	500	The backend port is not found.	Contact technical support.
APIGW.0612	An API cannot call itself.	500	An API cannot call itself.	Modify the backend configurations, and ensure that the number of layers the API is recursively called does not exceed 10.
APIGW.0613	The IAM service is currently unavailable.	503	IAM is currently unavailable.	Contact technical support.
APIGW.0705	Backend signature calculation failed.	500	Backend signature calculation failed.	Contact technical support.
APIGW.0801	The service is unavailable in the currently selected region.	403	The service is inaccessible in the current region.	Check whether the service supports cross-region access.
APIGW.0802	The IAM user is forbidden in the currently selected region.	403	The IAM user is not allowed to access the region.	Contact technical support.

5 FAQs

5.1 How Do I Call APIs in Multi-Project/Subproject Scenarios?

To access resources in a subproject by calling APIs, add the **X-Project-Id** parameter to the request header and set the parameter value to the subproject ID. For details on how to add the **X-Project-Id** parameter, see **Signing SDKs and Demo**.

5.2 Does API Gateway Support Persistent Connections?

Yes. But you should use persistent connections properly to avoid occupying too many resources.

5.3 Must the Request Body Be Signed?

No. If you do not want to sign the request body, add the following parameter and value to the message header:

X-Sdk-Content-Sha256:UNSIGNED-PAYLOAD

UNSIGNED-PAYLOAD indicates the hashed position value calculated based on the request body.

5.4 Are Request Header Parameters Required for Signing Requests?

If you sign API requests by following the instructions in **AK/SK Signing and Authentication Algorithm**, only **X-Sdk-Date** is required and other request header parameters are optional.

If you use an SDK provided by HUAWEI CLOUD to sign API requests, you do not need to consider which request header parameters are required. The SDK determines which parameters are required and automatically generates the

signature information. If the value of a request header parameter changes after requests are signed, assign a value to the parameter again after signing.

5.5 How Do I Use a Temporary AK/SK to Sign Requests?

Add the following parameter and value to the message header:

X-Security-Token:*{securityToken}*

Then, use a temporary AK/SK and the SDK used for AK/SK-based authentication to sign the request.

For details about how to obtain a temporary AK/SK and security token, see the ***IAM API Reference***.

5.6 Common Errors Related to IAM Authentication Information

You may encounter the following errors related to IAM authentication information:

- **Incorrect IAM authentication information: verify aksk signature fail**
- **Incorrect IAM authentication information: AK access failed to reach the limit,forbidden**
- **Incorrect IAM authentication information: decrypt token fail**
- **Incorrect IAM authentication information: Get secretKey failed**

Incorrect IAM authentication information: verify aksk signature fail

```
{
  "error_msg": "Incorrect IAM authentication information: verify aksk signature fail, ......
  "error_code": "APIGW.0301",
  "request_id": "******"
}
```

Possible Cause

The signature algorithm is incorrect, and the signature calculated by the client is different from that calculated by API Gateway.

Solution

Step 1 Obtain the canonicalRequest calculated by API Gateway.

Obtain the canonicalRequest calculated by API Gateway from the following error information:

```
{
  "error_msg": "Incorrect IAM authentication information: verify aksk signature fail,canonicalRequest:PUT|/v2/******/instances/******/configs/||authorization:SDK-HMAC-SHA256 Access=******, SignedHeaders=authorization;content-length;content-type;host;x-project-id;x-sdk-date, Signature=******|content-length:84|content-type:application/json;charset=UTF-8|host:******|x-project-id:******|x-sdk-date:20201117T072119Z||authorization;content-length;content-type;host;x-project-id;x-sdk-date|******",
  "error_code": "APIGW.0301",
  "request_id": "******"
}
```


Replace vertical bars (|) with line breakers to change the error information as follows:

```
{
  "error_msg": "Incorrect IAM authentication information: verify aksk signature fail,canonicalRequest:PUT
/v2/******/instances/******/configs/

  authorization:SDK-HMAC-SHA256 Access=GRFQJFPWGL34UZBRLSDJ,
SignedHeaders=authorization;content-length;content-type;host;x-project-id;x-sdk-date,
Signature=******
  content-length:84
  content-type:application/json;charset=UTF-8
  host:******
  x-project-id:******
  x-sdk-date:20201117T072119Z

  authorization;content-length;content-type;host;x-project-id;x-sdk-date
******",
  "error_code": "APIGW.0301",
  "request_id": "******"
}
```

Step 2 Obtain the canonicalRequest calculated by the client by printing logs or using debug interrupts. The following table describes the functions used to calculate the canonicalRequest in the SDKs of different languages.

Table 5-1 Functions for calculating canonicalRequest in the SDKs of common languages

Language	Function
Java (earlier than 3.1.0)	Sign function in com.cloud.sdk.auth.signer.DefaultSigner.class of **libs/java-sdk-core-*.jar**
Java (3.1.0 or later)	Sign function in com.cloud.sdk.auth.signer.Signer.class of **libs/java-sdk-core-*.jar**
C++	Signer::createSignature function in **signer.cpp**.
C#	Sign function in **signer.cs**
C	sig_sign function in **signer.c**
Go	Sign function in **signer.go**
JavaScript	Signer.prototype.Sign function in **signer.js**
PHP	Sign function in **signer.php**
Python	Sign function in **signer.py**

Step 3 Check whether the domain name, method, protocol, path, query strings, headers, and body parameters of canonicalRequest obtained in **Step 1** are the same as those obtained in **Step 2**.

- If they are different, the common causes are as follows:
  - Some HTTP clients automatically add **charset=utf-8** to the signature header content-type.

  - The user used a proxy to forward requests. The URL, query strings, headers, and body in the request forwarded by the proxy to API Gateway are inconsistent with those signed by the client.
  - Some HTTP clients automatically ignore the body of requests that use the GET or DELETE method.
  - Some earlier version SDKs do not allow special characters in URLs.
  - Some earlier version SDKs do not support query strings that contain a key with multiple values, for example, **?a=1&a=2**.
  - Some earlier version SDKs do not allow query strings in URLs.
  - The user-agent header in the actual request is different from the signed user-agent header.
  - Multiple headers with the same name exist.
  - Multiple query strings with the same name exist.
  - The cannonicalRequest contains the authorization header, which conflicts with the signature header.
- If they are consistent, check whether the AppSecret or SK is correct.

  Common cause: The AppSecret or SK contains unnecessary spaces.

----End

Incorrect IAM authentication information: AK access failed to reach the limit,forbidden

```
{
  "error_msg": "Incorrect IAM authentication information: AK access failed to reach the limit,forbidden." .....
  "error_code": "APIGW.0301",
  "request_id": "******"
}
```

Possible Causes

- The AK/SK signature calculation is incorrect. Resolve the problem by referring to **Incorrect IAM authentication information: verify aksk signature fail**.
- The AK and SK do not match.
- AK/SK authentication fails for more than five consecutive times, and the AK/SK pair is locked for five minutes. (Authentication requests are rejected within this period).
- An expired token is used for token authentication.

Incorrect IAM authentication information: decrypt token fail

```
{
  "error_msg": "Incorrect IAM authentication information: decrypt token fail",
  "error_code": "APIGW.0301",
  "request_id": "******"
}
```

Possible Cause

The token cannot be parsed for IAM authentication of the API.

Solution

- Check whether the token is correct.
- Check whether the token has been obtained in the environment where the API is called.

Incorrect IAM authentication information: Get secretKey failed

```
{
"error_msg": "Incorrect IAM authentication information: Get secretKey failed,ak:******,err:ak not exist",
"error_code": "APIGW.0301",
"request_id": "******"
}
```

Possible Cause

The AK used for IAM authentication of the API does not exist.

Solution

Check whether the AK is correct.

