

# API 网关

## 历史功能

## 产品文档



腾讯云

---

**【版权声明】**

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

**【商标声明】**

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

**【服务声明】**

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

## 文档目录

历史功能

    密钥对认证

        密钥管理

        密钥对认证

        Java（密钥对认证）

        Go（密钥对认证）

        Python（密钥对认证）

        JavaScript（密钥对认证）

        PHP（密钥对认证）

        C++（密钥对认证）

        Erlang（密钥对认证）

        签名生成说明

# 历史功能

## 密钥对认证

## 密钥管理

最近更新时间：2023-12-22 10:08:03

## 操作场景

该任务指导您在 API 网关控制台上管理密钥对。

**创建密钥：**创建新的密钥对，支持通过“自动生成密钥”和“自定义密钥”两种方式进行创建。创建成功的密钥对需要通过使用计划和 API 进行绑定后才能作为 API 的访问凭证。

**禁用/启用密钥：**禁用密钥后，API 网关将拒绝使用该密钥生成签名的所有请求；再次启用后可恢复正常请求。

**更换密钥：**在密钥名和 SecretId 保持不变的前提下，重置密钥的 SecretKey。

**删除密钥：**删除禁用状态的密钥，已启用的密钥不支持删除。

## 操作步骤

### 自动生成密钥

1. 登录 [API 网关控制台](#)。
2. 在左侧导航栏，单击**密钥**，进入密钥列表页。
3. 在密钥列表页单击**新建**，弹出新建密钥对话框。
4. 密钥类型选择**自动生成密钥**，输入密钥名称。
5. 单击**提交**，即可自动生成密钥。

Create Key

For the security of your services, please change the secret key regularly.  
Uppercase and lowercase letters are supported.

Secret Key Type

☒ Auto-generate key

☐ Custom Secret Key

Secret Key Name

Up to 50 chars, supporting letters, digits, underscores, and dashes.

Submit

Close

## 创建自定义密钥

1. 登录 [API 网关控制台](#)。
2. 在左侧导航栏，单击**密钥**，进入密钥列表页。
3. 在密钥列表页点击**新建**，弹出新建密钥对话框。
4. 密钥类型选择**自定义密钥**，输入密钥名称、SecretId、SecretKey。
5. 单击**提交**，即可成功创建自定义密钥。

### Create Key

 For the security of your services, please change the secret key regularly. Uppercase and lowercase letters are supported.

Secret Key Type

☐ Auto-generate key ☒ Custom Secret Key

Secret Key Name

Up to 50 chars, supporting letters, digits, underscores, and dashes.

SecretId

5-50 chars, supporting letters, digits, underscores, and dashes.

SecretKey

10-50 chars, supporting letters, digits, underscores, and dashes.

Submit

Close

## 禁用密钥

1. 登录 [API 网关控制台](#)。
2. 在左侧导航栏，单击**密钥**，进入密钥列表页。
3. 在密钥列表页找到状态为**使用中**且需要禁用的密钥，单击操作栏的**禁用**。
4. 在弹出的确认对话框中单击**确认**，即可禁用密钥，密钥状态变为**已停用**。

| Create          |                                                                  |        |                     |
|-----------------|------------------------------------------------------------------|--------|---------------------|
| Secret Key Name | Secret Key                                                       | Status | Modification Time   |
| secretA         | SecretId: AKIDjsLW66XH21mj8nvMS5iwZnPF3j2lDm6ABxgb<br>SecretKey: | In Use | 2020-03-23 14:53:06 |
| Total items: 1  |                                                                  |        |                     |

## 启用密钥

1. 登录 [API 网关控制台](#)。
2. 在左侧导航栏，单击**密钥**，进入密钥列表页。
3. 在密钥列表页找到状态为**已停用**且需要启用的密钥，单击操作栏的**启用**。
4. 在弹出的确认对话框中单击**确认**，即可启用密钥，密钥状态变为**使用中**。

## 更换密钥

1. 登录 [API 网关控制台](#)。
2. 在左侧导航栏，单击**密钥**，进入密钥列表页。
3. 在密钥列表页找到状态为**使用中**且需要更换 SecretKey 的密钥，单击操作栏的**更换**。
4. 在弹出的确认对话框中单击**确认**，即可成功更换密钥，重置密钥的 SecretKey。

## 删除密钥

1. 登录 [API 网关控制台](#)。
2. 在左侧导航栏，单击**密钥**，进入密钥列表页。
3. 在密钥列表页找到状态为**已停用**且需要删除的密钥，单击操作栏的**删除**。
4. 在弹出的确认对话框中单击**确认**，即可成功删除密钥。

## 注意事项

**已停用**状态下的密钥不支持更换、绑定使用计划，请启用后再进行操作。

**使用中**状态下的密钥不支持删除，请禁用后再进行操作。

自定义密钥的 SecretId 全地域唯一，如果自定义密钥创建失败，请调整 SecretId 后重试。

# 密钥对认证

最近更新时间：2023-12-22 10:08:14

您可以使用 `secret_id` 和 `secret_key` 对您的 API 进行认证管理。`secret_id` 和 `secret_key` 成对出现，这里将它们称为 `secret_id/secret_key` 对。

在使用 `secret_id/secret_key` 对认证前，您需要先创建一对 `secret_id` 和 `secret_key`。

## 密钥对鉴权 API

创建 API 时，您可以选择鉴权类型为“密钥对鉴权”。当选择密钥对鉴权类型的 API 发布时，只有使用正确密钥对发起的访问能够通过 API 网关的校验，不携带密钥或携带错误的密钥对不能通过 API 网关的鉴权。

API 创建完成后，您需要使用“使用计划”功能将密钥对与 API 或 API 所在服务进行绑定。配置详情请参考 [使用计划](#)。

## 密钥内容

**secret\_id** 示例：`AKIDCg*****j548pN` 用于标识所使用的哪个密钥，并参与签名计算，传输过程中体现。

**secret\_key** 示例：`ZxF2wh*****N2oPrC` 用于签名计算，传递过程中无体现。

## 计算方法

### 最终发送内容

最终发送的 HTTP 请求内至少包含两个 header：Date 和 X-Date 二选一以及 Authorization，可以包含 Source 等更多 header。

Date header 的值为 GMT 格式的 HTTP 请求构造时间，例如 Fri, 09 Oct 2015 00:00:00 GMT。

X-Date header 的值为 GMT 格式的 HTTP 请求构造时间，例如 Mon, 19 Mar 2018 12:08:40 GMT。15分钟超时。

Source header 代表签名水印值，可以填写任意值或不填写。

Authorization header 的例如 `Authorization: hmac id="secret_id", algorithm="hmac-sha1", headers="date source", signature="Base64(HMAC-SHA1(signing_str, secret_key))"`。

现对 Authorization 内的各部分分别解释：

**hmac** 固定内容，用于标识计算方法。

**ID** 其值为密钥内的 `secret_id` 的值。

**algorithm** 加密算法，当前支持的是 `hmac-sha1`。

**headers** 参与签名计算的 header，按实际计算时的顺序排列。

**signature** 计算签名后得到的签名，**signing\_str** 是签名内容。

## 签名计算方法

签名由两部分并根据指定加密算法进行计算，以 hmac-sha1 算法举例：

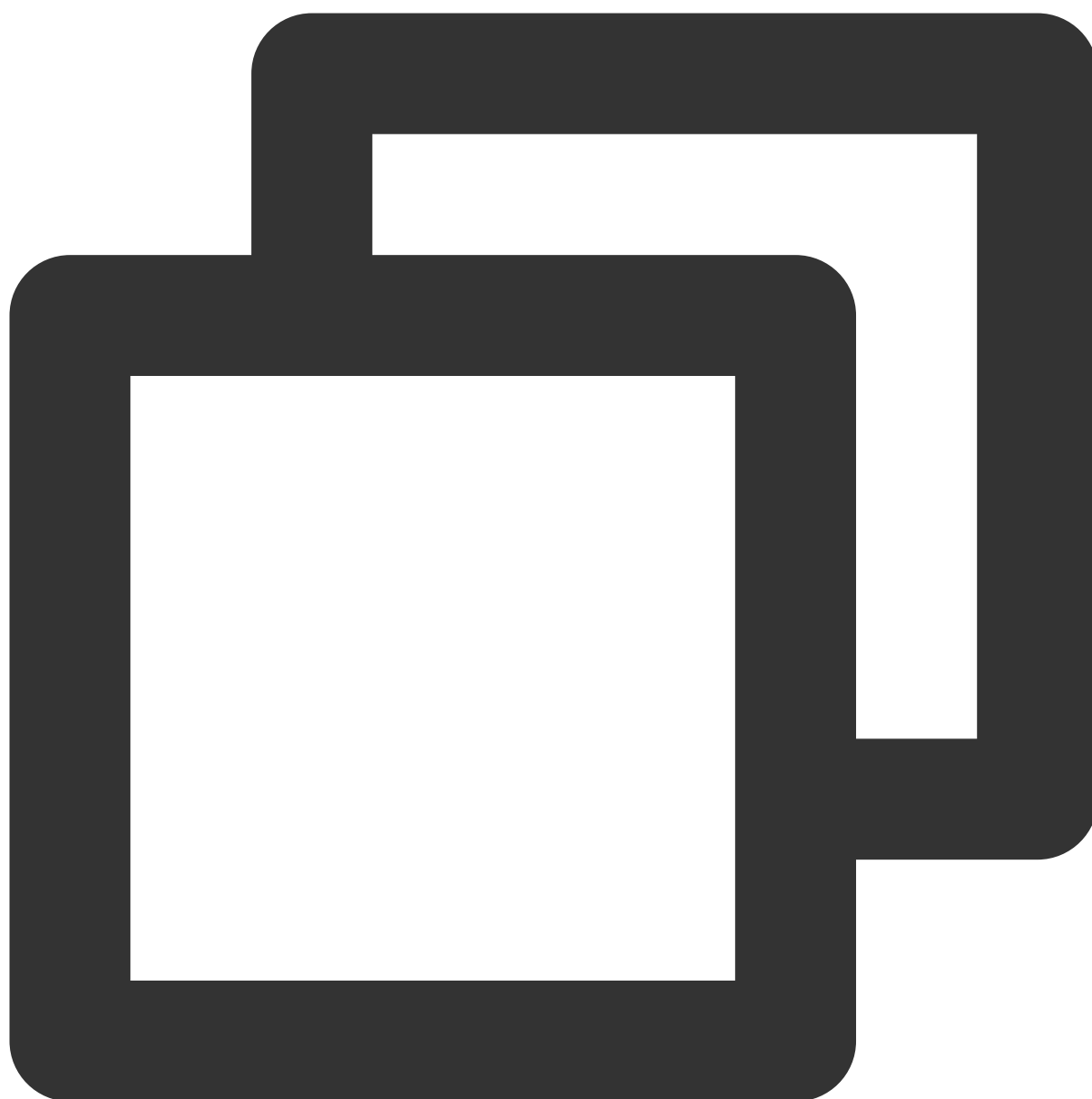
### 签名内容

首先生成签名内容，签名内容由自定义的 **header** 组成，**header** 内建议至少包含 **date**，可以包含更多其他 **header**。

**header** 按如下要求转换后按顺序排列：

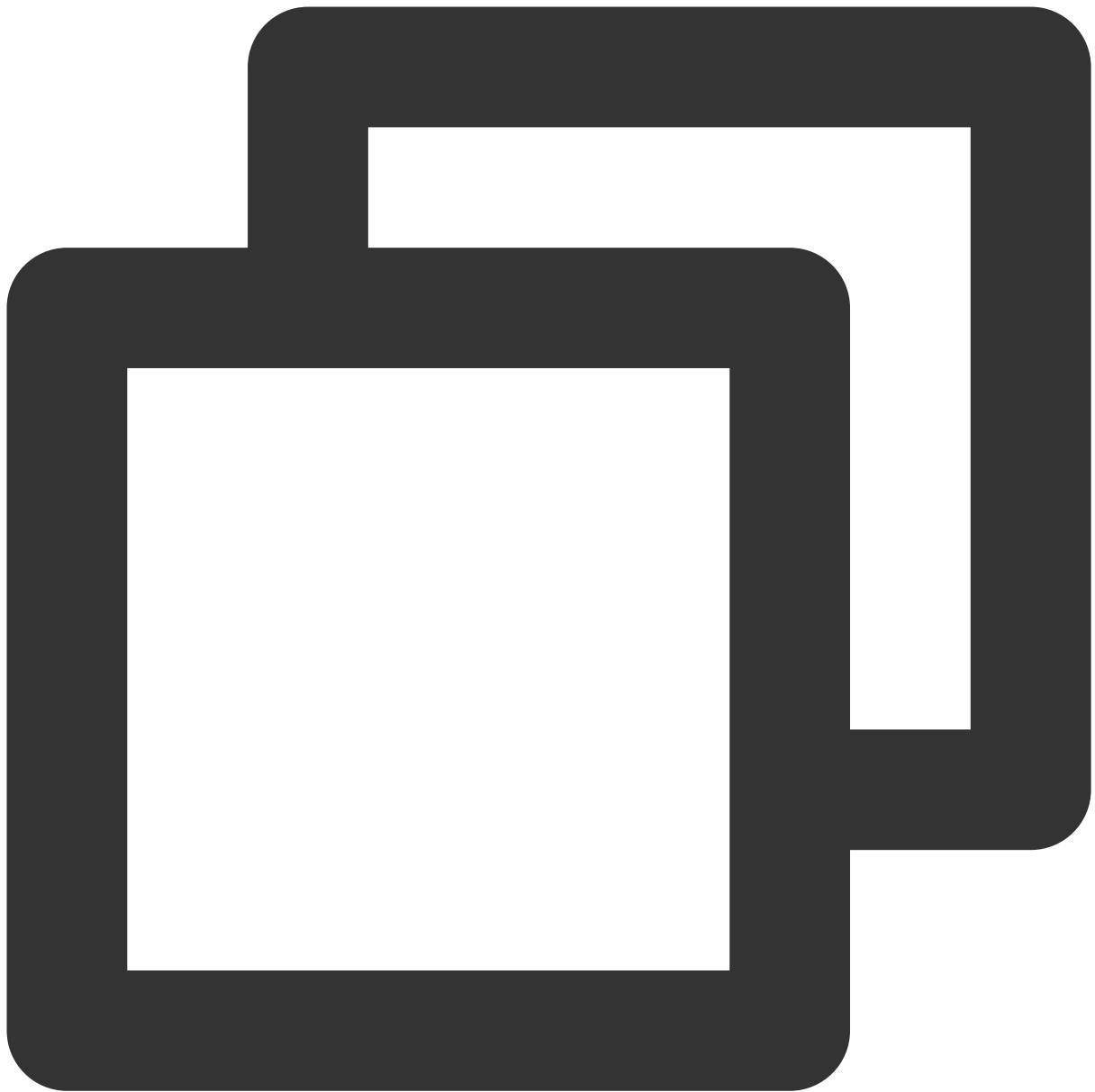
1. **header** 名转换为小写，小写 **header** 名后面为 **ascii 字符:** 和 **ascii 空格字符**。
2. 然后附加 **header** 值。
3. 如果不是最后一条需构造签名的 **header**，附上 **ascii 换行字符** `\\n`。

例如有两个 **header** 参与构建签名内容（仅为示例，请根据业务实际情况填写字段）：



Date: Fri, 09 Oct 2015 00:00:00 GMT  
Source: AndriodApp

生成的签名内容为：



```
date: Fri, 09 Oct 2015 00:00:00 GMT  
source: AndriodApp
```

### 计算签名

将上一步生成的签名内容，使用 `Base64(HMAC-SHA1(signing_str, secret_key))` 算法进行计算生成签名，也就是：使用签名内容作为输入信息，密钥内的 `secret_key` 内容作为密钥，使用 `HMAC-SHA1` 算法进行计算得出加密签名内容。

使用 `Base64` 对算出的加密签名内容进行转换生成可传递的签名内容。

## 使用签名

如**最终发送内容**中所示的一样，在 Authorization header 的 signature 处填入上一步计算完成后的签名。

## 注意事项

### header 对应

Authorization 中 headers 位置填入的需要是参与计算签名的 header 的名称，并建议转换为小写，以 ascii 空格分隔。例如，参与计算的 header 为 date 和 source 时，此位置的形式为 headers="date source"；参与计算的 header 仅为 x-date 时，此位置的形式为 headers="x-date"。

### 签名内容生成

排列内容时，请注意 header 名后面跟的冒号和空格，如有遗失也可能导致校验无法通过。SecretId、SecretKey、URL、Host 需要修改为真实信息。

#### 说明：

常用语言的签名 Demo 请参考 [开发者指南-多种语言生成签名](#)。

# Java（密钥对认证）

最近更新时间：2023-12-22 10:08:24

## 操作场景

该任务指导您使用 Java 语言，通过密钥对鉴权来对您的 API 进行认证管理。

## 操作步骤

1. 在 [API 网关控制台](#)，创建一个 API，选择鉴权类型为“密钥对鉴权”（参考 [创建 API 概述](#)）。
2. 将 API 所在服务发布至发布环境（参考 [服务发布与下线](#)）。
3. 在控制台密钥管理界面创建密钥对。
4. 在控制台使用计划界面创建使用计划，并将使用计划与已创建的密钥对绑定（参考 [使用计划示例](#)）。
5. 将使用计划与 API 或 API 所在服务进行绑定。
6. 参考 [示例代码](#)，使用 Java 语言生成签名内容。

## 注意事项

最终发送的 HTTP 请求内至少包含两个 Header：Date 和 X-Date 二选一以及 Authorization，可以包含更多 header。

如果使用 Date Header，服务端将不会校验时间；如果使用 X-Date Header，服务端将校验时间。

Source Header 是签名水印值，可以填写任意值，也可不填写，Demo 中默认使用“xxxxxx”。

Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Fri, 09 Oct 2015 00:00:00 GMT。

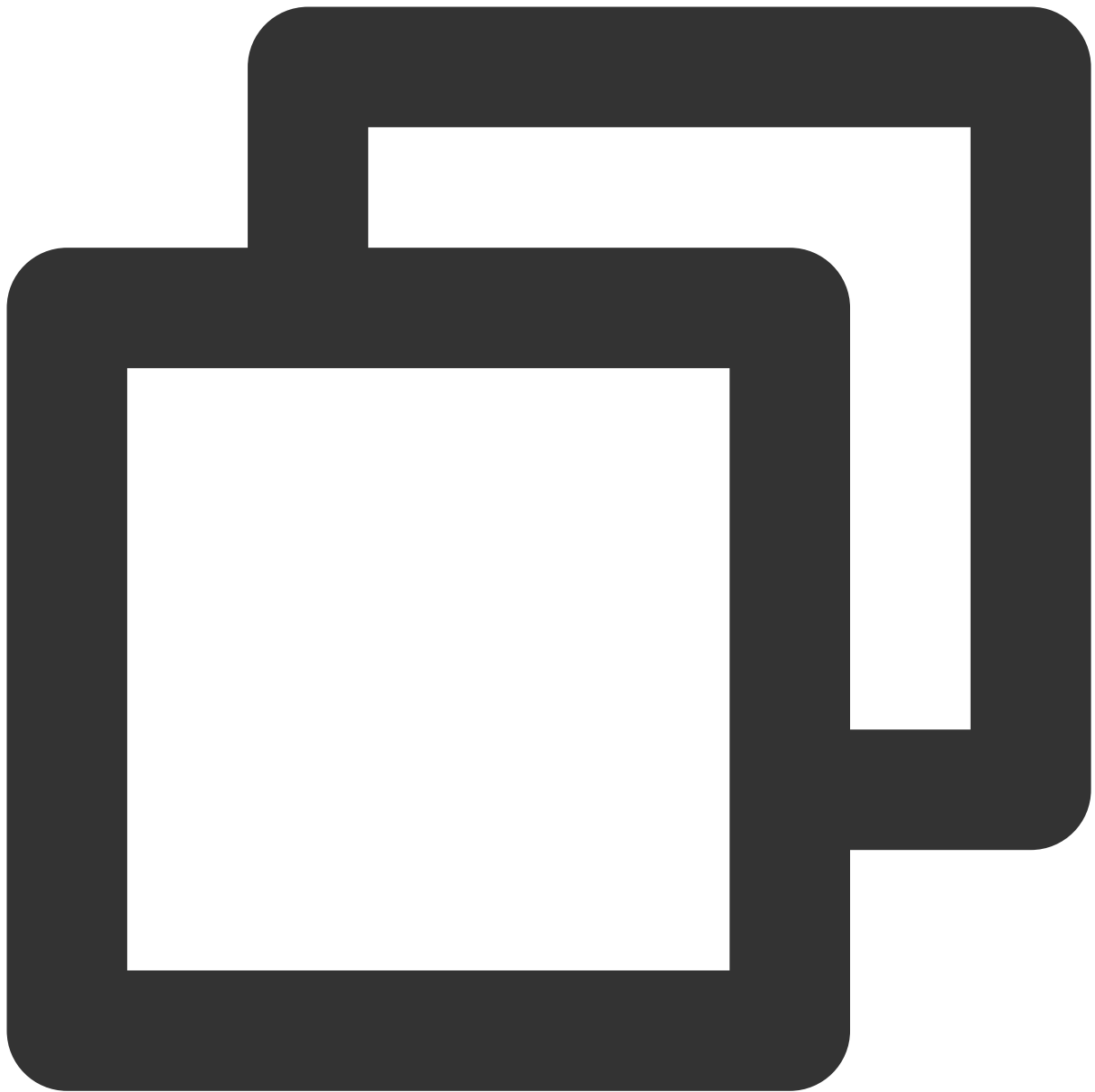
X-Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Mon, 19 Mar 2018 12:08:40 GMT。X-Date Header 里的时间和当前时间的差值不能超过15分钟。

如果是微服务 API，Header 中需要添加“X-NameSpace-Code”和“X-MicroService-Name”两个字段，通用 API 不需要添加，Demo 中默认添加了这两个字段。

本 Demo 中包含了 GET 方法和 POST 方法的示例，您可以根据自己的需求选用。

## 示例代码

**Base64.java**



```
package apigatewayDemo;

import java.io.UnsupportedEncodingException;

public class Base64 {
    private static char[] base64EncodeChars = new char[] { 'A', 'B', 'C', 'D',
        'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q',
        'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd',
        'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
        'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1', '2', '3',
        '4', '5', '6', '7', '8', '9', '+', '/' };
}
```

```
private static byte[] base64DecodeChars = new byte[] { -1, -1, -1, -1, -1,
    -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
    -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
    -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55, 56, 57, 58, 59,
    60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
    10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
    -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
    38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1,
    -1, -1 };

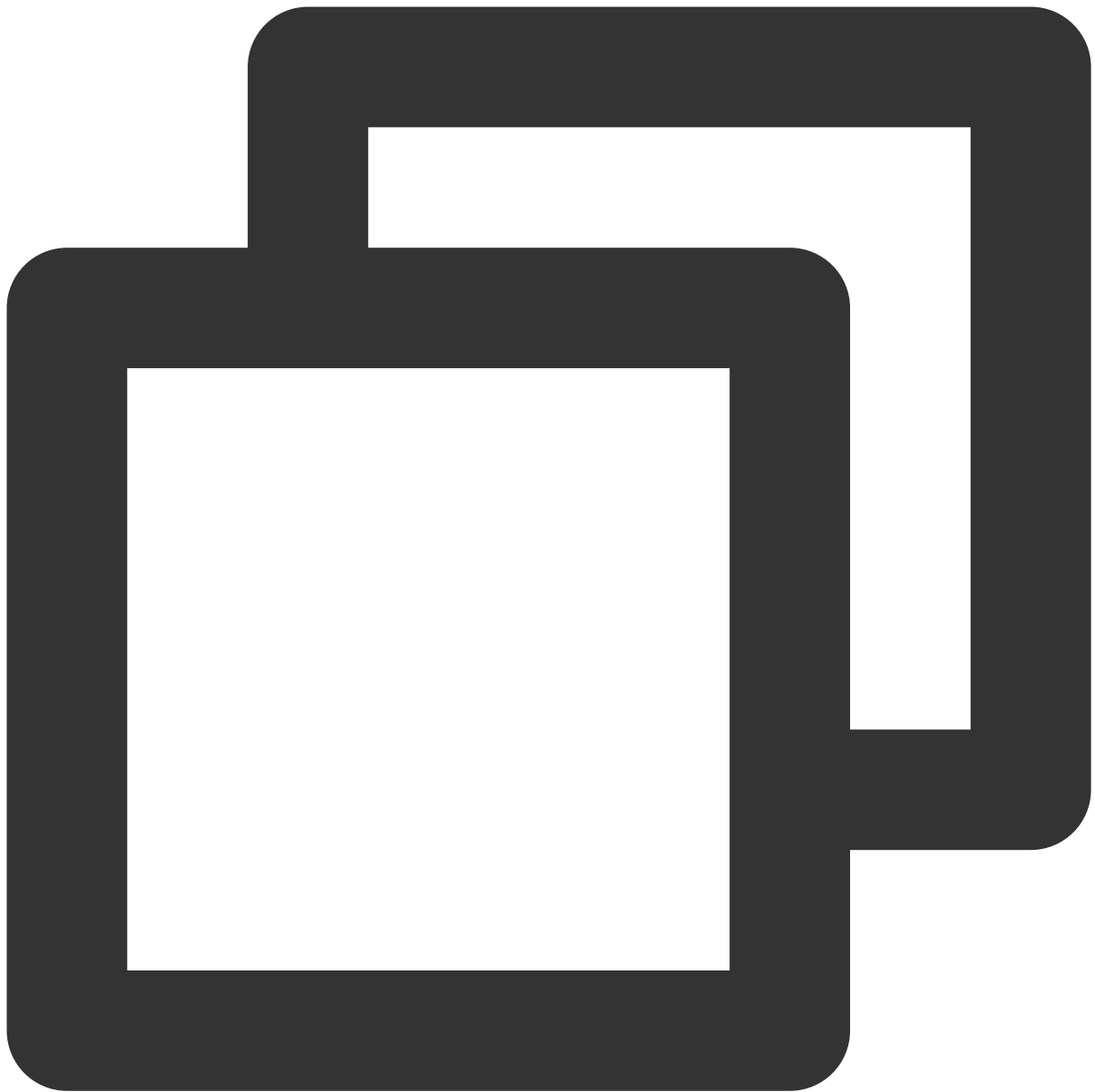
public static String encode(byte[] data) {
    StringBuffer sb = new StringBuffer();
    int len = data.length;
    int i = 0;

        int b1, b2, b3;
    while (i < len) {
        b1 = data[i++] & 0xff;
        if (i == len) {
            sb.append(base64EncodeChars[b1 >>> 2]);
            sb.append(base64EncodeChars[(b1 & 0x3) << 4]);
            sb.append("==");
            break;
        }
        b2 = data[i++] & 0xff;
        if (i == len) {
            sb.append(base64EncodeChars[b1 >>> 2]);
            sb.append(base64EncodeChars[((b1 & 0x03) << 4)
                | ((b2 & 0xf0) >>> 4)]);
            sb.append(base64EncodeChars[(b2 & 0x0f) << 2]);
            sb.append("=");
            break;
        }
        b3 = data[i++] & 0xff;
        sb.append(base64EncodeChars[b1 >>> 2]);
        sb.append(base64EncodeChars[((b1 & 0x03) << 4)
            | ((b2 & 0xf0) >>> 4)]);
        sb.append(base64EncodeChars[((b2 & 0x0f) << 2)
            | ((b3 & 0xc0) >>> 6)]);
        sb.append(base64EncodeChars[b3 & 0x3f]);
    }
    return sb.toString();
}

public static byte[] decode(String str) throws UnsupportedOperationException {
    StringBuffer sb = new StringBuffer();
    byte[] data = str.getBytes("US-ASCII");
```

```
int len = data.length;
int i = 0;
int b1, b2, b3, b4;
while (i < len) {
    /* b1 */
    do {
        b1 = base64DecodeChars[data[i++]];
    } while (i < len && b1 == -1);
    if (b1 == -1)
        break;
    /* b2 */
    do {
        b2 = base64DecodeChars[data[i++]];
    } while (i < len && b2 == -1);
    if (b2 == -1)
        break;
    sb.append((char) ((b1 << 2) | ((b2 & 0x30) >>> 4)));
    /* b3 */
    do {
        b3 = data[i++];
        if (b3 == 61)
            return sb.toString().getBytes("ISO-8859-1");
        b3 = base64DecodeChars[b3];
    } while (i < len && b3 == -1);
    if (b3 == -1)
        break;
    sb.append((char) (((b2 & 0x0f) << 4) | ((b3 & 0x3c) >>> 2)));
    /* b4 */
    do {
        b4 = data[i++];
        if (b4 == 61)
            return sb.toString().getBytes("ISO-8859-1");
        b4 = base64DecodeChars[b4];
    } while (i < len && b4 == -1);
    if (b4 == -1)
        break;
    sb.append((char) (((b3 & 0x03) << 6) | b4));
}
return sb.toString().getBytes("ISO-8859-1");
}
```

## SignAndSend.java



```
package apigatewayDemo;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.text.SimpleDateFormat;
import java.util.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.io.PrintWriter;
import java.net.URL;
import java.net.URLConnection;
import java.net.HttpURLConnection;
import java.io.OutputStreamWriter;

public class SignAndSend {
    private static final String CONTENT_CHARSET = "UTF-8";
    private static final String HMAC_ALGORITHM = "HmacSHA1";
    public static String sign(String secret, String timeStr)
        throws NoSuchAlgorithmException, UnsupportedEncodingException, InvalidKeyException {
        //获取签名字符串, Source是签名水印值, 可填写任意值, Demo中使用“xxxxxxx”
        String signStr = "date: "+timeStr+"\n"+"source: "+xxxxxxx";
        //获取接口签名
        String sig = null;
        Mac mac1 = Mac.getInstance(HMAC_ALGORITHM);
        byte[] hash;
        SecretKeySpec secretKey = new SecretKeySpec(secret.getBytes(CONTENT_CHARSET), HMAC_ALGORITHM);
        mac1.init(secretKey);
        hash = mac1.doFinal(signStr.getBytes(CONTENT_CHARSET));
        sig = new String(Base64.encode(hash));
        System.out.println("signValue--->" + sig);
        return sig;
    }

    public static HttpURLConnection NewHttpUrlCon(String url, String secretId, String secretKey) {
        //获取当前 GMT 时间
        Calendar cd = Calendar.getInstance();
        SimpleDateFormat sdf = new SimpleDateFormat("EEE, dd MMM yyyy HH:mm:ss 'GMT'");
        sdf.setTimeZone(TimeZone.getTimeZone("GMT"));
        String timeStr = sdf.format(cd.getTime());
        HttpURLConnection httpUrlCon = null;
        try {
            String urlNameString = url;
            URL realUrl = new URL(urlNameString);
            // 打开和 URL 之间的连接
            URLConnection connection = realUrl.openConnection();
            httpUrlCon = (HttpURLConnection)connection;
            // 设置通用的请求属性, Source 是签名水印值, 可填写任意值, Demo 中使用“xxxxxxx”
            httpUrlCon.setRequestProperty("Host", url);
            httpUrlCon.setRequestProperty("Accept", "text/html, */*; q=0.01");
            httpUrlCon.setRequestProperty("Source", "xxxxxxx");
            httpUrlCon.setRequestProperty("Date", timeStr);
            String sig = sign(secretKey, timeStr);
            String authen = "hmac id=\\\\"+secretId+"\\", algorithm=\\\\"hmac-sha1\\",
            System.out.println("authen --->" + authen);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return httpUrlCon;
    }
}
```

```
        httpUrlCon.setRequestProperty("Authorization", authen);
        httpUrlCon.setRequestProperty("X-Requested-With", "XMLHttpRequest");
        httpUrlCon.setRequestProperty("Accept-Encoding", "gzip, deflate, sdch");

        // 如果是微服务 API, Header 中需要添加 'X-NameSpace-Code'、'X-MicroService-Name'
        httpUrlCon.setRequestProperty("X-NameSpace-Code", "testmic");
        httpUrlCon.setRequestProperty("X-MicroService-Name", "provider-demo");
    } catch (Exception e) {
        System.out.println("新建连接异常!" + e);
        e.printStackTrace();
    }
    return httpUrlCon;
}

public static String sendGet(String url, String secretId, String secretKey) {
    String result = "";
    BufferedReader in = null;
    try {
        // 新建连接
        HttpURLConnection httpUrlCon = NewHttpUrlCon(url, secretId, secretKey);

        // 建立实际的连接
        httpUrlCon.connect();

        // 获取所有响应头字段
        Map<String, List<String>> map = httpUrlCon.getHeaderFields();

        // 遍历所有的响应头字段
        for (String key : map.keySet()) {
            System.out.println(key + "--->" + map.get(key));
        }

        // 定义 BufferedReader 输入流来读取 URL 的响应
        in = new BufferedReader(new InputStreamReader(
            httpUrlCon.getInputStream()));
        String line;
        while ((line = in.readLine()) != null) {
            result += line;
        }
    } catch (Exception e) {
        System.out.println("发送GET请求出现异常!" + e);
        e.printStackTrace();
    }

    // 使用 finally 块来关闭输入流
    finally {
        try {
```

```
        if (in != null) {
            in.close();
        }
    } catch (Exception e2) {
        e2.printStackTrace();
    }
}
return result;
}

public static String sendPost(String url, String secretId, String secretKey) {
    String result = "";
    BufferedReader in = null;
    try {

        // 新建连接
        HttpURLConnection httpUrlCon = NewHttpUrlCon(url, secretId, secretKey);

        //设置 post 请求
        httpUrlCon.setRequestMethod("POST");
        httpUrlCon.setUseCaches(false); //Post 请求不能使用缓存
        httpUrlCon.setDoOutput(true);
        httpUrlCon.setDoInput(true);

        //将请求数据放到 body 里
        OutputStreamWriter out = new OutputStreamWriter(httpUrlCon.getOutputStream());
        String jsonStr = "{\\"caller\\":\\"apigw\\", \\"data\\":\\"test post\\"}";
        out.write(jsonStr);
        out.flush();
        out.close();

        // 建立实际的连接
        httpUrlCon.connect();

        // 获取所有响应头字段
        Map<String, List<String>> map = httpUrlCon.getHeaderFields();

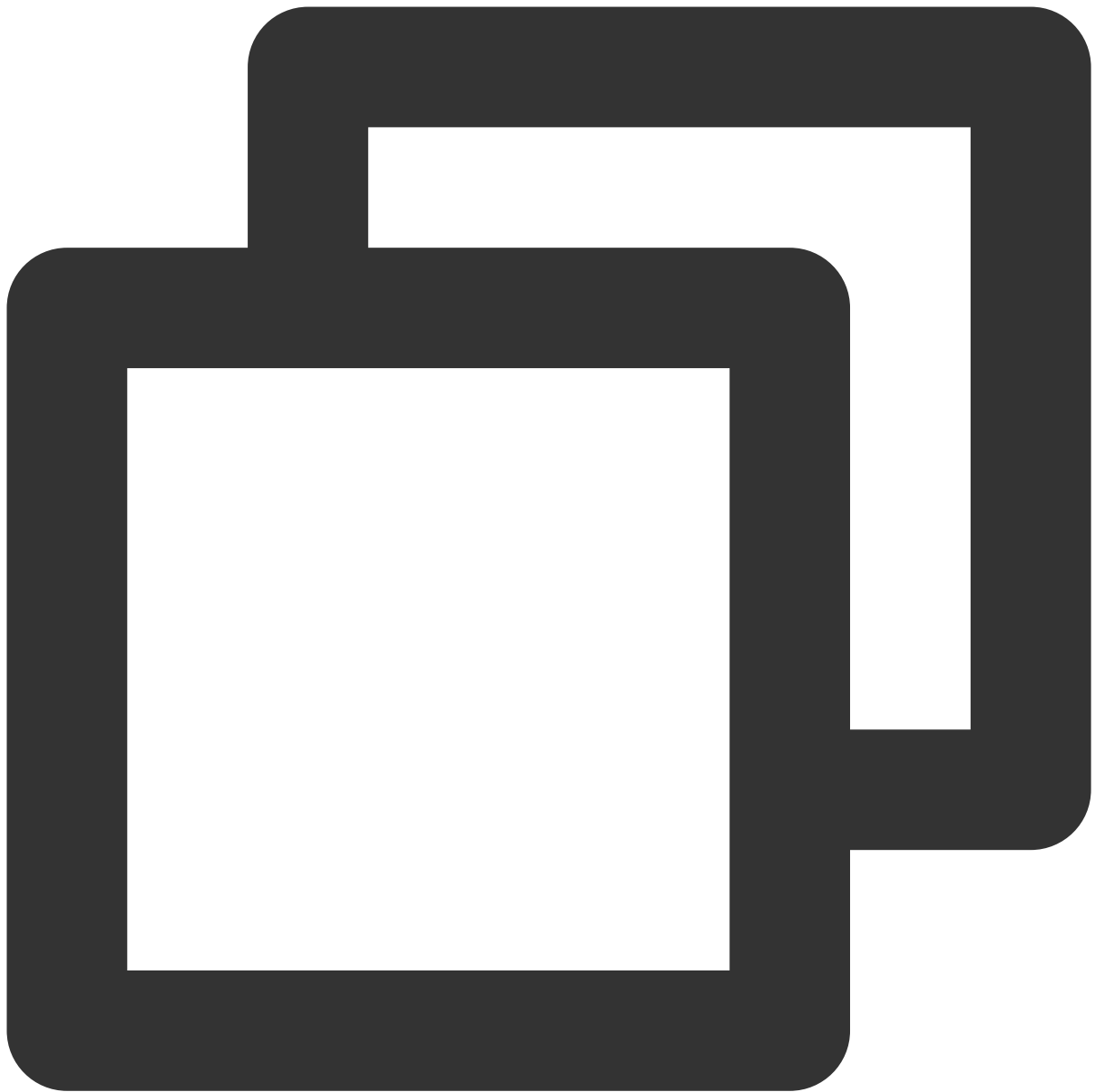
        // 遍历所有的响应头字段
        for (String key : map.keySet()) {
            System.out.println(key + "---->" + map.get(key));
        }

        // 定义 BufferedReader 输入流来读取 URL 的响应
        in = new BufferedReader(new InputStreamReader(
            httpUrlCon.getInputStream()));
        String line;
        while ((line = in.readLine()) != null) {
```

```
        result += line;
    }
} catch (Exception e) {
    System.out.println("发送POST请求出现异常！" + e);
    e.printStackTrace();
}

// 使用 finally 块来关闭输入流
finally {
    try {
        if (in != null) {
            in.close();
        }
    } catch (Exception e2) {
        e2.printStackTrace();
    }
}
return result;
}
}
```

### Demo.java



```
package apigatewayDemo;
public class Demo {
    public static void main(String[] args) {
        String secretId = "your secretId"; // 密钥对的 SecretId
        String secretKey = "your secretKey"; // 密钥对的 SecretKey
        SignAndSend signAndSendInstance = new SignAndSend();

        //get 请求
        String getUrl = "http://service-xxxxxxx-1234567890.gz.apigw.tencentcs.com:
        String getResult = SignAndSend.sendGet(getUrl, secretId, secretKey);
        System.out.println(getResult);
```

```
//post 请求
String postUrl = "http://service-xxxxxxx-1234567890.gz.apigw.tencentcs.com
String postResult = SignAndSend.sendPost(postUrl, secretId, secretKey);
System.out.println(postResult);
}
}
```

# Go（密钥对认证）

最近更新时间：2023-12-22 10:08:33

## 操作场景

该任务指导您使用 Go 语言，通过密钥对鉴权来对您的 API 进行认证管理。

## 操作步骤

1. 在 [API 网关控制台](#)，创建一个 API，选择鉴权类型为“密钥对鉴权”（参考 [创建 API 概述](#)）。
2. 将 API 所在服务发布至发布环境（参考 [服务发布与下线](#)）。
3. 在控制台密钥管理界面创建密钥对。
4. 在控制台使用计划界面创建使用计划，并将使用计划与已创建的密钥对绑定（参考 [使用计划示例](#)）。
5. 将使用计划与 API 或 API 所在服务进行绑定。
6. 参考 [示例代码](#)，使用 Go 语言生成签名内容。

## 注意事项

最终发送的 HTTP 请求内至少包含两个 Header：Date 和 X-Date 二选一以及 Authorization，可以包含更多 header。

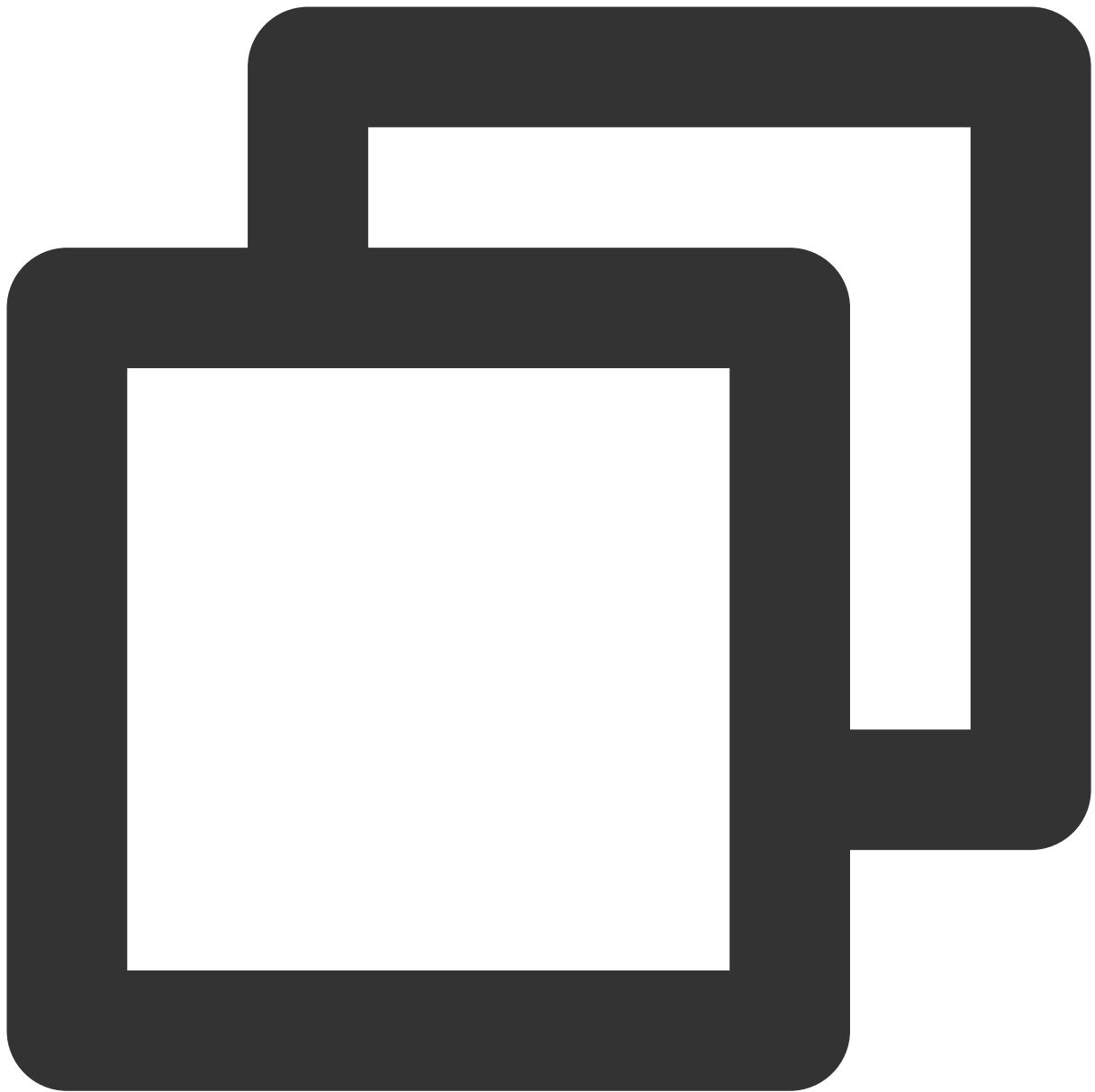
如果使用 Date Header，服务端将不会校验时间；如果使用 X-Date Header，服务端将校验时间。

Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Fri, 09 Oct 2015 00:00:00 GMT。

X-Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Mon, 19 Mar 2018 12:08:40 GMT。X-Date Header 里的时间和当前时间的差值不能超过15分钟。

如果是微服务 API，Header 中需要添加“X-NameSpace-Code”和“X-MicroService-Name”两个字段，通用 API 不需要添加，Demo 中默认添加了这两个字段。

## 示例代码



```
package main

import (
    "time"
    "fmt"
    "crypto/hmac"
    "crypto/sha1"
    "io"
    "io/ioutil"
    "encoding/base64"
    "net/http"
```

```
)

func calcAuthorization(source string, secretId string, secretKey string) (sign string,
timeLocation, _ := time.LoadLocation("Etc/GMT")
dateTime = time.Now().In(timeLocation).Format("Mon, 02 Jan 2006 15:04:05 GMT")
sign = fmt.Sprintf("x-date: %s\\nsource: %s", dateTime, source)
fmt.Println(sign)

//hmac-sha1
h := hmac.New(sha1.New, []byte(secretKey))
io.WriteString(h, sign)
sign = fmt.Sprintf("%x", h.Sum(nil))
sign = string(h.Sum(nil))
fmt.Println("sign:", fmt.Sprintf("%s", h.Sum(nil)))

//base64
sign = base64.StdEncoding.EncodeToString([]byte(sign))
fmt.Println("sign:", sign)

auth := fmt.Sprintf("hmac id=\\\"%s\\\", algorithm=\\\"hmac-sha1\\\", headers=\\\"x-
secretId, sign)
fmt.Println("auth:", auth)

return auth, dateTime, nil
}

func main () {
    SecretId := "your SecretId" // 密钥对的 SecretId
    SecretKey := "your SecretKey" // 密钥对的 SecretKey
    source := "xxxxxxx" // 签名水印值, 可填写任意值

    sign, dateTime, err := calcAuthorization(source, SecretId, SecretKey)
    if err != nil {
        fmt.Println(err)
        return
    }

    defaultDomain := "service-xxxxxxx-1234567890.ap-guangzhou.apigateway.myqcloud.

    reqUrl := "https://service-xxxxxxx-1234567890.ap-guangzhou.apigateway.myqcloud
    client := &http.Client{
        Timeout: 7 * time.Second, //set timeout
    }

    req, err := http.NewRequest("GET", reqUrl, nil) //set body
    if err != nil {
```

```
        fmt.Println(err)
        return
    }

    req.Header.Set("Accept", "*/*")
    req.Header.Set("Accept-Charset", "utf-8;")
    req.Header.Set("Host", defaultDomain)
    req.Header.Set("Source", source)
    req.Header.Set("X-Date", dateTime)
    req.Header.Set("Authorization", sign)

    // 如果是微服务 API, Header 中需要添加'X-NameSpace-Code'、'X-MicroService-Name'两个字
    req.Header.Set("x-NameSpace-Code", "testmic")
    req.Header.Set("x-MicroService-Name", "provider-demo")

    resp, err := client.Do(req)
    if err != nil {
        fmt.Println(err)
        return
    }
    defer resp.Body.Close()

    fmt.Println("status code:", resp.StatusCode)

    //get resp header
    var headerMsg string
    for key, _ := range resp.Header {
        headerMsg += fmt.Sprintf("\n%s:%s", key, resp.Header.Get(key))
    }
    fmt.Println(headerMsg)

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        fmt.Println(err)
        return
    }

    fmt.Println(string(body))
}
```

# Python（密钥对认证）

最近更新时间：2023-12-22 10:08:43

## 操作场景

该任务指导您使用 Python 语言，通过密钥对鉴权来对您的 API 进行认证管理。

## 操作步骤

1. 在 [API 网关控制台](#)，创建一个 API，选择鉴权类型为**密钥对鉴权**（参考 [创建 API 概述](#)）。
2. 将 API 所在服务发布至发布环境（参考 [服务发布与下线](#)）。
3. 在控制台密钥管理界面创建密钥对。
4. 在控制台使用计划界面创建使用计划，并将使用计划与已创建的密钥对绑定（参考 [使用计划示例](#)）。
5. 将使用计划与 API 或 API 所在服务进行绑定。
6. 参考 [示例代码](#)，使用 Python 语言生成签名内容。

## 环境依赖

API 网关提供 Python 2.7 和 Python 3 两个版本的示例代码，请您根据自己的 Python 版本合理选择。

## 注意事项

最终发送的 HTTP 请求内至少包含两个 Header：Date 和 X-Date 二选一以及 Authorization，可以包含更多 header。

如果使用 Date Header，服务端将不会校验时间；如果使用 X-Date Header，服务端将校验时间。

Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Fri, 09 Oct 2015 00:00:00 GMT。

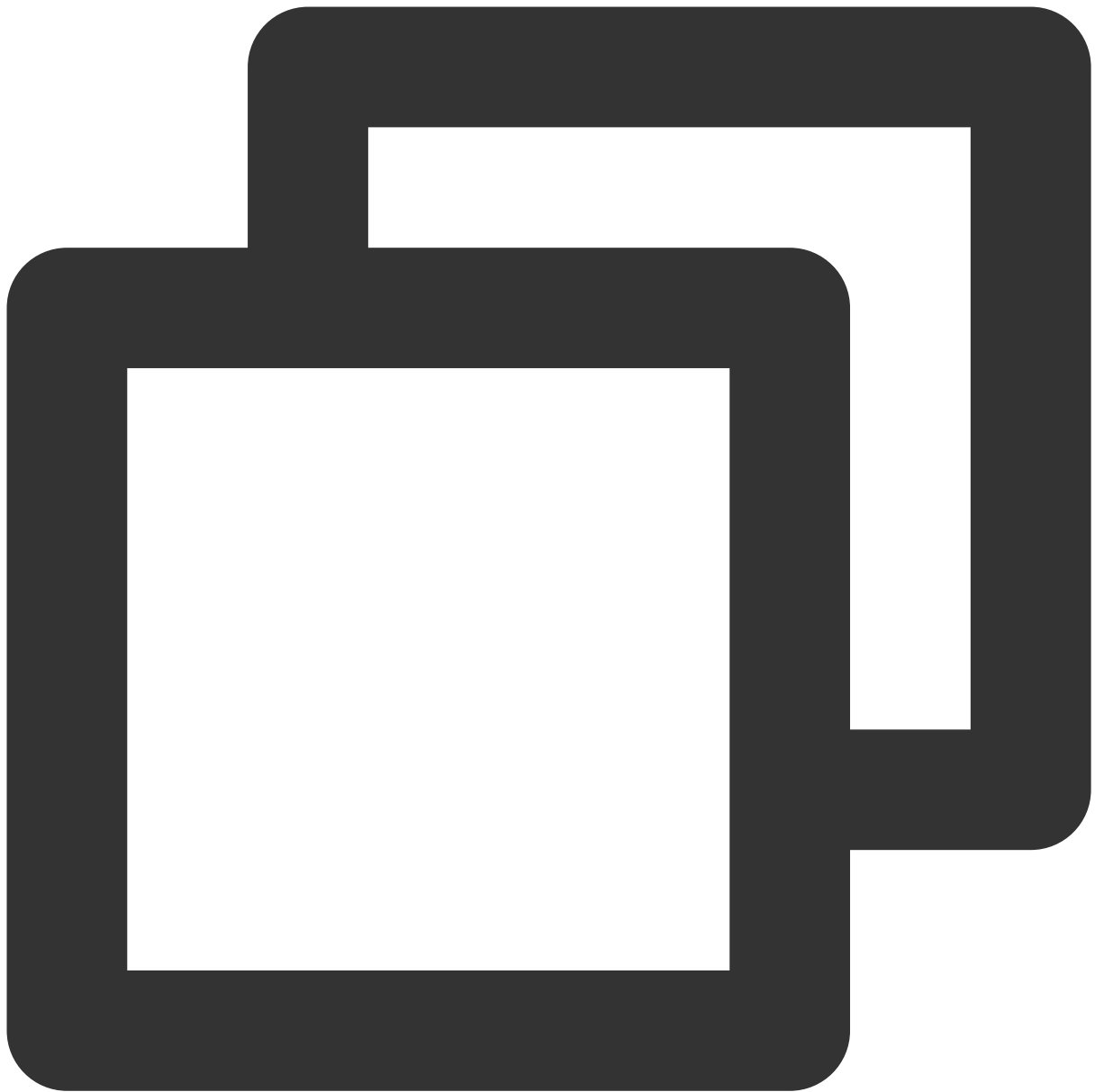
X-Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Mon, 19 Mar 2018 12:08:40

GMT。X-Date Header 里的时间和当前时间的差值不能超过15分钟。

如果是微服务 API，Header 中需要添加“X-NameSpace-Code”和“X-MicroService-Name”两个字段，通用 API 不需要添加，Demo 中默认添加了这两个字段。

## 示例代码

### Python 2.7 示例代码



```
# -*- coding: utf-8 -*-
import requests
import datetime
import hashlib
from hashlib import sha1
import hmac
import base64

GMT_FORMAT = '%a, %d %b %Y %H:%M:%S GMT'
```

```
def getSimpleSign(source, SecretId, SecretKey) :
    dateTime = datetime.datetime.utcnow().strftime(GMT_FORMAT)
    auth = "hmac id=\\\"" + SecretId + "\\\"", algorithm=\\\""hmac-sha1\\\"", headers=\\\""d
    signStr = "date: " + dateTime + "\\n" + "source: " + source
    sign = hmac.new(SecretKey, signStr, hashlib.sha1).digest()
    sign = base64.b64encode(sign)
    sign = auth + sign + "\\\""
    return sign, dateTime

SecretId = 'your SecretId' # 密钥对的 SecretId
SecretKey = 'your SecretKey' # 密钥对的 SecretKey
url = 'http://service-xxxxxx-1234567890.ap-guangzhou.apigateway.myqcloud.com/releas

#header = {}
header = { 'Host': 'service-xxxxxx-1234567890.ap-guangzhou.apigateway.myqcloud.com',
            'Accept': 'text/html, */*; q=0.01',
            'X-Requested-With': 'XMLHttpRequest',
            'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML
            'Accept-Encoding': 'gzip, deflate, sdch',
            'Accept-Language': 'zh-CN,zh;q=0.8,ja;q=0.6'
        }

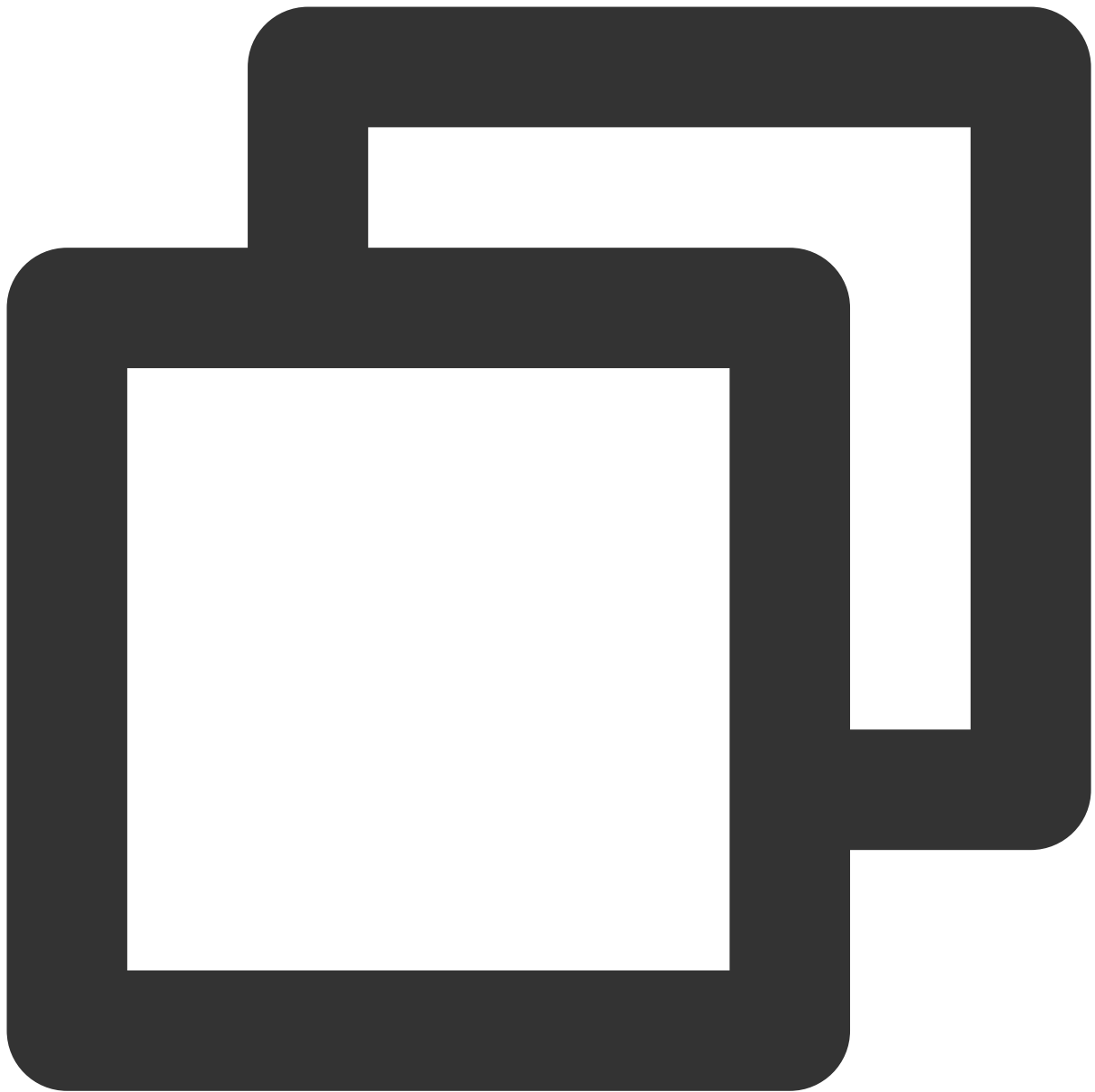
Source = 'xxxxxx' # 签名水印值, 可填写任意值
sign, dateTime = getSimpleSign(Source, SecretId, SecretKey)
header['Authorization'] = sign
header['Date'] = dateTime
header['Source'] = Source

# 如果是微服务 API, Header 中需要添加'X-NameSpace-Code'、'X-MicroService-Name'两个字段, 通
header['X-NameSpace-Code'] = 'testmic'
header['X-MicroService-Name'] = 'provider-demo'

print header

r = requests.get(url, headers=header)
print r
print r.text
```

## Python 3 示例代码



```
# -*- coding: utf-8 -*-  
import base64  
import datetime  
import hashlib  
import hmac  
  
import requests  
  
GMT_FORMAT = '%a, %d %b %Y %H:%M:%S GMT'
```

```
def getSimpleSign(source, SecretId, SecretKey):
    dateTime = datetime.datetime.utcnow().strftime(GMT_FORMAT)
    auth = "hmac id=\\\"" + SecretId + "\\\"", algorithm=\\\""hmac-sha1\\\"", headers=\\\""d
    signStr = "date: " + dateTime + "\\n" + "source: " + source
    sign = hmac.new(SecretKey.encode(), signStr.encode(), hashlib.sha1).digest()
    sign = base64.b64encode(sign).decode()
    sign = auth + sign + "\\\""
    return sign, dateTime

SecretId = 'your SecretId' # 密钥对的 SecretId
SecretKey = 'your SecretKey' # 密钥对的 SecretKey
url = 'http://service-xxxxxx-1234567890.ap-guangzhou.apigateway.myqcloud.com/releas

header = {'Host': 'service-xxxxxx-1234567890.ap-guangzhou.apigateway.myqcloud.com',
          'Accept': 'text/html, */*; q=0.01',
          'X-Requested-With': 'XMLHttpRequest',
          'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
          'Accept-Encoding': 'gzip, deflate, sdch',
          'Accept-Language': 'zh-CN,zh;q=0.8,ja;q=0.6'
          }

Source = 'xxxxxx' # 签名水印值, 可填写任意值
sign, dateTime = getSimpleSign(Source, SecretId, SecretKey)
header['Authorization'] = sign
header['Date'] = dateTime
header['Source'] = Source

r = requests.get(url, headers=header)
print(r.text)
```

# JavaScript（密钥对认证）

最近更新时间：2023-12-22 10:08:55

## 操作场景

该任务指导您使用 JavaScript 语言，通过密钥对鉴权来对您的 API 进行认证管理。

## 操作步骤

1. 在 [API 网关控制台](#)，创建一个 API，选择鉴权类型为“密钥对鉴权”（参考 [创建 API 概述](#)）。
2. 将 API 所在服务发布至发布环境（参考 [服务发布与下线](#)）。
3. 在控制台密钥管理界面创建密钥对。
4. 在控制台使用计划界面创建使用计划，并将使用计划与已创建的密钥对绑定（参考 [使用计划示例](#)）。
5. 将使用计划与 API 或 API 所在服务进行绑定。
6. 参考 [示例代码](#)，使用 JavaScript 语言生成签名内容。

## 注意事项

最终发送的 HTTP 请求内至少包含两个 Header：Date 和 X-Date 二选一以及 Authorization，可以包含更多 header。

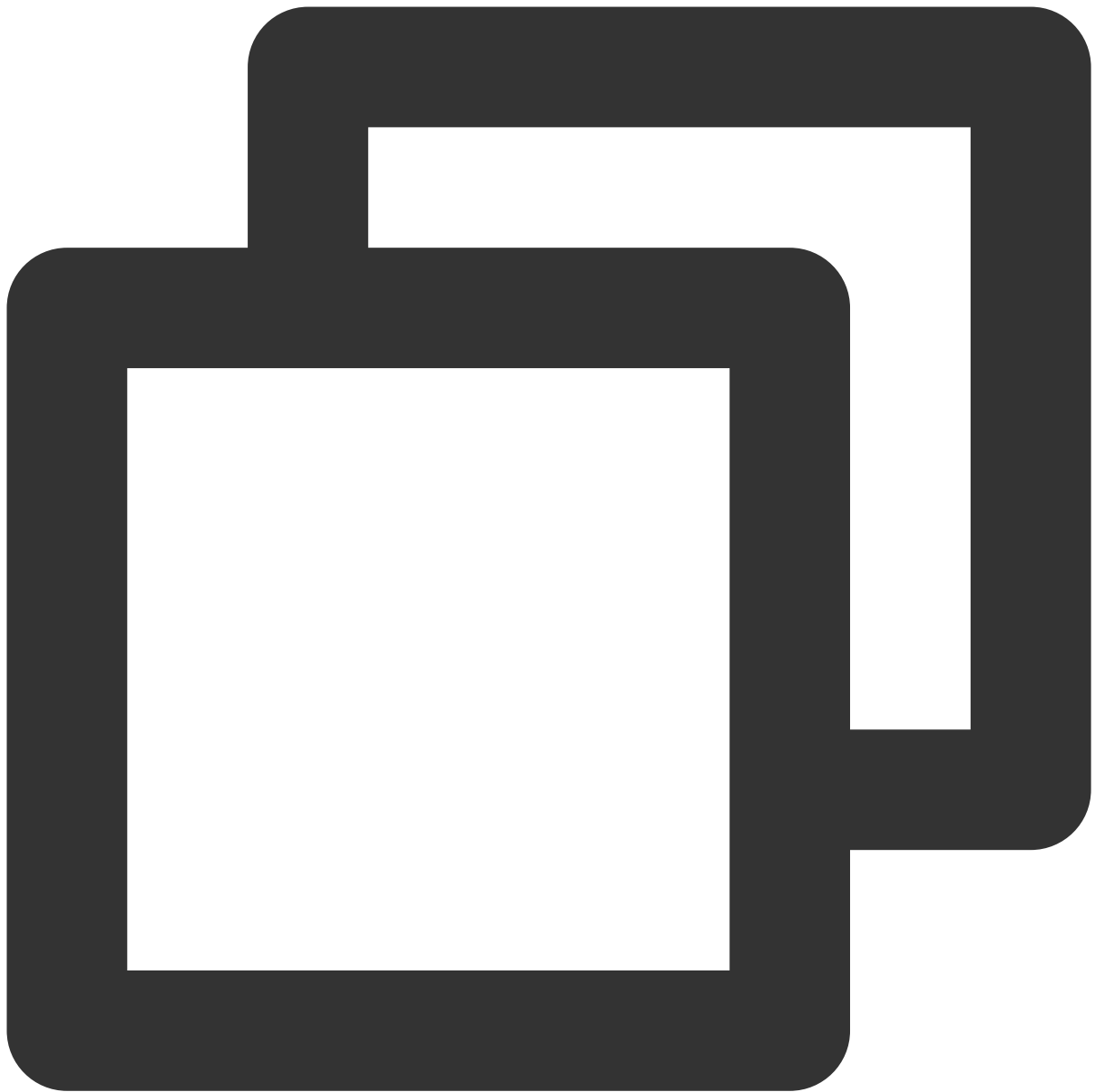
如果使用 Date Header，服务端将不会校验时间；如果使用 X-Date Header，服务端将校验时间。

Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Fri, 09 Oct 2015 00:00:00 GMT。

X-Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Mon, 19 Mar 2018 12:08:40 GMT。X-Date Header 里的时间和当前时间的差值不能超过15分钟。

如果是微服务 API，Header 中需要添加“X-NameSpace-Code”和“X-MicroService-Name”两个字段，通用 API 不需要添加，Demo 中默认添加了这两个字段。

## 示例代码



```
/*
used...
<script src="js/vue.js"></script>
<script src="js/axios.js"></script>
<script src="js/qs.js"></script>
<script src="js/crypto-js/crypto-js.js"></script>
*/
var nowDate = new Date();

var dateTime = nowDate.toUTCString();
//dateTime = "Mon, 19 Mar 2018 12:00:44 GMT"
```

```
var SecretId = 'your SecretId'; // 密钥对的 SecretId
var SecretKey = 'your SecretKey'; // 密钥对的 SecretKey
var source = 'xxxxxx'; // 签名水印值, 可填写任意值
var auth = "hmac id=\\\"" + SecretId + "\\\"", algorithm=\\\""hmac-sha1\\\"", headers=\\\""x
var signStr = "x-date: " + dateTime + "\\n" + "source: " + source;
console.log(signStr)
var sign = CryptoJS.HmacSHA1(signStr, SecretKey)
console.log(sign.toString())
sign = CryptoJS.enc.Base64.stringify(sign)
sign = auth + sign + "\\\""
console.log(sign)
console.log(dateTime)

var instance = axios.create({
  baseURL: 'http://service-xxxxxxx-1234567890.ap-guangzhou.apigateway.myqcloud.c
  timeout: 5000,
  headers: {
    "Source":source,
    "X-Date":dateTime,
    "Authorization":sign

    // 如果是微服务 API, Header 中需要添加'X-NameSpace-Code'、'X-MicroService
    "X-NameSpace-Code": "testmic",
    "X-MicroService-Name": "provider-demo",
  },
  withCredentials: true
});

instance.get()
.then(function (response) {
  console.log(response);
})
.catch(function (error) {
  console.log(error);
});;
```

# PHP（密钥对认证）

最近更新时间：2023-12-22 10:09:05

## 操作场景

该任务指导您使用 PHP 语言，通过密钥对鉴权来对您的 API 进行认证管理。

## 操作步骤

1. 在 [API 网关控制台](#)，创建一个 API，选择鉴权类型为“密钥对鉴权”（参考 [创建 API 概述](#)）。
2. 将 API 所在服务发布至发布环境（参考 [服务发布与下线](#)）。
3. 在控制台密钥管理界面创建密钥对。
4. 在控制台使用计划界面创建使用计划，并将使用计划与已创建的密钥对绑定（参考 [使用计划示例](#)）。
5. 将使用计划与 API 或 API 所在服务进行绑定。
6. 参考 [示例代码](#)，使用 PHP 语言生成签名内容。

## 注意事项

最终发送的 HTTP 请求内至少包含两个 Header：Date 和 X-Date 二选一以及 Authorization，可以包含更多 header。

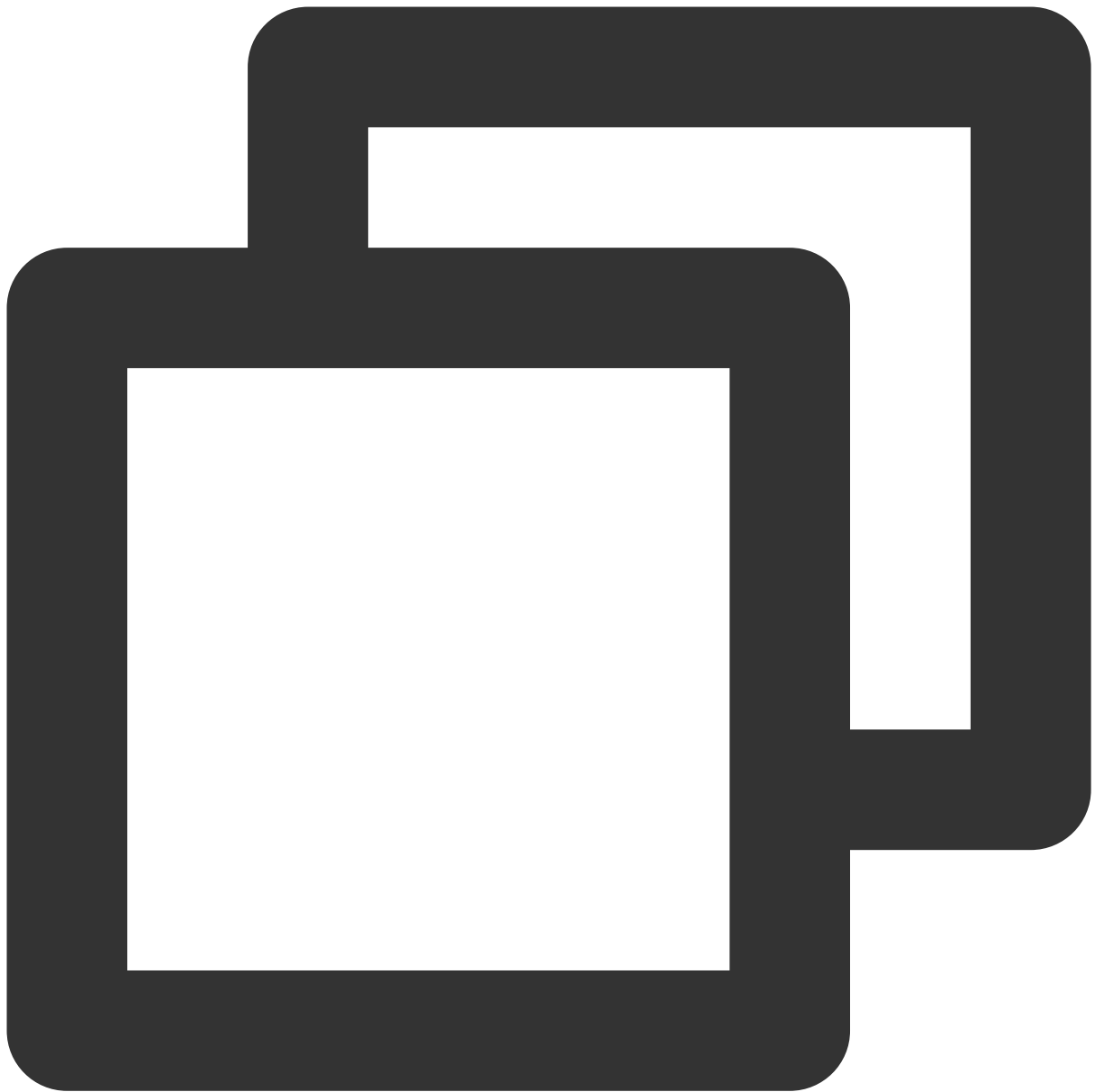
如果使用 Date Header，服务端将不会校验时间；如果使用 X-Date Header，服务端将校验时间。

Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Fri, 09 Oct 2015 00:00:00 GMT。

X-Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Mon, 19 Mar 2018 12:08:40 GMT。X-Date Header 里的时间和当前时间的差值不能超过15分钟。

如果是微服务 API，Header 中需要添加“X-NameSpace-Code”和“X-MicroService-Name”两个字段，通用 API 不需要添加，Demo 中默认添加了这两个字段。

## 示例代码



```
<?php

$dateTime = gmdate("D, d M Y H:i:s T");
$SecretId = 'your SecretId'; # 密钥对的 SecretId
$SecretKey = 'your SecretKey'; # 密钥对的 SecretKey
$srcStr = "date: ".$dateTime."\n"."source: ".$xxxxxx"; # 签名水印值, 可填写任意值
$Authen = 'hmac id="'.$SecretId.'"', algorithm="hmac-sha1", headers="date source", s
$signStr = base64_encode(hash_hmac('sha1', $srcStr, $SecretKey, true));
# echo $signStr;
$Authen = $Authen.$signStr."\\\\";
echo $Authen;
```

```
# echo '</br>';

$url = 'http://service-xxxxxxx-1234567890.ap-guangzhou.apigateway.myqcloud.com/rel
$headers = array(
    'Host:service-xxxxxxx-1234567890.ap-guangzhou.apigateway.myqcloud.com', # 用户
    'Accept:text/html, */*; q=0.01',
    'Source: xxxxxx',
    'Date: '.$dateTime,
    'Authorization: '.$Authen,
    'X-Requested-With: XMLHttpRequest',
    # 'Accept-Encoding: gzip, deflate, sdch',

    # 如果是微服务 API, Header 中需要添加'X-NameSpace-Code'、'X-MicroService-Name'两个字
    'X-NameSpace-Code: testmic',
    'X-MicroService-Name: provider-demo'
);
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL,$url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_TIMEOUT, 60);
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "GET");

$data = curl_exec($ch);

if (curl_errno($ch)) {
    print "Error: " . curl_error($ch);
} else {
    # Show me the result
    var_dump($data);
    curl_close($ch);
}

?>
```

# C++（密钥对认证）

最近更新时间：2023-12-22 10:09:20

## 操作场景

该任务指导您使用 C++ 语言，通过密钥对鉴权来对您的 API 进行认证管理。

## 操作步骤

1. 在 [API 网关控制台](#)，创建一个 API，选择鉴权类型为“密钥对”（参考 [创建 API 概述](#)）。
2. 将 API 所在服务发布至发布环境（参考 [服务发布与下线](#)）。
3. 在控制台密钥管理界面创建密钥对。
4. 在控制台使用计划界面创建使用计划，并将使用计划与已创建的密钥对绑定（参考 [使用计划示例](#)）。
5. 将使用计划与 API 或 API 所在服务进行绑定。
6. 参考 [示例代码](#)，使用 C++ 语言生成签名内容。

## 环境依赖

本 Demo 中使用 libcurl 发起 HTTP 请求,故编译机器需要安装 libcurl 库。

## 注意事项

最终发送的 HTTP 请求内至少包含两个 Header：Date 和 X-Date 二选一以及 Authorization，可以包含更多 Header。

如果使用 Date Header，服务端将不会校验时间；如果使用 X-Date Header，服务端将校验时间。

Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Fri, 09 Oct 2015 00:00:00 GMT。

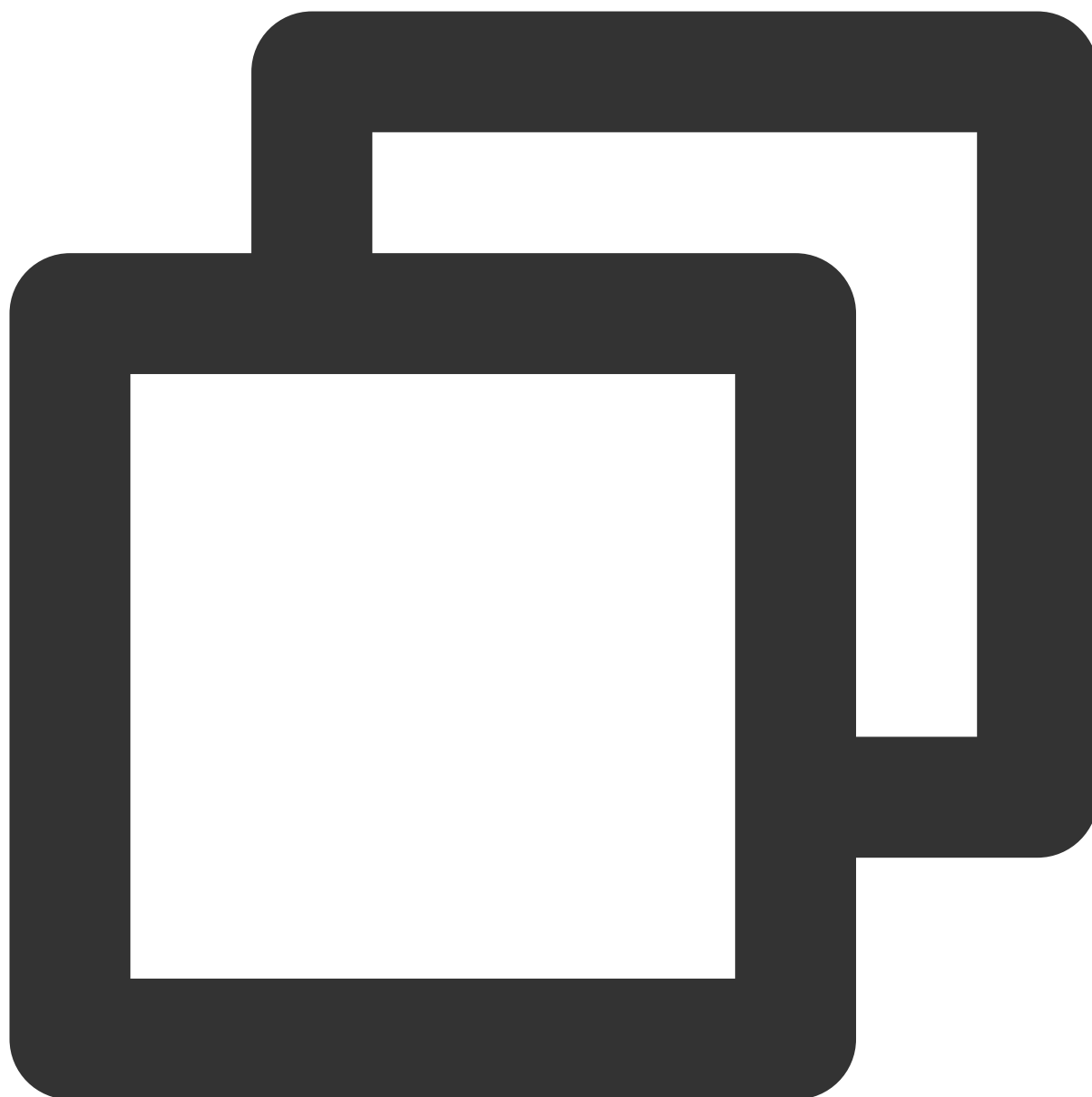
X-Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Mon, 19 Mar 2018 12:08:40

GMT。X-Date Header 里的时间和当前时间的差值不能超过15分钟。

如果是微服务 API，Header 中需要添加“X-NameSpace-Code”和“X-MicroService-Name”两个字段，通用 API 不需要添加，Demo 中默认添加了这两个字段。

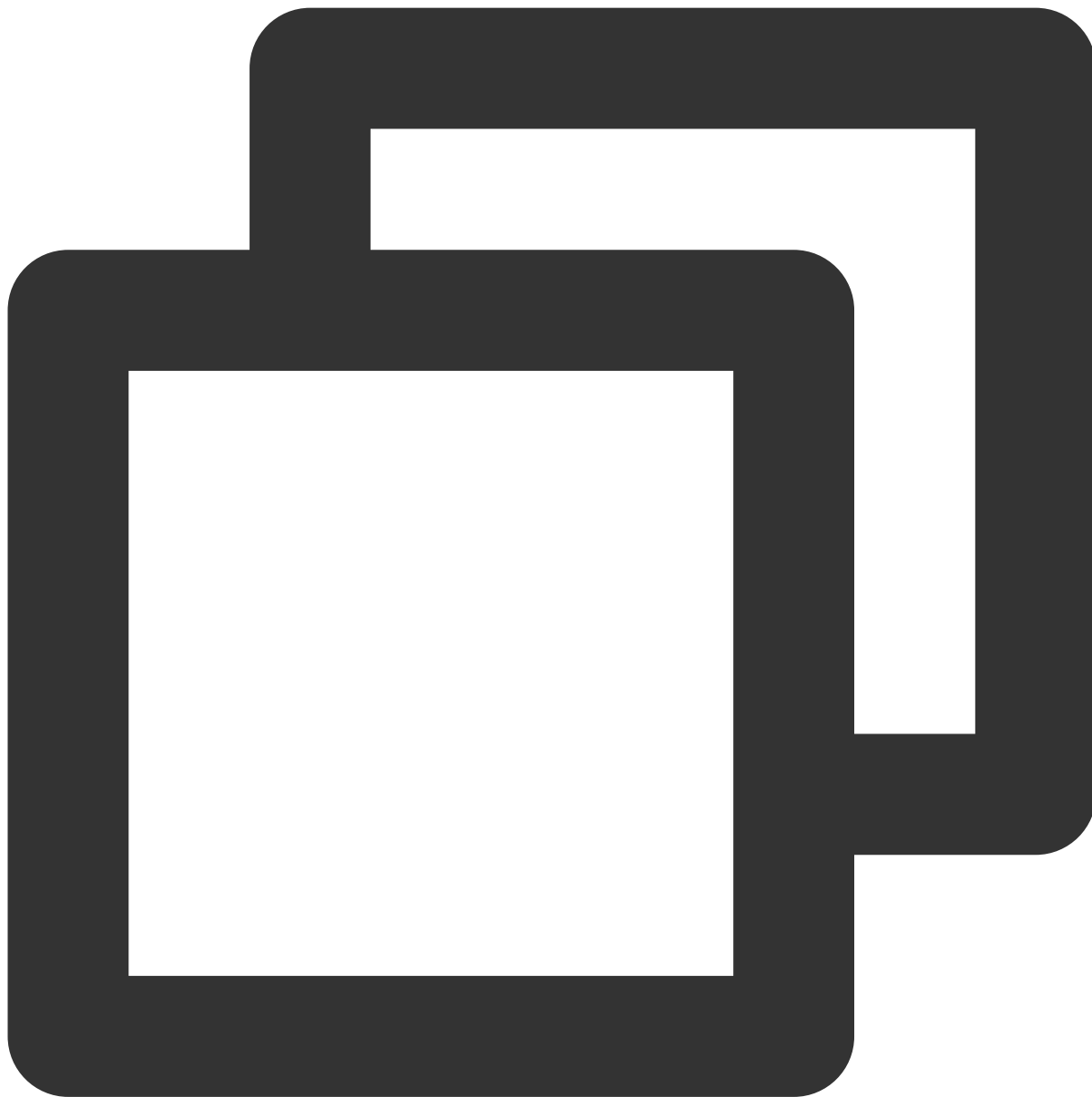
## 目录结构

本 Demo 中共包含 7 个文件，目录结构如下：



```
├─AuthenticationDemo.cpp
├─request.cpp
├─base64.h
├─base64.cpp
├─hmac.h
├─sha1.h
└─sha1.cpp
```

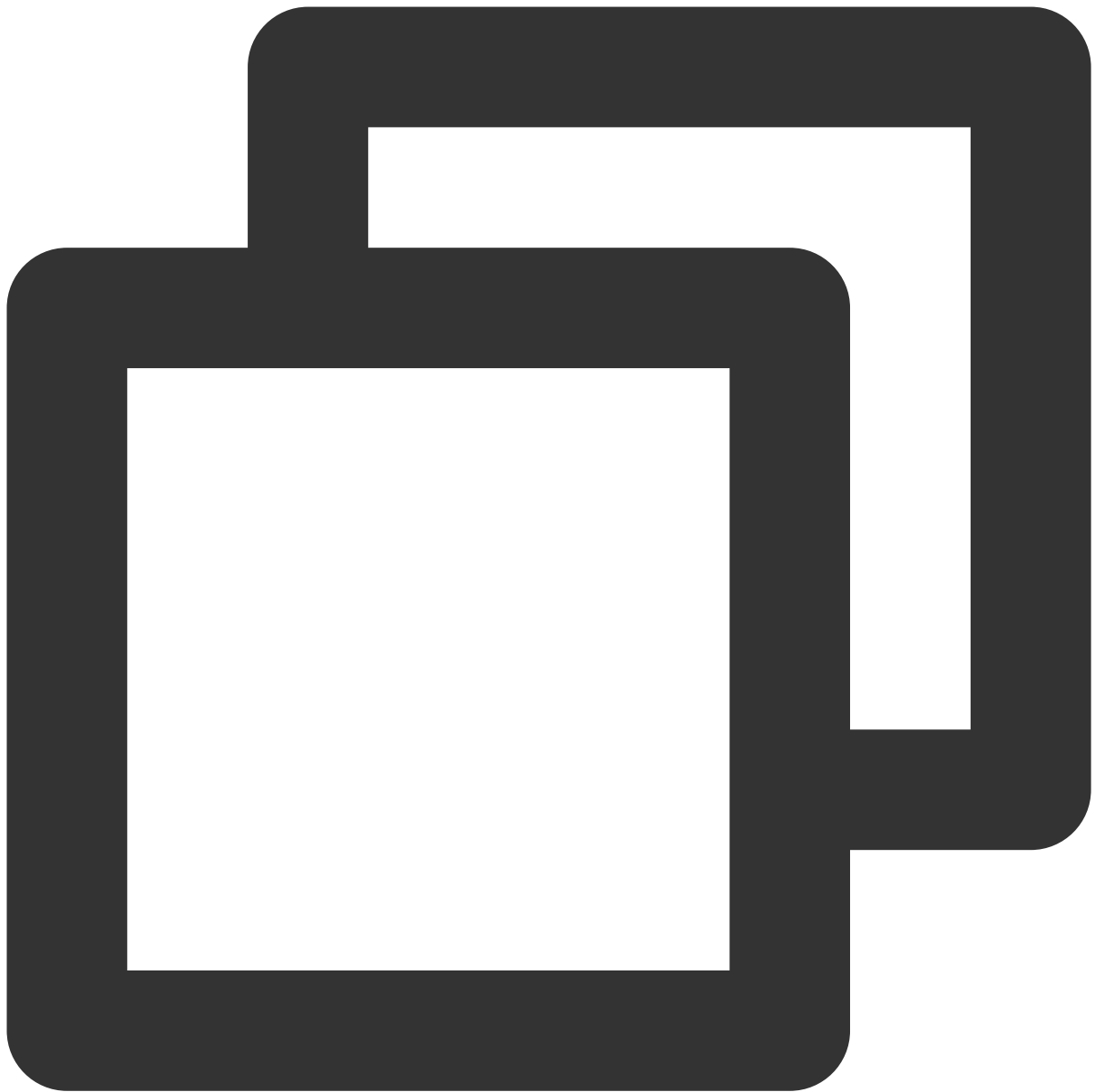
## 编译命令



```
g++ -o AuthenticationDemo AuthenticationDemo.cpp request.cpp base64.cpp sha1.cpp -l
```

## 示例代码

**AuthenticationDemo.cpp**



```
/*本Demo中使用libcurl发起http请求,故编译机器需要安装libcurl库*/  
/*编译命令 g++ -o AuthenticationDemo AuthenticationDemo.cpp request.cpp base64.cpp sl  
  
#include <iostream>  
#include <stdio.h>  
#include "hmac.h"  
#include "sha1.h"  
#include "base64.h"  
  
extern void get_request(const string &defaultDomain, const string &source, const st  
extern void post_request(const string &defaultDomain, const string &source, const s
```

```
using namespace std;

void GetGmtTime(string &szGmtTime)
{
    time_t rawTime;
    struct tm* timeInfo;
    char szTemp[30]={0};
    time(&rawTime);
    timeInfo = gmtime(&rawTime);
    strftime(szTemp,sizeof(szTemp),"%a, %d %b %Y %H:%M:%S GMT",timeInfo);
    szGmtTime = szTemp;
}

int calcAuthorization(const string &source, const string &secretId, const string &secretKey)
{
    GetGmtTime(dateTime);
    sign = "x-date: " + dateTime + "\\nsource: " + source;
    sign = hmac<SHA1>(sign, secretKey);
    string binDight;
    HexToBin(sign , binDight);
    BinToBase64(binDight , sign);
    char tempauth[1024] = {0};
    snprintf(tempauth,sizeof(tempauth)-1,"hmac id=\\\"%s\\\"", algorithm=\\\"hmac-sha1\\")
    sign = tempauth;

    return 0;
}

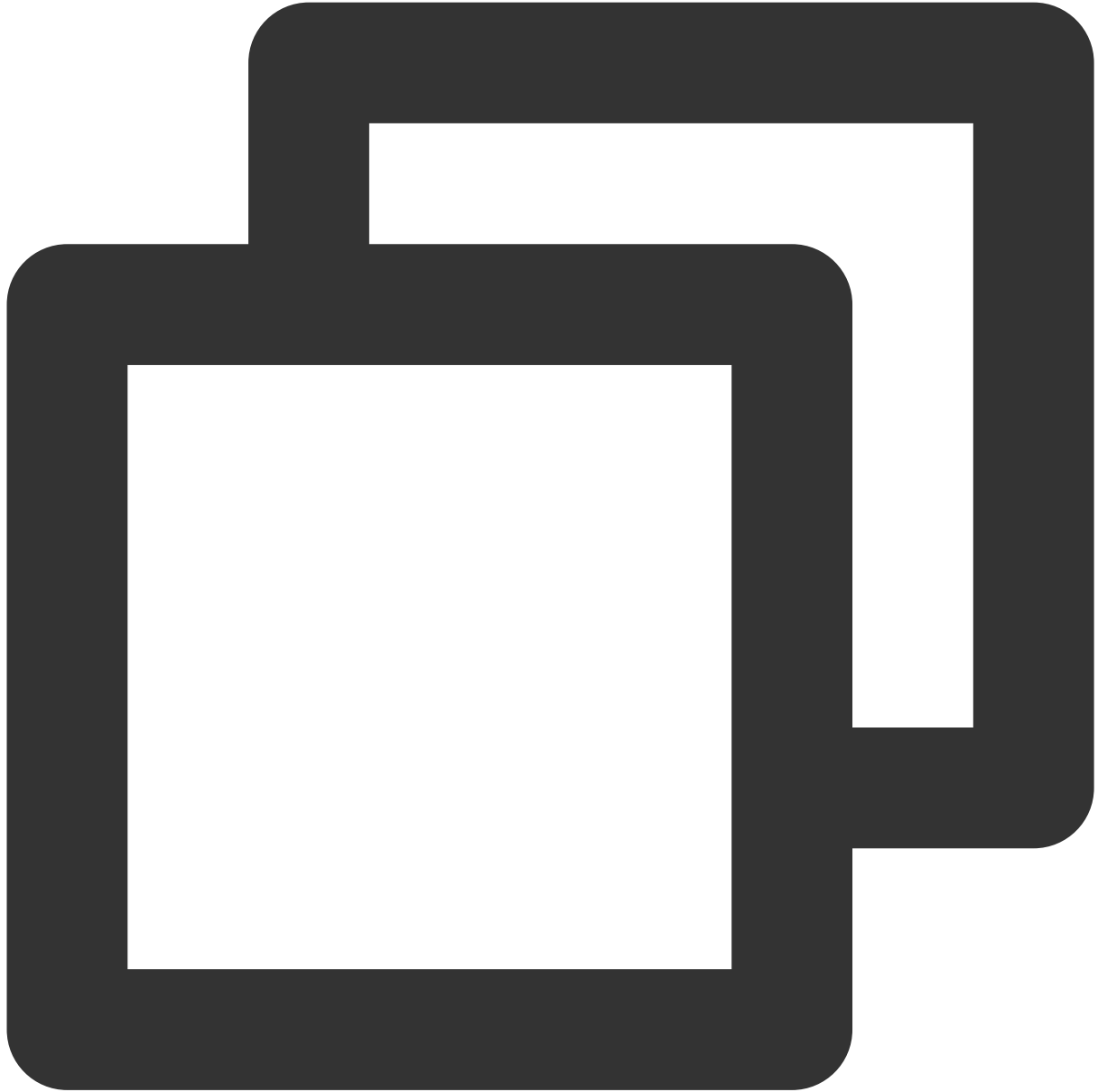
/*下面代码中secretId,secretKey,defaultDomain,reqUrl根据业务实际情况填写即可*/

int main()
{
    const string secretId = "your secretId";// 密钥对的 SecretId
    const string secretKey = "your secretKey";// 密钥对的 SecretKey
    const string source = "xxxxxxx"; // 签名水印值, 可填写任意值
    string sign, dateTime;
    calcAuthorization(source, secretId, secretKey, sign, dateTime);
    const string defaultDomain = "service-xxxxxxx-1234567890.ap-guangzhou.apigateway.tencentcloudapi.com";
    const string reqUrl = "https://service-xxxxxxx-1234567890.ap-guangzhou.apigateway.tencentcloudapi.com";

    get_request(defaultDomain, source, dateTime, sign, reqUrl);
    //post_request(defaultDomain, source, dateTime, sign, reqUrl);

    return 0;
}
```

request.cpp



```
#include <iostream>
#include <cstring>
#include "curl/curl.h"

using namespace std;

size_t req_reply(void *ptr, size_t size, size_t nmemb, void *stream)
{
```

```
((std::string*)stream)->append((char*)ptr, size*nmemb);
return size * nmemb;
}

void get_request(const string &defaultDomain, const string &source, const string &d
{
    CURL* curl = curl_easy_init();
    if (curl)
    {
        curl_easy_setopt(curl, CURLOPT_URL, reqUrl.c_str());
        curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0L);
        struct curl_slist * slist = NULL;
        slist = curl_slist_append(slist, "Accept:/*/*");
        slist = curl_slist_append(slist, "Accept-Charset:utf-8;");
        string headDomain = "Host:" + defaultDomain;
        slist = curl_slist_append(slist, headDomain.c_str());
        string headSource = "Source:" + source;
        slist = curl_slist_append(slist, headSource.c_str());
        string headDatetime = "X-Date:" + dateTime;
        slist = curl_slist_append(slist, headDatetime.c_str());
        string headAuthorization = "Authorization:" + sign;
        slist = curl_slist_append(slist, headAuthorization.c_str());
        // 如果是微服务 API, Header 中需要添加'X-NameSpace-Code'、'X-MicroService-Name'
        slist = curl_slist_append(slist, "x-NameSpace-Code:testmic");
        slist = curl_slist_append(slist, "x-MicroService-Name:provider-demo");
        curl_easy_setopt(curl, CURLOPT_HTTPHEADER, slist);
        curl_easy_setopt(curl, CURLOPT_CONNECTTIMEOUT, 5);
        curl_easy_setopt(curl, CURLOPT_TIMEOUT, 5);
        curl_easy_setopt(curl, CURLOPT_READFUNCTION, NULL);
        std::string response_data;
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response_data);
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, req_reply);

        CURLcode res = curl_easy_perform(curl);
        if (res != CURLE_OK)
        {
            fprintf(stderr, "curl_easy_perform() failed: %s\\n",
                curl_easy_strerror(res));
        }
        else
        {
            // get response code
            long response_code;
            curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &response_code);
            printf("response code %d \\n", response_code);
            printf("response data : %s\\n ", response_data.c_str());
        }
    }
}
```

```
        curl_slist_free_all(slist);
        curl_easy_cleanup(curl);
    }
    curl_global_cleanup();
}

void post_request(const string &defaultDomain, const string &source, const string &
{
    CURL* curl = curl_easy_init();
    if (curl)
    {
        curl_easy_setopt(curl, CURLOPT_URL, reqUrl.c_str());
        curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0L);
        curl_easy_setopt(curl, CURLOPT_POST, 1);
        struct curl_slist * slist = NULL;
        slist = curl_slist_append(slist, "Accept:*/");
        slist = curl_slist_append(slist, "Accept-Charset:utf-8;");
        string headDomain = "Host:" + defaultDomain;
        slist = curl_slist_append(slist, headDomain.c_str());
        string headSource = "Source:" + source;
        slist = curl_slist_append(slist, headSource.c_str());
        string headDatetime = "X-Date:" + dateTime;
        slist = curl_slist_append(slist, headDatetime.c_str());
        string headAuthorization = "Authorization:" + sign;
        slist = curl_slist_append(slist, headAuthorization.c_str());
        // 如果是微服务 API, Header 中需要添加'X-NameSpace-Code'、'X-MicroService-Name'
        slist = curl_slist_append(slist, "x-NameSpace-Code:testmic");
        slist = curl_slist_append(slist, "x-MicroService-Name:provider-demo");
        curl_easy_setopt(curl, CURLOPT_HTTPHEADER, slist);

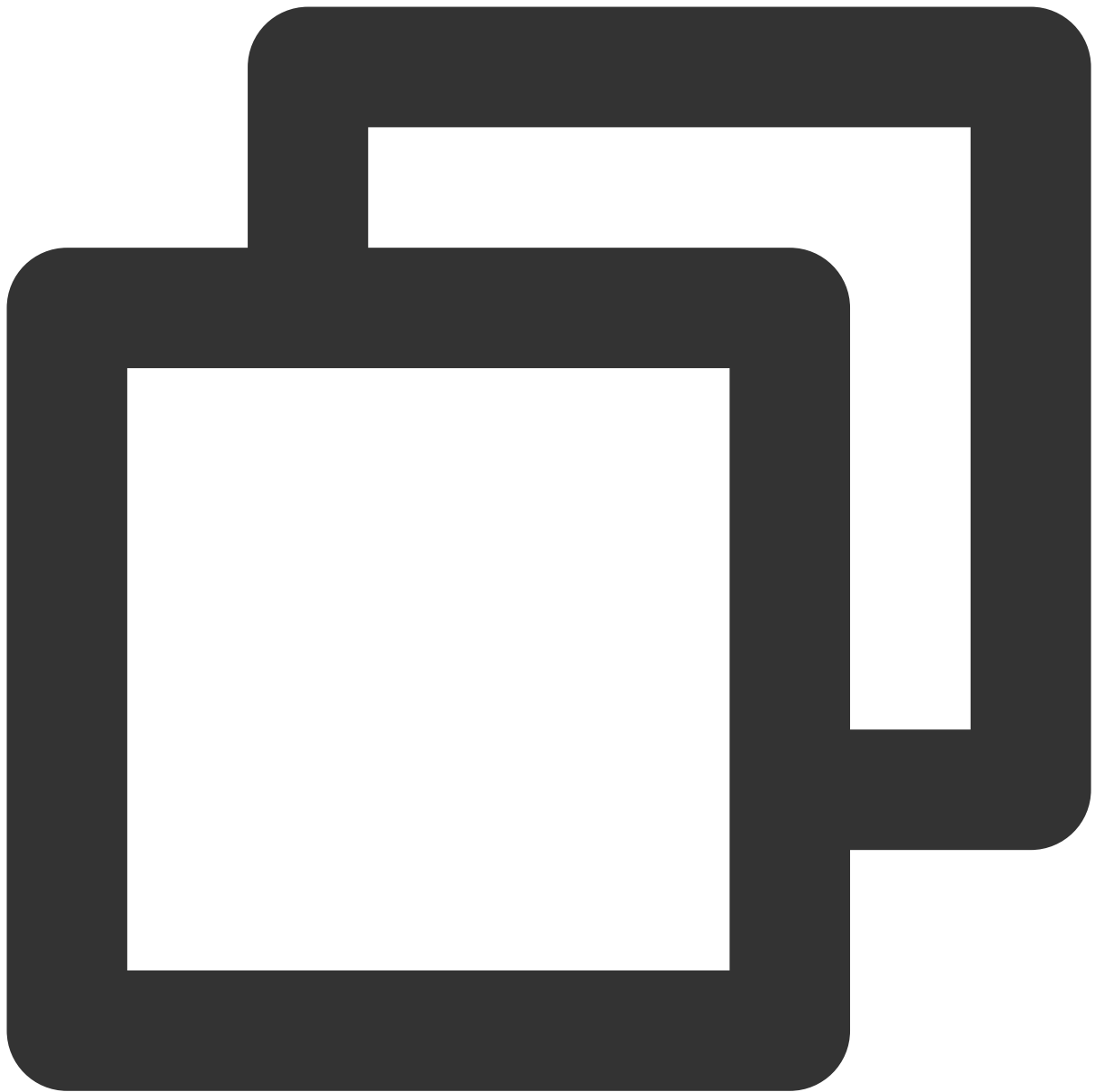
        // set body
        std::string body = "{\\
            \\\"title\\\":\\\"post title\\\",\\
            \\\"body\\\" : \\\"post body\\\",\\
            \\\"userId\\\" : 1}";
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, body.c_str());

        curl_easy_setopt(curl, CURLOPT_CONNECTTIMEOUT, 5);
        curl_easy_setopt(curl, CURLOPT_TIMEOUT, 5);
        curl_easy_setopt(curl, CURLOPT_READFUNCTION, NULL);
        std::string response_data;
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response_data);
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, req_reply);

        CURLcode res = curl_easy_perform(curl);
        if (res != CURLE_OK)
        {
```

```
        fprintf(stderr, "curl_easy_perform() failed: %s\\n",
        curl_easy_strerror(res));
    }
    else
    {
        // get response code
        long response_code;
        curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &response_code);
        printf("response code %d \\n", response_code);
        printf("response data : %s\\n ", response_data.c_str());
    }
    curl_slist_free_all(slist);
    curl_easy_cleanup(curl);
}
curl_global_cleanup();
}
```

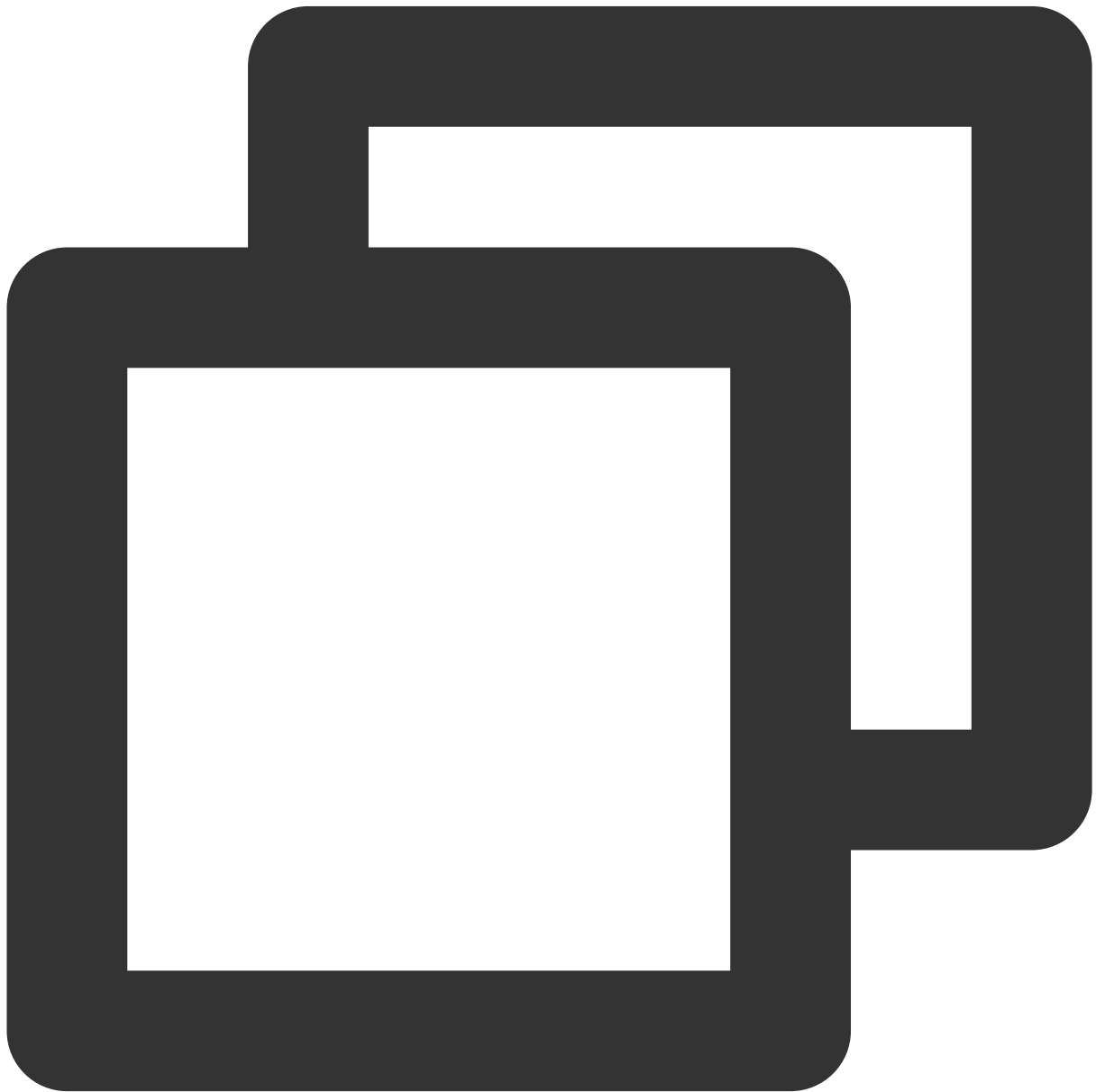
**base64.h**



```
//Base64编码表
#include<string>
using namespace std;
const char Base64EncodeMap[64] =
{
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
    'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
    'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
    'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f',
    'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
    'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
```

```
'w', 'x', 'y', 'z', '0', '1', '2', '3',  
'4', '5', '6', '7', '8', '9', '+', '/'  
};  
  
int BinToDecInt(string strBin);  
void BinToBase64(string binStr , string &base64Str);  
void HexToBin(string hexDight , string& binDight);
```

### base64.cpp



```
#include "base64.h"
```

```
int BinToDecInt(string strBin){
    int num = 0;
    int b = 0;
    for(int i = 0; i < strBin.length() ;i++){
        num = num * 2;
        b = static_cast<int>(strBin[i]-'0');
        num = num + b;
    }
    return num;
}
```

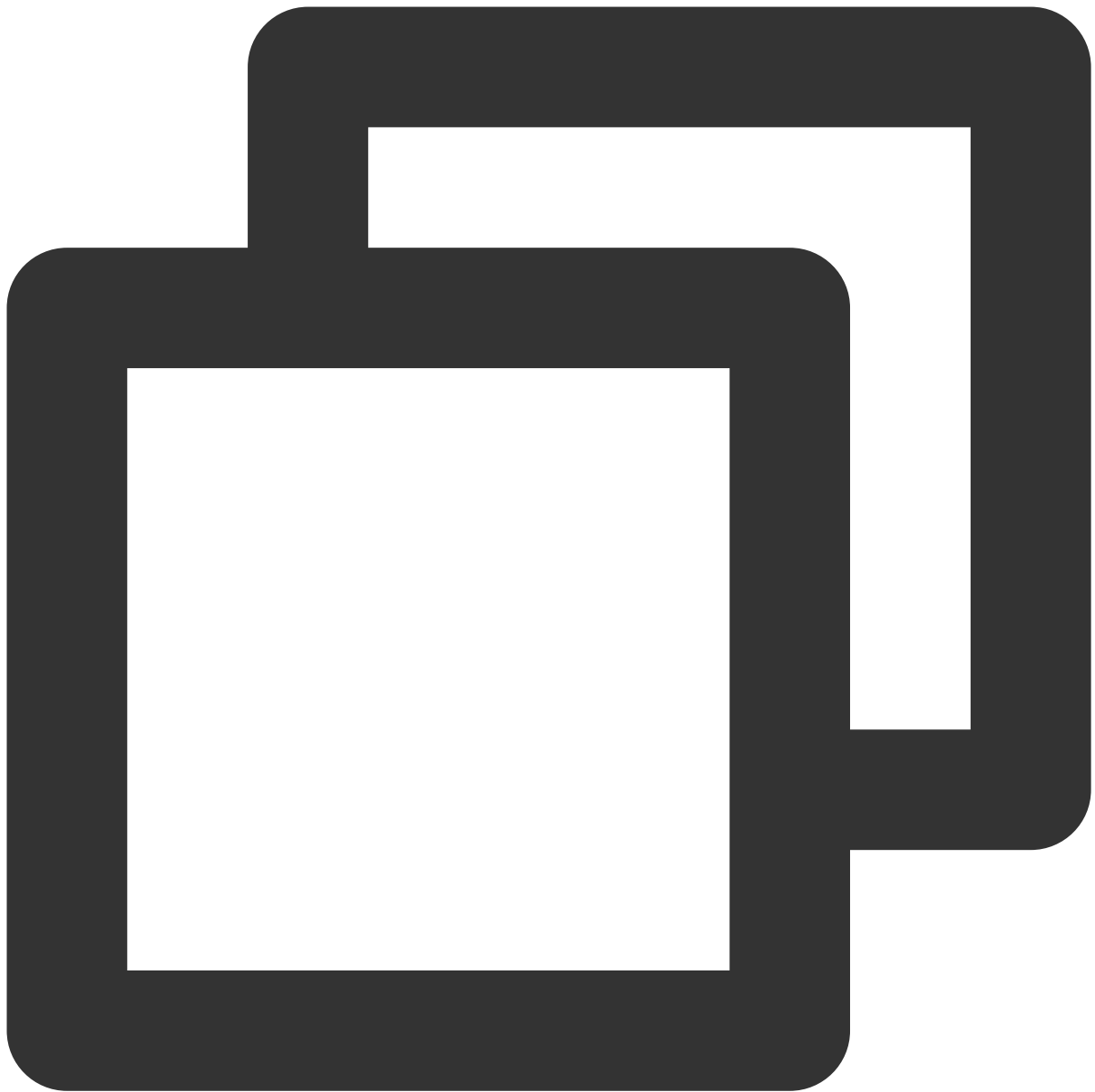
```
void BinToBase64(string binStr , string &base64Str)
{
    while(binStr.length() % 6 != 0){
        binStr = binStr + "0";
    }
    base64Str = "";
    string tmp = "";
    int index = 0;
    int num = 0;
    while(index < binStr.length()){
        tmp = binStr.substr(index , 6);
        index = index + 6;
        num = BinToDecInt(tmp);
        base64Str = base64Str + Base64EncodeMap[num];
    }
    base64Str = base64Str + "=";
}
```

```
void HexToBin(string hexDight , string& binDight){
    binDight = "";
    int f = 0,c = 0;
    char e;
    for(int f = 0; f < hexDight.length() ; f++){
        e = hexDight[f];
        if(e >= 'a' && e <= 'f'){
            int a = static_cast<int>(e-'a'+10);
            switch(a){
                case 10 : binDight = binDight + "1010";
                    break;
                case 11 : binDight = binDight + "1011";
                    break;
                case 12 : binDight = binDight + "1100";
                    break;
            }
        }
    }
}
```

```
        case 13 : binDight = binDight + "1101";
                break;
        case 14 : binDight = binDight + "1110";
                break;
        case 15 : binDight = binDight + "1111";
                break;
    }
}
else if( e >= '0' && e <= '9'){
    int b = static_cast<int>(e-'0');
    if(f == 0){
        switch(b){
            case 0: binDight = binDight + "0000";
                    break;
            case 1: binDight = binDight + "0001";
                    break;
            case 2: binDight = binDight + "0010";
                    break;
            case 3: binDight = binDight + "0011";
                    break;
            case 4: binDight = binDight + "0100";
                    break;
            case 5: binDight = binDight + "0101";
                    break;
            case 6: binDight = binDight + "0110";
                    break;
            case 7: binDight = binDight + "0111";
                    break;
            case 8: binDight = binDight + "1000";
                    break;
            case 9: binDight = binDight + "1001";
                    break;
        }
    }
    else{
        switch(b){
            case 0 : binDight = binDight + "0000";
                    break;
            case 1: binDight = binDight + "0001";
                    break;
            case 2: binDight = binDight + "0010";
                    break;
            case 3: binDight = binDight + "0011";
                    break;
            case 4: binDight = binDight + "0100";
                    break;
            case 5: binDight = binDight + "0101";
```

```
        break;
    case 6: binDight = binDight + "0110";
        break;
    case 7: binDight = binDight + "0111";
        break;
    case 8: binDight = binDight + "1000";
        break;
    case 9: binDight = binDight + "1001";
        break;
    }
}
}
}
```

**hmac.h**



```
#pragma once

#include <string>
#include <cstring>

/// compute HMAC hash of data and key using MD5, SHA1 or SHA256
template <typename HashMethod>
std::string hmac(const void* data, size_t numDataBytes, const void* key, size_t num
{
    unsigned char usedKey[HashMethod::BlockSize] = {0};
```

```
if (numKeyBytes <= HashMethod::BlockSize)
{
    memcpy(usedKey, key, numKeyBytes);
}
else
{
    HashMethod keyHasher;
    keyHasher.add(key, numKeyBytes);
    keyHasher.getHash(usedKey);
}

for (size_t i = 0; i < HashMethod::BlockSize; i++)
    usedKey[i] ^= 0x36;

unsigned char inside[HashMethod::HashBytes];
HashMethod insideHasher;
insideHasher.add(usedKey, HashMethod::BlockSize);
insideHasher.add(data, numDataBytes);
insideHasher.getHash(inside);

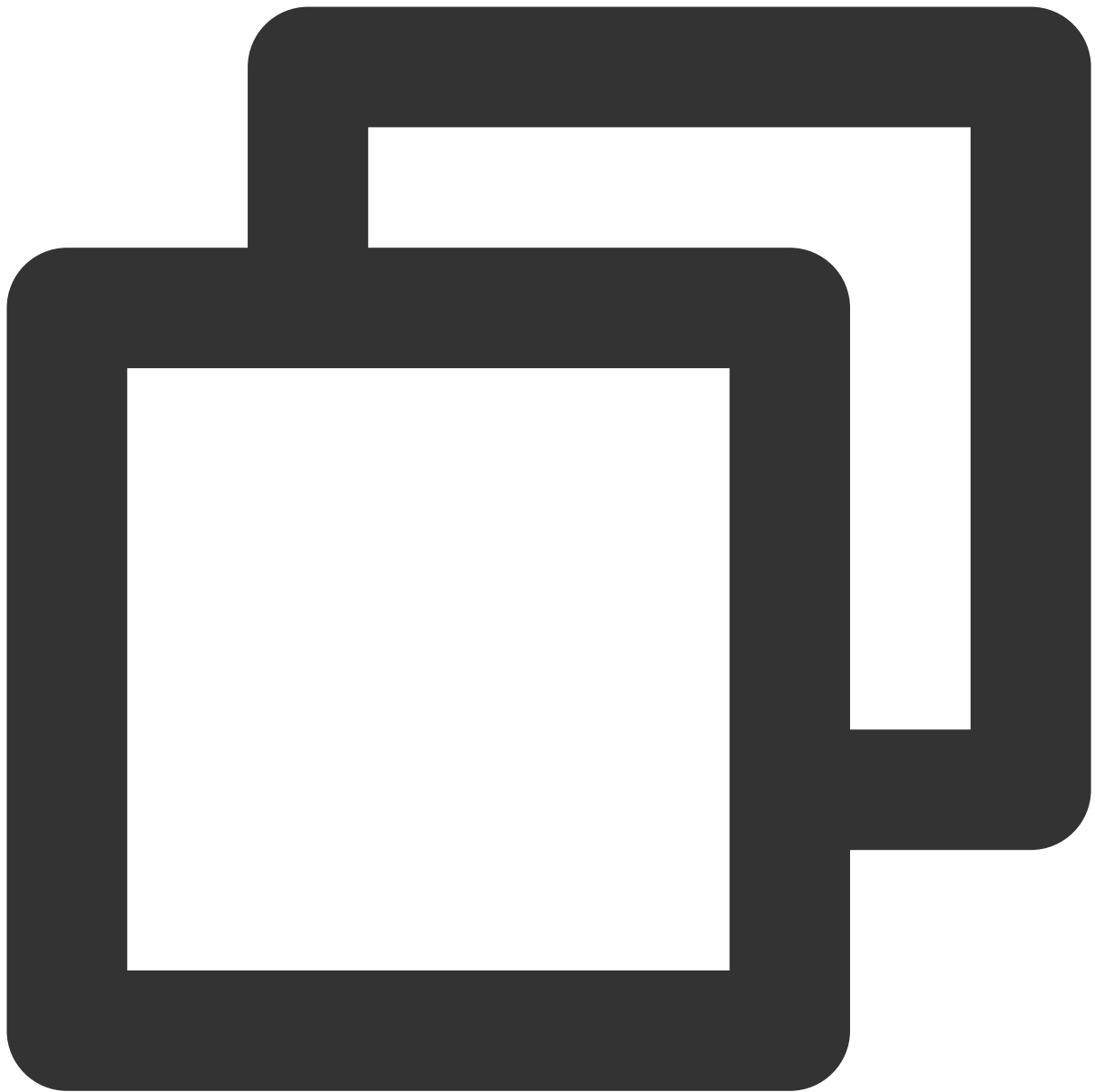
for (size_t i = 0; i < HashMethod::BlockSize; i++)
    usedKey[i] ^= 0x5C ^ 0x36;

HashMethod finalHasher;
finalHasher.add(usedKey, HashMethod::BlockSize);
finalHasher.add(inside, HashMethod::HashBytes);

return finalHasher.getHash();
}

template <typename HashMethod>
std::string hmac(const std::string& data, const std::string& key)
{
    return hmac<HashMethod>(data.c_str(), data.size(), key.c_str(), key.size());
}
```

## sha1.h



```
#pragma once

#include <string>

#ifdef _MSC_VER
// Windows
typedef unsigned __int8  uint8_t;
typedef unsigned __int32 uint32_t;
typedef unsigned __int64 uint64_t;
#else
// GCC
```

```
#include <stdint.h>
#ifdef

class SHA1 //: public Hash
{
public:
    enum { BlockSize = 512 / 8, HashBytes = 20 };

    SHA1();

    std::string operator()(const void* data, size_t numBytes);

    std::string operator()(const std::string& text);

    void add(const void* data, size_t numBytes);

    std::string getHash();

    void getHash(unsigned char buffer[HashBytes]);

    void reset();

private:
    void processBlock(const void* data);

    void processBuffer();

    uint64_t m_numBytes;

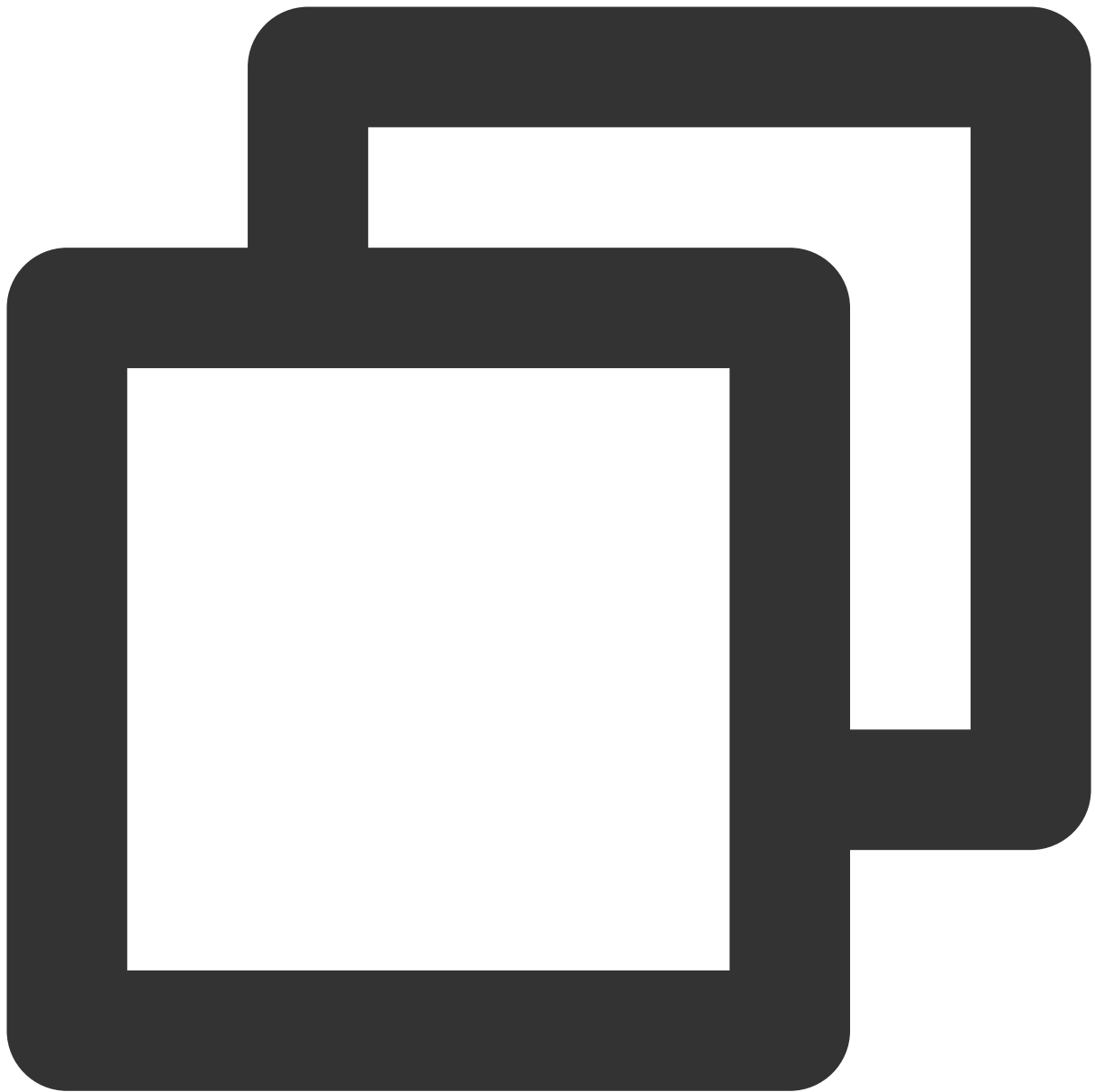
    size_t    m_bufferSize;

    uint8_t    m_buffer[BlockSize];

    enum { HashValues = HashBytes / 4 };

    uint32_t m_hash[HashValues];
};
```

**sha1.cpp**



```
#include "sha1.h"

#ifdef _MSC_VER
#include <endian.h>
#endif

SHA1::SHA1()
{
    reset();
}
```

```
void SHA1::reset()
{
    m_numBytes    = 0;
    m_bufferSize  = 0;

    m_hash[0] = 0x67452301;
    m_hash[1] = 0xefcdab89;
    m_hash[2] = 0x98badcfe;
    m_hash[3] = 0x10325476;
    m_hash[4] = 0xc3d2e1f0;
}

namespace
{
    inline uint32_t f1(uint32_t b, uint32_t c, uint32_t d)
    {
        return d ^ (b & (c ^ d));
    }

    inline uint32_t f2(uint32_t b, uint32_t c, uint32_t d)
    {
        return b ^ c ^ d;
    }

    inline uint32_t f3(uint32_t b, uint32_t c, uint32_t d)
    {
        return (b & c) | (b & d) | (c & d);
    }

    inline uint32_t rotate(uint32_t a, uint32_t c)
    {
        return (a << c) | (a >> (32 - c));
    }

    inline uint32_t swap(uint32_t x)
    {
#ifdef __GNUC__ || defined(__clang__)
        return __builtin_bswap32(x);
#endif
#ifdef MSC_VER
        return _byteswap_ulong(x);
#endif

        return (x >> 24) |
```

```
        ((x >> 8) & 0x0000FF00) |  
        ((x << 8) & 0x00FF0000) |  
        (x << 24);  
    }  
}  
  
void SHA1::processBlock(const void* data)  
{  
    uint32_t a = m_hash[0];  
    uint32_t b = m_hash[1];  
    uint32_t c = m_hash[2];  
    uint32_t d = m_hash[3];  
    uint32_t e = m_hash[4];  
  
    const uint32_t* input = (uint32_t*) data;  
  
    uint32_t words[80];  
    for (int i = 0; i < 16; i++)  
#if defined(__BYTE_ORDER) && (__BYTE_ORDER != 0) && (__BYTE_ORDER == __BIG_ENDIAN)  
        words[i] = input[i];  
#else  
        words[i] = swap(input[i]);  
#endif  
  
    for (int i = 16; i < 80; i++)  
        words[i] = rotate(words[i-3] ^ words[i-8] ^ words[i-14] ^ words[i-16], 1);  
  
    for (int i = 0; i < 4; i++)  
    {  
        int offset = 5*i;  
        e += rotate(a,5) + f1(b,c,d) + words[offset] + 0x5a827999; b = rotate(b,30);  
        d += rotate(e,5) + f1(a,b,c) + words[offset+1] + 0x5a827999; a = rotate(a,30);  
        c += rotate(d,5) + f1(e,a,b) + words[offset+2] + 0x5a827999; e = rotate(e,30);  
        b += rotate(c,5) + f1(d,e,a) + words[offset+3] + 0x5a827999; d = rotate(d,30);  
        a += rotate(b,5) + f1(c,d,e) + words[offset+4] + 0x5a827999; c = rotate(c,30);  
    }  
  
    for (int i = 4; i < 8; i++)  
    {  
        int offset = 5*i;  
        e += rotate(a,5) + f2(b,c,d) + words[offset] + 0x6ed9eba1; b = rotate(b,30);  
        d += rotate(e,5) + f2(a,b,c) + words[offset+1] + 0x6ed9eba1; a = rotate(a,30);  
        c += rotate(d,5) + f2(e,a,b) + words[offset+2] + 0x6ed9eba1; e = rotate(e,30);  
        b += rotate(c,5) + f2(d,e,a) + words[offset+3] + 0x6ed9eba1; d = rotate(d,30);  
        a += rotate(b,5) + f2(c,d,e) + words[offset+4] + 0x6ed9eba1; c = rotate(c,30);  
    }  
}
```

```
for (int i = 8; i < 12; i++)
{
    int offset = 5*i;
    e += rotate(a,5) + f3(b,c,d) + words[offset] + 0x8f1bbcdc; b = rotate(b,30);
    d += rotate(e,5) + f3(a,b,c) + words[offset+1] + 0x8f1bbcdc; a = rotate(a,30);
    c += rotate(d,5) + f3(e,a,b) + words[offset+2] + 0x8f1bbcdc; e = rotate(e,30);
    b += rotate(c,5) + f3(d,e,a) + words[offset+3] + 0x8f1bbcdc; d = rotate(d,30);
    a += rotate(b,5) + f3(c,d,e) + words[offset+4] + 0x8f1bbcdc; c = rotate(c,30);
}

for (int i = 12; i < 16; i++)
{
    int offset = 5*i;
    e += rotate(a,5) + f2(b,c,d) + words[offset] + 0xca62c1d6; b = rotate(b,30);
    d += rotate(e,5) + f2(a,b,c) + words[offset+1] + 0xca62c1d6; a = rotate(a,30);
    c += rotate(d,5) + f2(e,a,b) + words[offset+2] + 0xca62c1d6; e = rotate(e,30);
    b += rotate(c,5) + f2(d,e,a) + words[offset+3] + 0xca62c1d6; d = rotate(d,30);
    a += rotate(b,5) + f2(c,d,e) + words[offset+4] + 0xca62c1d6; c = rotate(c,30);
}

m_hash[0] += a;
m_hash[1] += b;
m_hash[2] += c;
m_hash[3] += d;
m_hash[4] += e;
}

void SHA1::add(const void* data, size_t numBytes)
{
    const uint8_t* current = (const uint8_t*) data;

    if (m_bufferSize > 0)
    {
        while (numBytes > 0 && m_bufferSize < BlockSize)
        {
            m_buffer[m_bufferSize++] = *current++;
            numBytes--;
        }
    }

    if (m_bufferSize == BlockSize)
    {
        processBlock((void*)m_buffer);
        m_numBytes += BlockSize;
        m_bufferSize = 0;
    }
}
```

```
if (numBytes == 0)
    return;

while (numBytes >= BlockSize)
{
    processBlock(current);
    current      += BlockSize;
    m_numBytes += BlockSize;
    numBytes     -= BlockSize;
}

while (numBytes > 0)
{
    m_buffer[m_bufferSize++] = *current++;
    numBytes--;
}
}

void SHA1::processBuffer()
{
    size_t paddedLength = m_bufferSize * 8;

    paddedLength++;

    size_t lower11Bits = paddedLength & 511;
    if (lower11Bits <= 448)
        paddedLength +=      448 - lower11Bits;
    else
        paddedLength += 512 + 448 - lower11Bits;

    paddedLength /= 8;

    unsigned char extra[BlockSize];

    if (m_bufferSize < BlockSize)
        m_buffer[m_bufferSize] = 128;
    else
        extra[0] = 128;

    size_t i;
    for (i = m_bufferSize + 1; i < BlockSize; i++)
        m_buffer[i] = 0;
    for (; i < paddedLength; i++)
        extra[i - BlockSize] = 0;

    uint64_t msgBits = 8 * (m_numBytes + m_bufferSize);
```

```
unsigned char* addLength;
if (paddedLength < BlockSize)
    addLength = m_buffer + paddedLength;
else
    addLength = extra + paddedLength - BlockSize;

*addLength++ = (unsigned char)((msgBits >> 56) & 0xFF);
*addLength++ = (unsigned char)((msgBits >> 48) & 0xFF);
*addLength++ = (unsigned char)((msgBits >> 40) & 0xFF);
*addLength++ = (unsigned char)((msgBits >> 32) & 0xFF);
*addLength++ = (unsigned char)((msgBits >> 24) & 0xFF);
*addLength++ = (unsigned char)((msgBits >> 16) & 0xFF);
*addLength++ = (unsigned char)((msgBits >> 8) & 0xFF);
*addLength = (unsigned char)(msgBits & 0xFF);

processBlock(m_buffer);

if (paddedLength > BlockSize)
    processBlock(extra);
}

std::string SHA1::getHash()
{
    unsigned char rawHash[HashBytes];
    getHash(rawHash);

    std::string result;
    result.reserve(2 * HashBytes);
    for (int i = 0; i < HashBytes; i++)
    {
        static const char dec2hex[16+1] = "0123456789abcdef";
        result += dec2hex[(rawHash[i] >> 4) & 15];
        result += dec2hex[rawHash[i] & 15];
    }

    return result;
}

void SHA1::getHash(unsigned char buffer[SHA1::HashBytes])
{
    uint32_t oldHash[HashValues];
    for (int i = 0; i < HashValues; i++)
        oldHash[i] = m_hash[i];

    processBuffer();
}
```

```
unsigned char* current = buffer;
for (int i = 0; i < HashValues; i++)
{
    *current++ = (m_hash[i] >> 24) & 0xFF;
    *current++ = (m_hash[i] >> 16) & 0xFF;
    *current++ = (m_hash[i] >> 8) & 0xFF;
    *current++ = m_hash[i] & 0xFF;

    m_hash[i] = oldHash[i];
}
}

std::string SHA1::operator()(const void* data, size_t numBytes)
{
    reset();
    add(data, numBytes);
    return getHash();
}

std::string SHA1::operator()(const std::string& text)
{
    reset();
    add(text.c_str(), text.size());
    return getHash();
}
```

# Erlang（密钥对认证）

最近更新时间：2023-12-22 10:09:32

## 操作场景

该任务指导您使用 Erlang 语言，通过密钥对鉴权来对您的 API 进行认证管理。

## 操作步骤

1. 在 [API 网关控制台](#)，创建一个 API，选择鉴权类型为“密钥对鉴权”（参考 [创建 API 概述](#)）。
2. 将 API 所在服务发布至发布环境（参考 [服务发布与下线](#)）。
3. 在控制台密钥管理界面创建密钥对。
4. 在控制台使用计划界面创建使用计划，并将使用计划与已创建的密钥对绑定（参考 [使用计划示例](#)）。
5. 将使用计划与 API 或 API 所在服务进行绑定。
6. 参考 [示例代码](#)，使用 Erlang 语言生成签名内容。

## 注意事项

最终发送的 HTTP 请求内至少包含两个 Header：Date 和 X-Date 二选一以及 Authorization，可以包含更多 header。

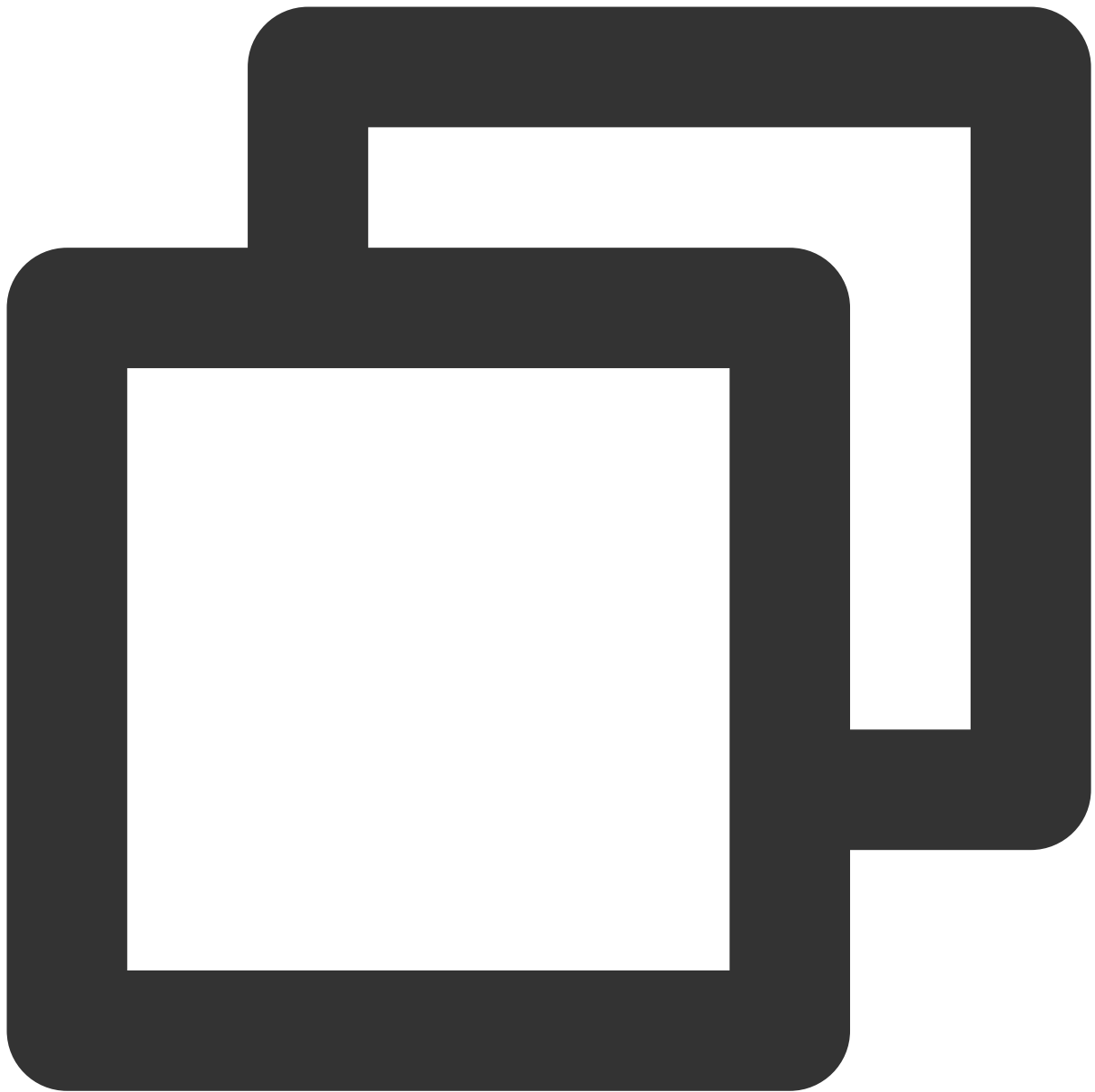
如果使用 Date Header，服务端将不会校验时间；如果使用 X-Date Header，服务端将校验时间。

Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Fri, 09 Oct 2015 00:00:00 GMT。

X-Date Header 的值为格林威治时间（GMT）格式的 HTTP 请求构造时间，例如 Mon, 19 Mar 2018 12:08:40 GMT。X-Date Header 里的时间和当前时间的差值不能超过15分钟。

如果是微服务 API，Header 中需要添加“X-NameSpace-Code”和“X-MicroService-Name”两个字段，通用 API 不需要添加，Demo 中未添加这两个字段。

## 示例代码



```
-module(apigateway_erlang_demo).  
  
%% API  
-export([request_api/0]).  
  
%% request_api()  
request_api() ->  
  
%%      启动inets服务，项目如果已启动此服务，去掉这里  
      inets:start(),
```

```
Url = "http://service-xxxxxxx-1234567890.ap-beijing.apigateway.myqcloud.co
Source = "xxxxxx",
GMTDate = now_to_utc_string(),
SecretId = "xxxDf4estwodtdzoke1234567890i3j9jv18wt9u",
SecretKey = "xxxSNF0CEp3OhN4t91234567890AWrct960X9192",
Sign = simple_sign(Source, GMTDate, SecretId, SecretKey),
Header = [
    {"Source", Source},
    {"x-Date", GMTDate},
    {"Authorization", Sign},
    {"Content-Type", "application/x-www-form-urlencoded"}
],

case httpc:request(get, {Url, Header}, [], []) of
    {ok, {_StatusLine, _Header, Result}} ->
        Result;

%% %%
        Result -> need decode

        Error ->
            Error
end.

simple_sign(Source, GMTDate, SecretId, SecretKey) ->
    Auth = "hmac id=\\\" \" ++ SecretId ++ \"\\\", algorithm=\\\"hmac-sha1\\\", header
    SecretKey = "xxxSNF0CEp3OhN4t91234567890AWrct960X9192",
    Source = "xxxxxx",
    SignStr = "x-date: \" ++ GMTDate ++ \"\\n\" ++ \"source: \" ++ Source,
    Mac = crypto:hmac(sha, SecretKey, SignStr),
    Sign = base64:encode(Mac),
    Sign2 = binary_to_list(Sign),
    Sign3 = Auth ++ Sign2 ++ \"\\\"\",
    Sign3.

%% 获取当前时间并转为 \"Mon, 02 Jan 2006 15:04:05 GMT\" 格式
now_to_utc_string() ->
    {{Year, Month, Day}, {Hour, Minute, Second}} = calendar:universal_time(),
    WeekNum = week_num(),
    Month1 = month_to_english(Month),
    WeekNum1 = week_to_english(WeekNum),
    Day1 = lists:flatten(io_lib:format(\"~2..0w\", [Day])),
    Date1 = WeekNum1 ++ \" \" ++ Day1 ++ \" \" ++ Month1,
    Date2 = lists:flatten(
        io_lib:format(\" ~4..0w ~2..0w:~2..0w:~2..0w GMT\",
            [Year, Hour, Minute, Second])),
    Date1 ++ Date2.
```

```
week_num() ->
    {Date, _} = calendar:local_time(),
    calendar:day_of_the_week(Date).

%% day_of_the_week
week_to_english(1) ->
    "Mon";
week_to_english(2) ->
    "Tue";
week_to_english(3) ->
    "Wed";
week_to_english(4) ->
    "Thu";
week_to_english(5) ->
    "Fri";
week_to_english(6) ->
    "Sat";
week_to_english(7) ->
    "Sun".

month_to_english(1) ->
    "Jan";
month_to_english(2) ->
    "Feb";
month_to_english(3) ->
    "Mar";
month_to_english(4) ->
    "Apr";
month_to_english(5) ->
    "May";
month_to_english(6) ->
    "Jun";
month_to_english(7) ->
    "Jul";
month_to_english(8) ->
    "Aug";
month_to_english(9) ->
    "Sept";
month_to_english(10) ->
    "Oct";
month_to_english(11) ->
    "Nov";
month_to_english(12) ->
    "Dec".
```

# 签名生成说明

最近更新时间：2023-12-22 10:09:43

## 注意事项

若创建 API 时选择密钥对鉴权，则该 API 所在的服务需要发布、该发布环境需要绑定使用计划、而且该使用计划需要绑定密钥对，然后使用对应的密钥对生成签名内容来访问该 API。

含有中文和空格的 query，请求体 Body 需要对值进行 urlencode 编码，编码方式为 UTF-8。

通过参数计算签名时，必须使用编码前的值进行签名，不支持使用 urlencode 编码后的字符串进行签名。请您在签名之后再对 query、body 的值做 urlencode 编码。

## 客户端错误码

| 错误码                                                                            | 状态码 | 语义                                                  | 解决方案                                          |
|--------------------------------------------------------------------------------|-----|-----------------------------------------------------|-----------------------------------------------|
| HMAC signature cannot be verified, a validate authorization header is required | 401 | 请求并未携带 Authorization 字段                             | 请参照各语言签名 demo 构造该字段                           |
| authorization headers is invalidate                                            | 403 | 请求 Authorization 字段格式不正确                            | 请参照各语言签名 demo 构造该字段                           |
| id or signature missing                                                        | 403 | 请求 Authorization 字段无 ID 或者 signature 字段             | 请参照各语言签名 demo 构造该字段                           |
| HMAC signature cannot be verified, a valid \$header header is required         | 403 | 请求 Authorization 字段中的 headers 字段在请求 header 中找不到对应字段 | 请参照各语言签名 demo 构造该字段                           |
| HMAC signature cannot be verified, a valid date header is required             | 403 | 请求 Authorization 必须需要使用 date 字段作为鉴权字段之一             | 请参照各语言签名 demo 构造该字段                           |
| Found no validate usage plan                                                   | 403 | 请求鉴权失败，API 需要鉴权，但该 API 并没有绑定使用计划                    | 关闭该 API 的鉴权或者给该 API 所在的发布环境绑定使用计划，给该使用计划绑定密钥对 |
| HMAC signature cannot be                                                       | 403 | 请求鉴权失败，Authorization 字                              | 请检查该密钥对是否可用，                                  |

|                                                                          |     |                                                         |                                              |
|--------------------------------------------------------------------------|-----|---------------------------------------------------------|----------------------------------------------|
| verified                                                                 |     | 段中携带的 Key ID 并未绑定该 API 所在的发布环境，或者 Key ID 非法，或者该 Key 被禁用 | 另外，检查该密钥对是否绑定对应使用计划/发布环境                     |
| HMAC signature does not match                                            | 403 | 请求鉴权失败，计算出来的 HMAC 值并不一致，请重新检查并计算                        | 计算的 HMAC 值错误，请参照各语言签名 demo 构造该字段             |
| Not Found Host                                                           | 404 | 请求中没有携带 Host 字段                                         | 请求携带 Host 字段，该字段值需要填服务器的域名，且为 String 类型      |
| Get Host Fail                                                            | 404 | 请求中携带的 Host 字段值并不是 String 类型                            | 将 Host 字段值改为 String 类型                       |
| Could not support method                                                 | 404 | 并不支持该请求方法类型                                             | 检查请求方法是否合法                                   |
| There is no api match host[\$host]                                       | 404 | 找不到请求服务器域名/地址                                           | 检查请求地址是否正确                                   |
| There is no api match env_mapping[\$env_mapping]                         | 404 | 自定义域名后的 env_mapping 字段错误                                | 检查绑定的自定义域名配置的 env_mapping 是否与自己填写的一致         |
| There is no api match default env_mapping[\$env_mapping]                 | 404 | 默认域名后的 env_mapping 字段需要是 test/prepub/release            | 默认域名后的 env_mapping 字段需要是 test/prepub/release |
| There is no api match uri[\$uri]                                         | 404 | 在该请求地址对应的服务下找不到对应 URI 匹配的 API                           | 请检查 URI 填写是否正确                               |
| There is no api match method[\$method]                                   | 404 | 在该请求地址+URI 对应的 API 并不支持该请求方法                            | 请检查请求方法是否正确                                  |
| "Not allow use HTTPS protocol或者Not allow use HTTP protocol"              | 404 | 该请求地址对应的服务并不支持对应 HTTP 协议类型                              | 请检查请求协议类型是否正确                                |
| req is cross origin, api \$uri need open cors flag on qcloud apigateway. | 429 | 该请求是跨域请求，但对应的 API 并未打开跨域开关                              | 请打开该 API 的跨域开关                               |