云 API

开发指南

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【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

开发指南
- C++ API
- .NET API
- GO API
- Java API
- Node.js API
- PHP API
- Python API

版权所有：腾讯云计算（北京）有限责任公司

开发指南

C++ API

最近更新时间：2023-03-07 18:16:40

腾讯云 API 全新升级3.0，该版本进行了性能优化且全地域部署、支持就近和按地域接入、访问时延下降显著，接口描述更加详细、错误码描述更加全面、SDK 增加接口级注释，让您更加方便快捷的使用腾讯云产品。这里针对 C++ API 调用方式进行简单说明。
现已支持云服务器（CVM）、云硬盘（CBS）、私有网络（VPC）、云数据库（TencentDB）等 腾讯云产品，后续会支持其他的云产品接入，敬请期待。

了解请求结构

1. 服务地址（endpoint）

API 支持就近地域接入（例如：cvm 产品域名为 `cvm.tencentcloudapi.com` ），也支持指定地域域名访问（例如：广州地域的域名为 `cvm.ap-guangzhou.tencentcloudapi.com` ），各地域参数见下文公共参数中的地域列表，详情请参见各产品的“请求结构”文档说明判断是否支持该地域。

注意：

对时延敏感的业务，建议指定带地域的域名。

2. 通信协议

腾讯云 API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法：
POST（推荐）
GET
POST 请求支持的 Content-Type 类型：
application/json（推荐），必须使用签名方法 v3（TC3-HMAC-SHA256）。
application/x-www-form-urlencoded，必须使用签名方法 v1（HmacSHA1 或 HmacSHA256）。
multipart/form-data（仅部分接口支持），必须使用签名方法 v3（TC3-HMAC-SHA256）。
GET 请求的请求包大小不得超过32KB。POST 请求使用签名方法 v1（HmacSHA1、HmacSHA256）时不得超过1MB。POST 请求使用签名方法 v3（TC3-HMAC-SHA256）时支持10MB。

4. 字符编码

均使用 UTF-8 编码。

版权所有：腾讯云计算（北京）有限责任公司

公共参数

说明：
公共参数是用于标识用户和接口签名的参数，每次请求均需要携带这些参数，才能正常发起请求。

签名方法 V3公共参数

签名方法 v3（有时也称作 TC3-HMAC-SHA256）相比签名方法 v1（部分文档简称为“签名方法”），更安全，支持更大的请求包，支持 POST JSON 格式，性能有一定提升，推荐使用该签名方法计算签名。使用方法见下文“签名方法介绍”。

参数名称	类型	必选	描述
X-TC-Action	String	**是**	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
X-TC-Timestamp	Integer	**是**	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702。**注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。**
X-TC-Version	String	**是**	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本2017-03-12。
Authorization	String	**是**	HTTP 标准身份认证头部字段，例如： TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=72e494ea8**********a96525168 其中： TC3-HMAC-SHA256：签名方法，目前固定取该值。 Credential：签名凭证，AKIDEXAMPLE 是 SecretId。 Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致。 service 为产品名，通常为域名前缀，例如域名 cvm.tencentcloudapi.com 意味着产品名是 cvm。本产品取值为 cvm。 SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部。 Signature：签名摘要，计算过程详见下文。
X-TC-Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

地域列表

由于各个产品支持地域不同，具体详情请参考各产品文档中的地域列表。
例如，您可以参考云服务器的 地域列表。

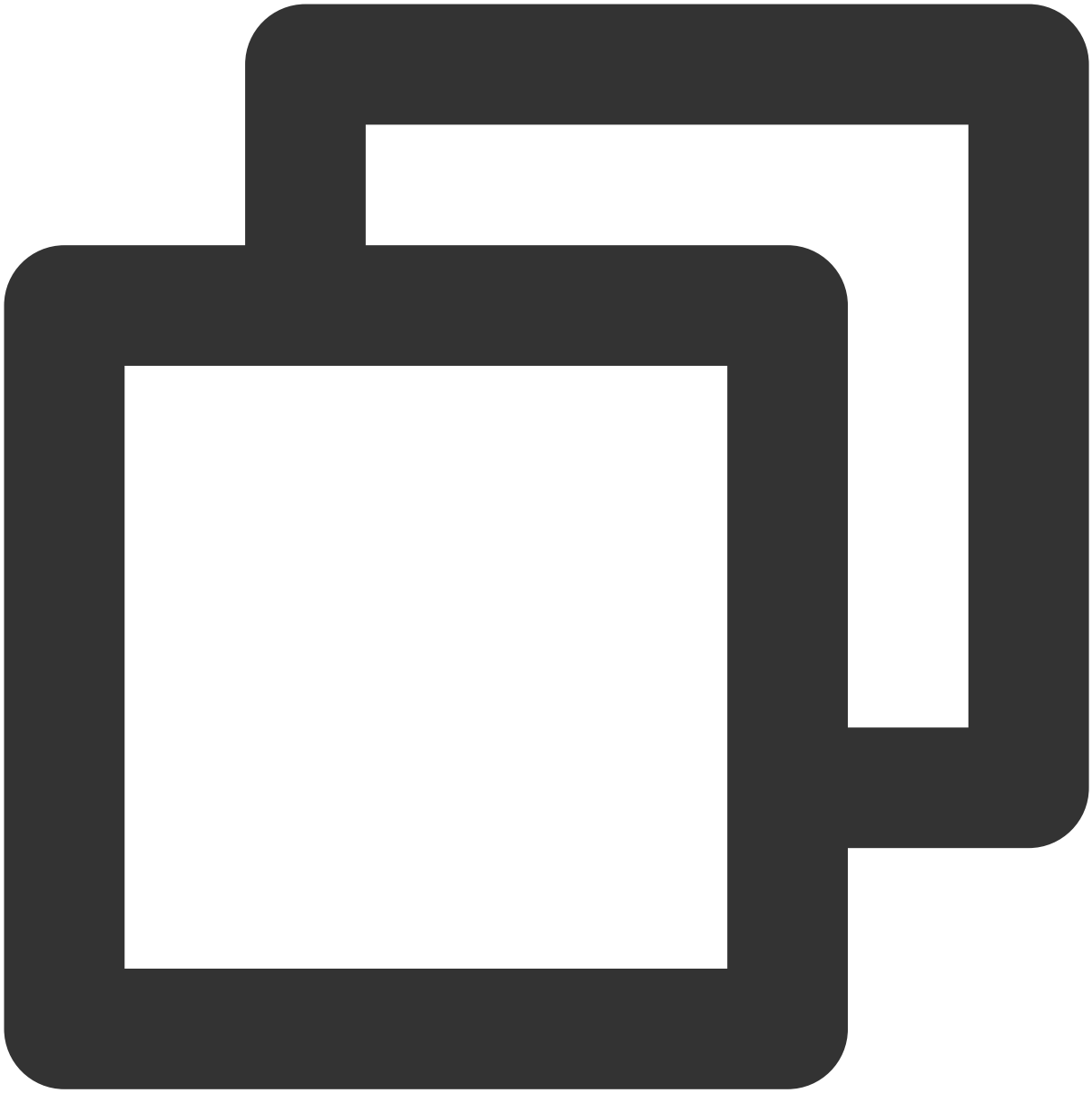
C++ API 调用方式

腾讯云 API 会对每个请求进行身份验证，用户需要使用安全凭证，经过特定的步骤对请求进行签名（Signature），每个请求都需要在公共请求参数中指定该签名结果并以指定的方式和格式发送请求。

注意：

目前 C++ 暂不支持 API 3.0签名 V1版本。

假设用户的 SecretId 和 SecretKey 分别是： `AKIDz8krbsJ5**********mLPx3EXAMPLE` 和 `Gu5t9xGAR***********EXAMPLE` 。用户想查看广州区云服务器名为 “未命名” 的主机状态，只返回一条数据。则请求可能为：



```
curl -X POST https://cvm.tencentcloudapi.com \\
-H "Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5*********mLPx3EXAMPLE/20
-H "Content-Type: application/json; charset=utf-8" \\
-H "Host: cvm.tencentcloudapi.com" \\
-H "X-TC-Action: DescribeInstances" \\
-H "X-TC-Timestamp: 1551113065" \\
-H "X-TC-Version: 2017-03-12" \\
-H "X-TC-Region: ap-guangzhou" \\
-d '{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instanc
```

步骤1：申请安全凭证

本文使用的安全凭证为密钥，密钥包括 SecretId 和 SecretKey。每个用户最多可以拥有两对密钥。

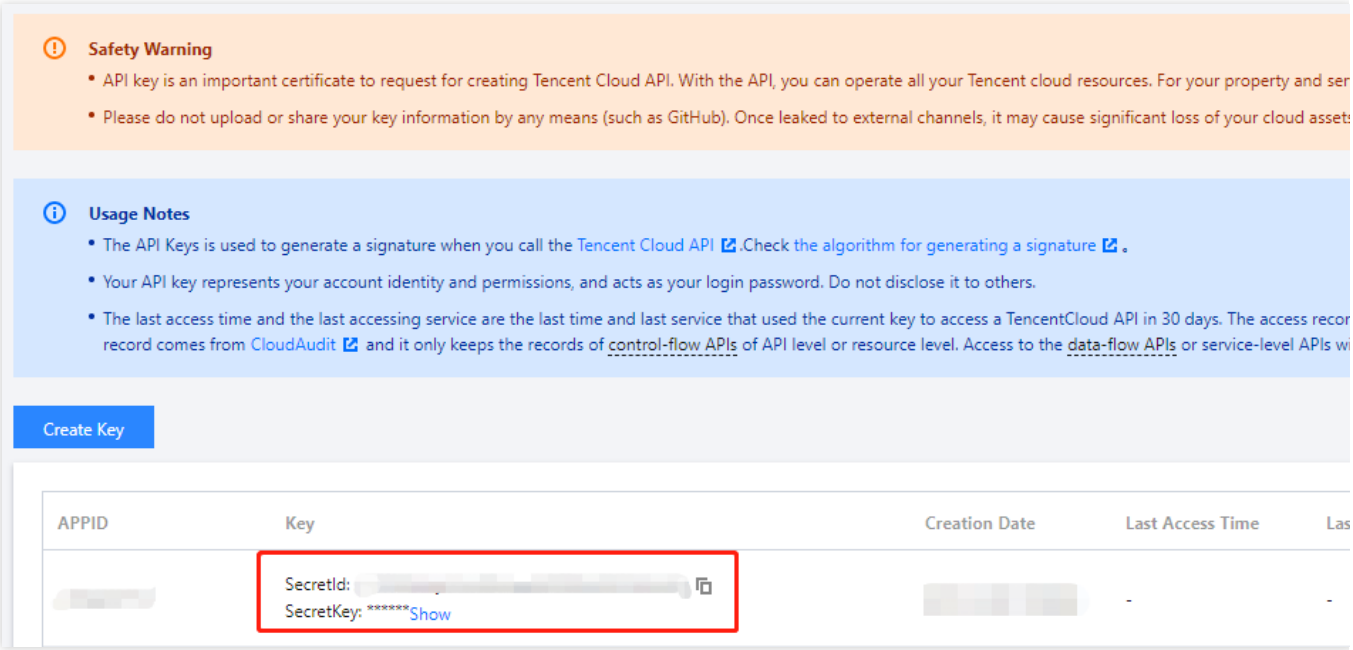
SecretId：用于标识 API 调用者身份，可以简单类比为用户名。

SecretKey：用于验证 API 调用者的身份，可以简单类比为密码。

注意：

用户必须严格保管安全凭证，避免泄露，否则将危及财产安全。如已泄漏，请立刻禁用该安全凭证。

前往 API 密钥管理 页面，即可进行获取。如下图所示：

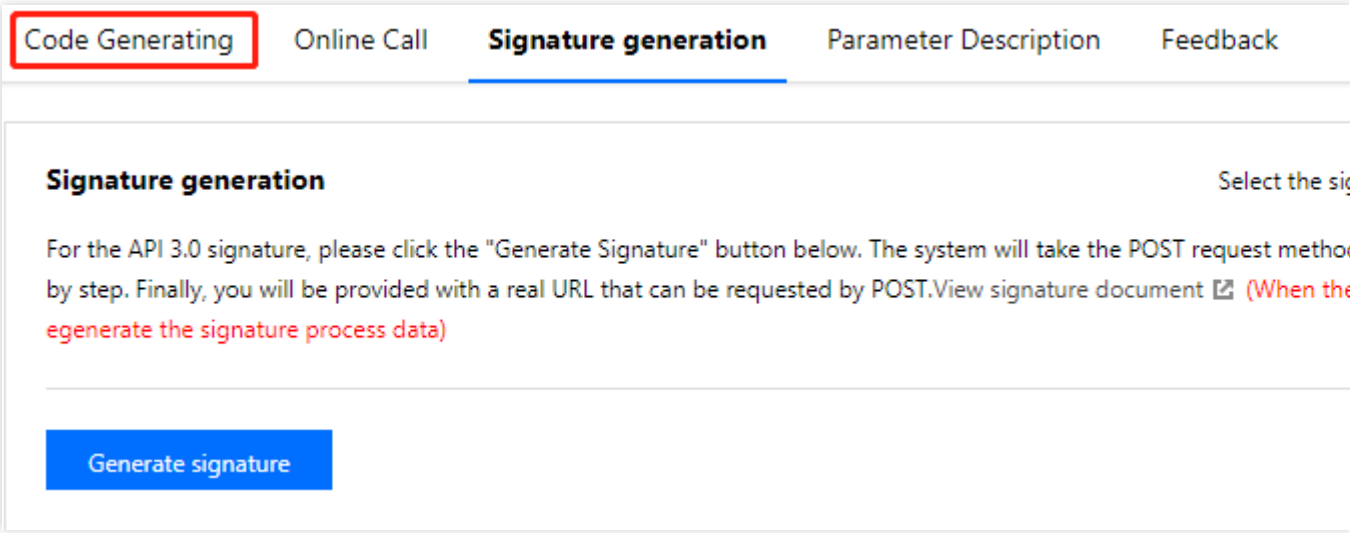


步骤2：获取 API 3.0 V3 版本签名

签名方法 v3（TC3-HMAC-SHA256）功能上覆盖了以前的签名方法 v1，而且更安全，支持更大的请求，支持 json 格式，性能有一定提升，推荐使用该签名方法计算签名。如下图所示：

说明：

首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为 “API 3.0 签名 v3”，可以生成签名过程进行验证，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的7种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

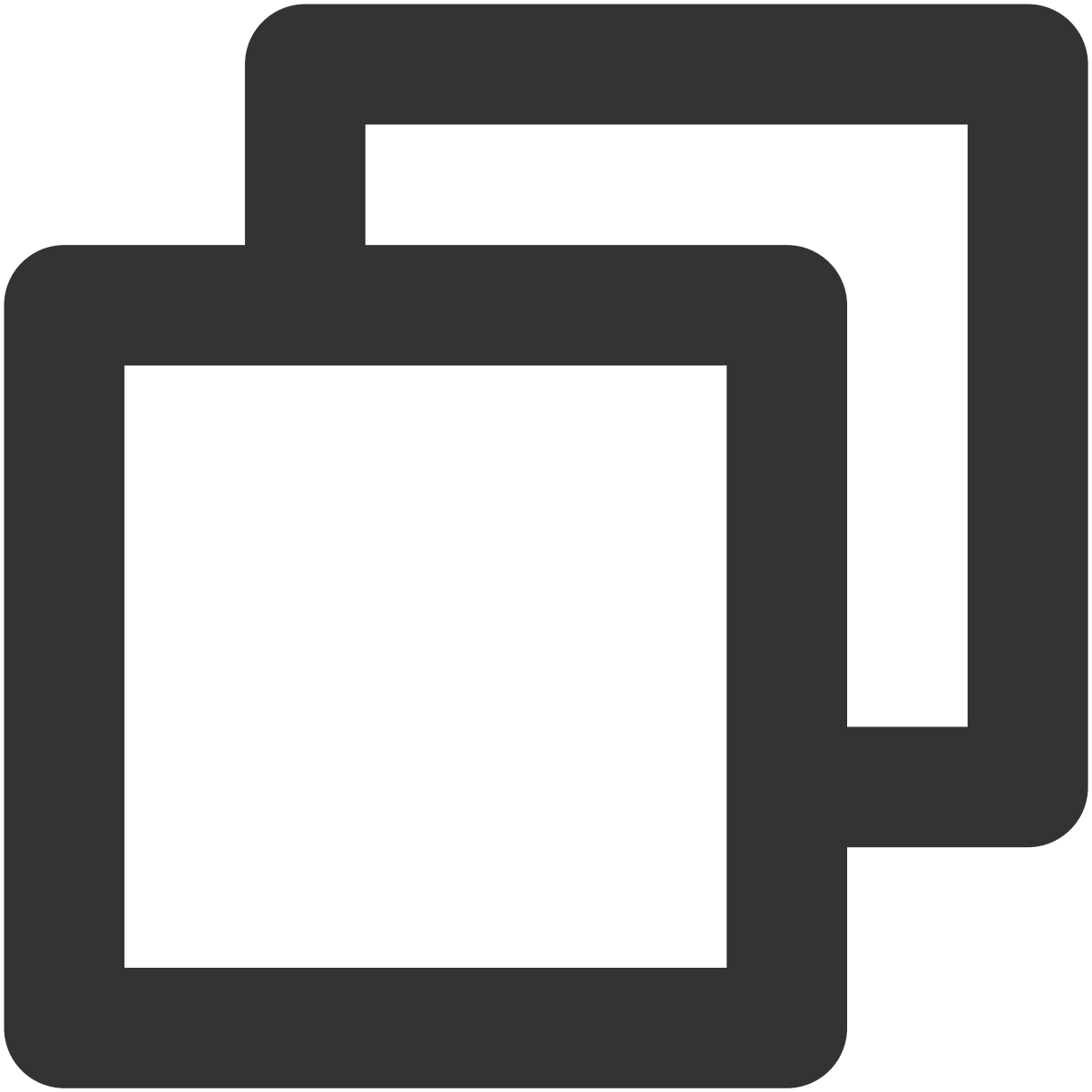


云 API 支持 GET 和 POST 请求。对于 GET 方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。对于 POST 方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式绝大多数接口均支持，multipart 格式只有特定接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。推荐使用 POST 请求，因为两者的结果并无差异，但 GET 请求只支持32KB以内的请求包。

下面以 查看实例列表 接口为例，分步骤介绍签名的计算过程。我们选择该接口原因是：

1. 云服务器默认已开通，该接口很常用。
2. 该接口是只读的，不会改变现有资源的状态。
3. 接口覆盖的参数种类较全，可以演示包含数据结构的数组如何使用。

1. 拼接规范请求串

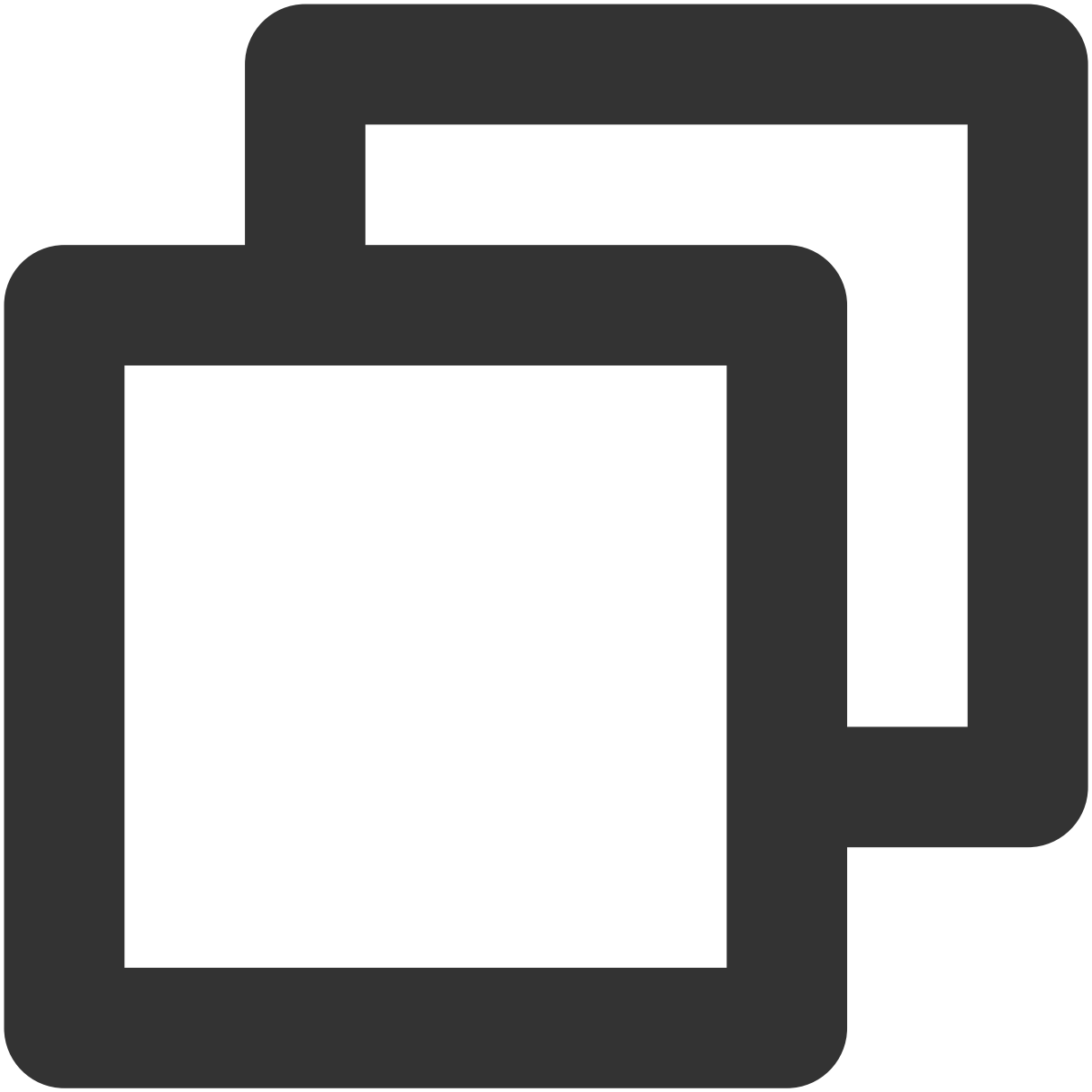


```
CanonicalRequest =
    HTTPRequestMethod + '\\n' +
    CanonicalURI + '\\n' +
    CanonicalQueryString + '\\n' +
    CanonicalHeaders + '\\n' +
    SignedHeaders + '\\n' +
    HashedRequestPayload
```

字段名称	解释

HTTPRequestMethod	HTTP 请求方法（GET、POST）。此示例取值为 `POST` 。
CanonicalURI	URI 参数，API 3.0 固定为正斜杠（/）。
CanonicalQueryString	发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串""，对于 G 求，则为 URL 中问号（?）后面的字符串内容，例如：Limit=10&Offset=0。 注意：CanonicalQueryString 需要参考 RFC3986 进行 URLEncode，字符集 UTF8，推荐使用 语言标准库，所有特殊字符均需编码，大写形式。
CanonicalHeaders	参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部 签名以提高自身请求的唯一性和安全性。 拼接规则：头部 key 和 value 统一转成小写，掉首尾空格，按照 key:value\\n 格式拼接；多个头部，按照头部 key（小写）的 ASCII 行拼接。此示例计算结果是 `content-type:application/json; charset=ut 8\\nhost:cvm.tencentcloudapi.com\\n` 。 注意：content-type 必须和实际发 符合，有些编程语言网络库即使未指定也会自动添加 charset 值，如果签名时和发送时 致，服务器会返回签名校验失败。
SignedHeaders	参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包 部内容是一一对应的。content-type 和 host 为必选头部。 拼接规则：头部 key 统一转 写；多个头部 key（小写）按照 ASCII 升序进行拼接，并且以分号（;）分隔。此示例 content-type;host
HashedRequestPayload	请求正文（Requestpayload，即 body，此示例为 `{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-name"` 的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(RequestPayload))), HTTP 请求正文做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。 GET 请求，RequestPayload 固定为空字符串。此示例计算结果是 `35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f`

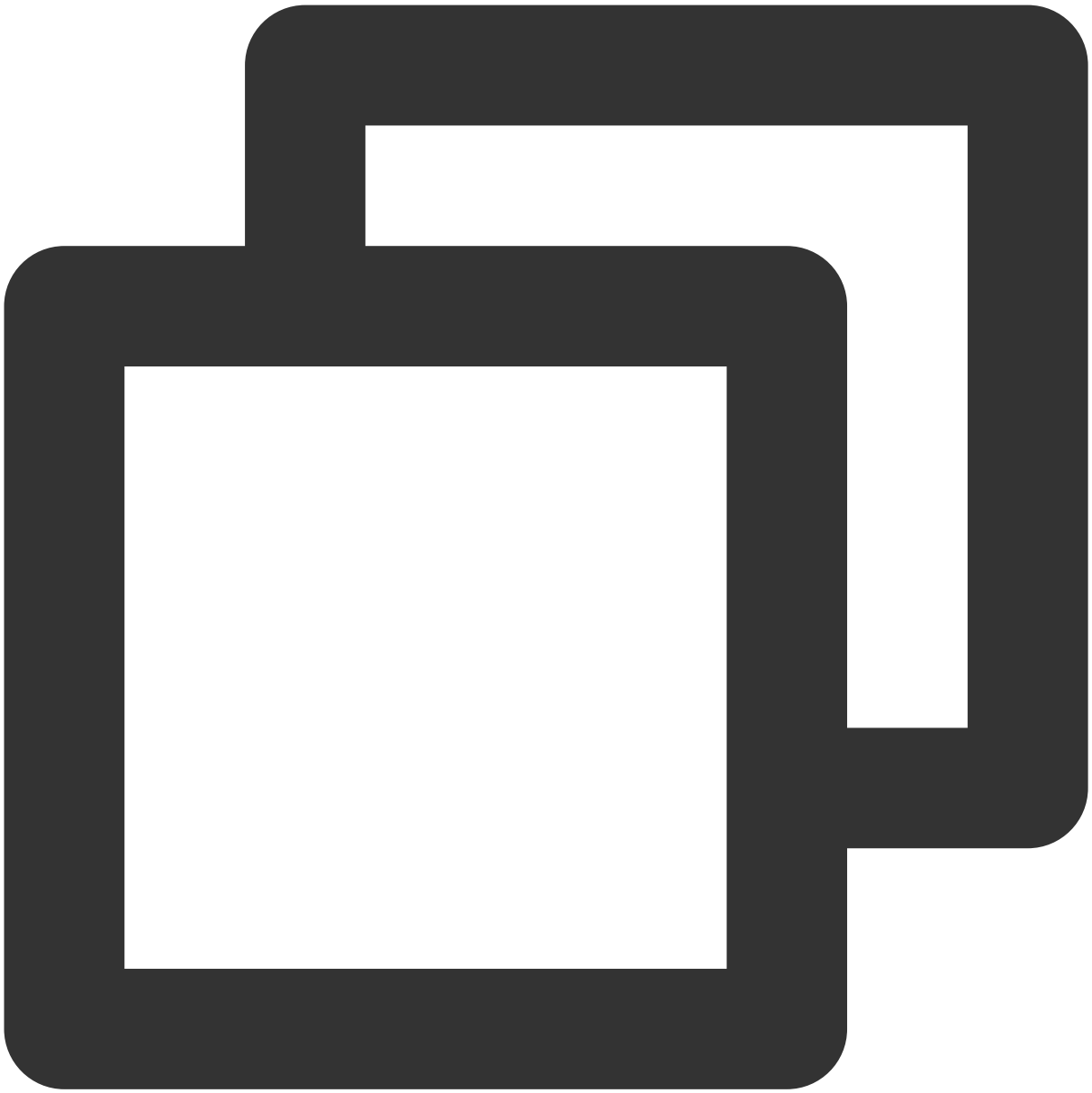
根据以上规则，示例中得到的规范请求串如下：



```
POST
/

content-type:application/json; charset=utf-8
host:cvm.tencentcloudapi.com

content-type;host
35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f064
```

2. 拼接待签名字符串

按如下格式拼接待签名字符串：



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

字段名称	解释
Algorithm	签名算法，目前固定为 `TC3-HMAC-SHA256` 。

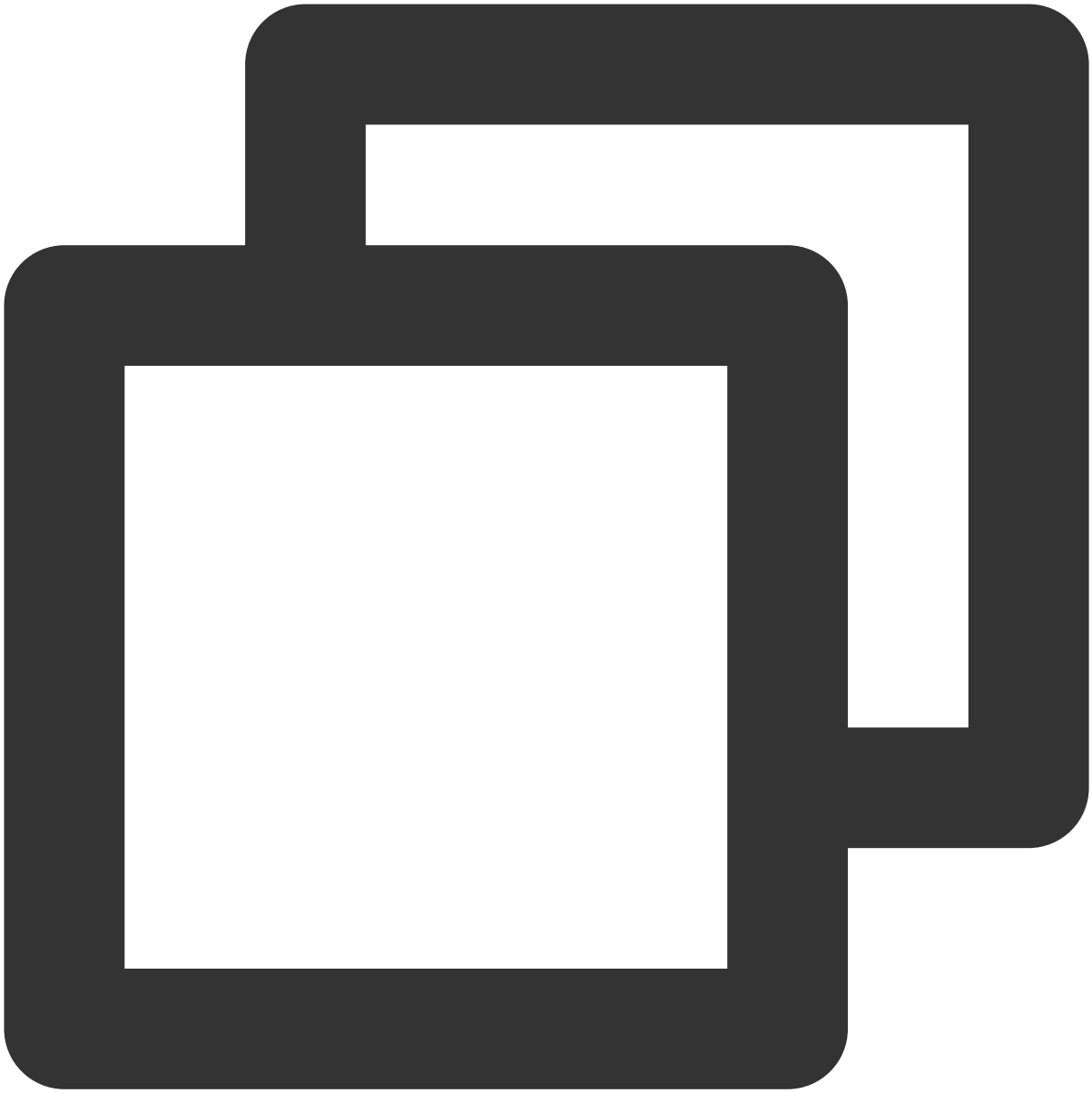
RequestTimestamp	请求时间戳，即请求头部的公共参数 X-TC-Timestamp 取值，取当前时间 UNIX 时间到秒。此示例取值为 `1551113065` 。
CredentialScope	凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符串（tc3_request）。**Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Time 算的 UTC 标准时间日期一致**；service 为产品名，必须与调用的产品域名一致。此示果是2019-02-25/cvm/tc3_request。
HashedCanonicalRequest	前述步骤拼接所得规范请求串的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))。此示例计算结果是 `5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca1`

注意：

Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为1551113065，在东八区的时间是2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为2019-02-25，而不是2019-02-26。

Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败，返回签名过期错误。

根据以上规则，示例中得到的待签名字符串如下：

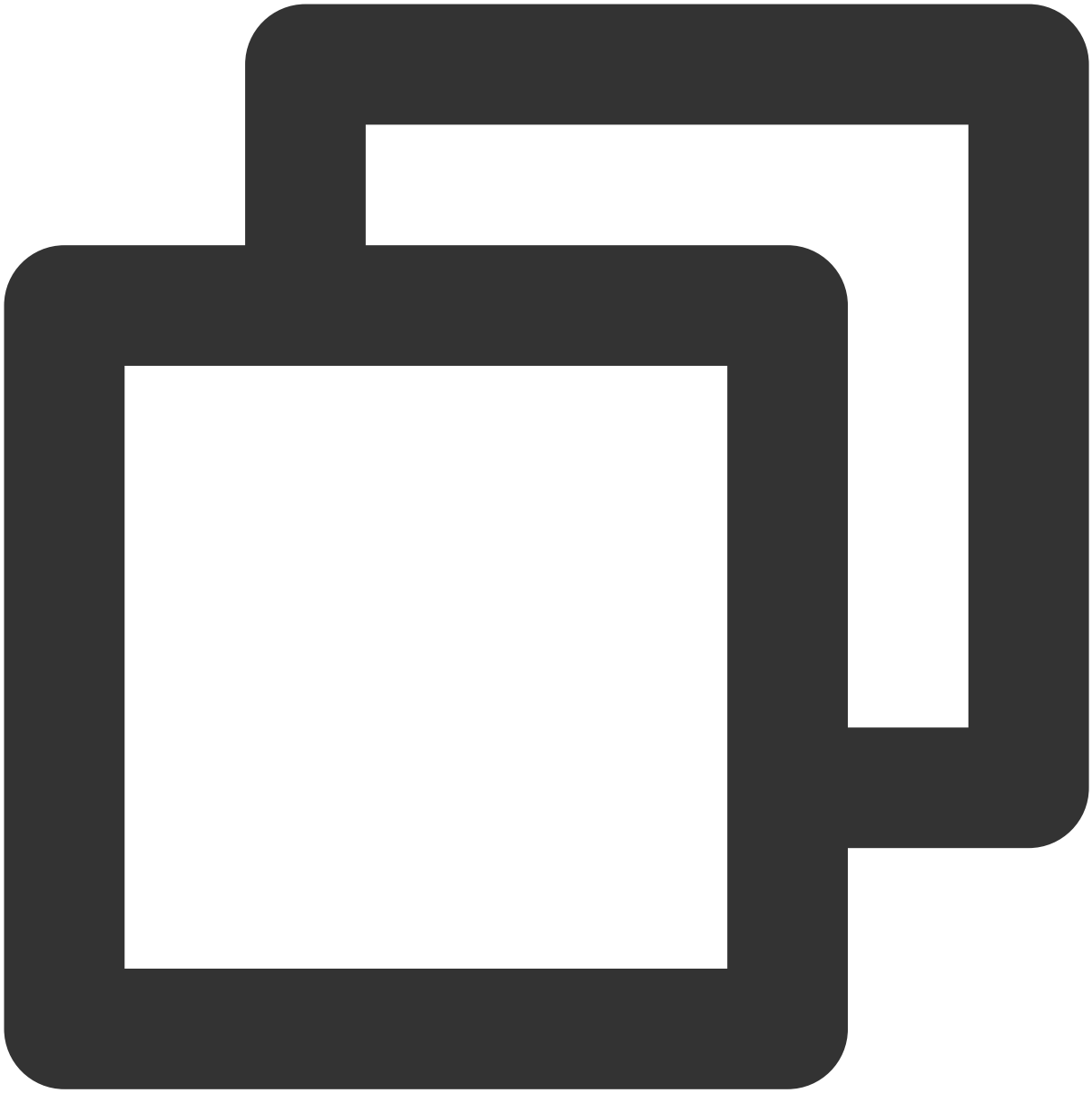


```
TC3-HMAC-SHA256
1551113065
2019-02-25/cvm/tc3_request
5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d7031
```

3. 计算签名（伪代码）

实际请参考下面示例：

1）计算派生签名密钥，伪代码如下：



```
#include <tencentcloud/core/Sign.h>
#include <tencentcloud/core/utils/Utils.h>


using namespace TencentCloud;
using namespace std;

string Sign::Tc3Sign(const string &credDate, const string &serviceName, const strin
{
    string kKey = "TC3"+this->m_secretKey;
    string kDate = Utils::HmacSha256(kKey, credDate);
```

```
    string kService = Utils::HmacSha256(kDate, serviceName);
    string kSigning = Utils::HmacSha256(kService, "tc3_request");
    return Utils::HexEncode(Utils::HmacSha256(kSigning, signStr));
}
```

派生出的密钥 `SecretDate` 、 `SecretService` 和 `SecretSigning` 是二进制的数据，可能包含不可打印字符，此处不展示中间结果。

注意：

不同的编程语言，HMAC 库函数中参数顺序可能不一样，此处的伪代码密钥参数在后，请以实际编程语言为准。通常标准库函数会提供二进制格式的计算值，也即此处使用的，也会提供打印友好的十六进制格式的计算值，将在下面计算签名结果时使用。

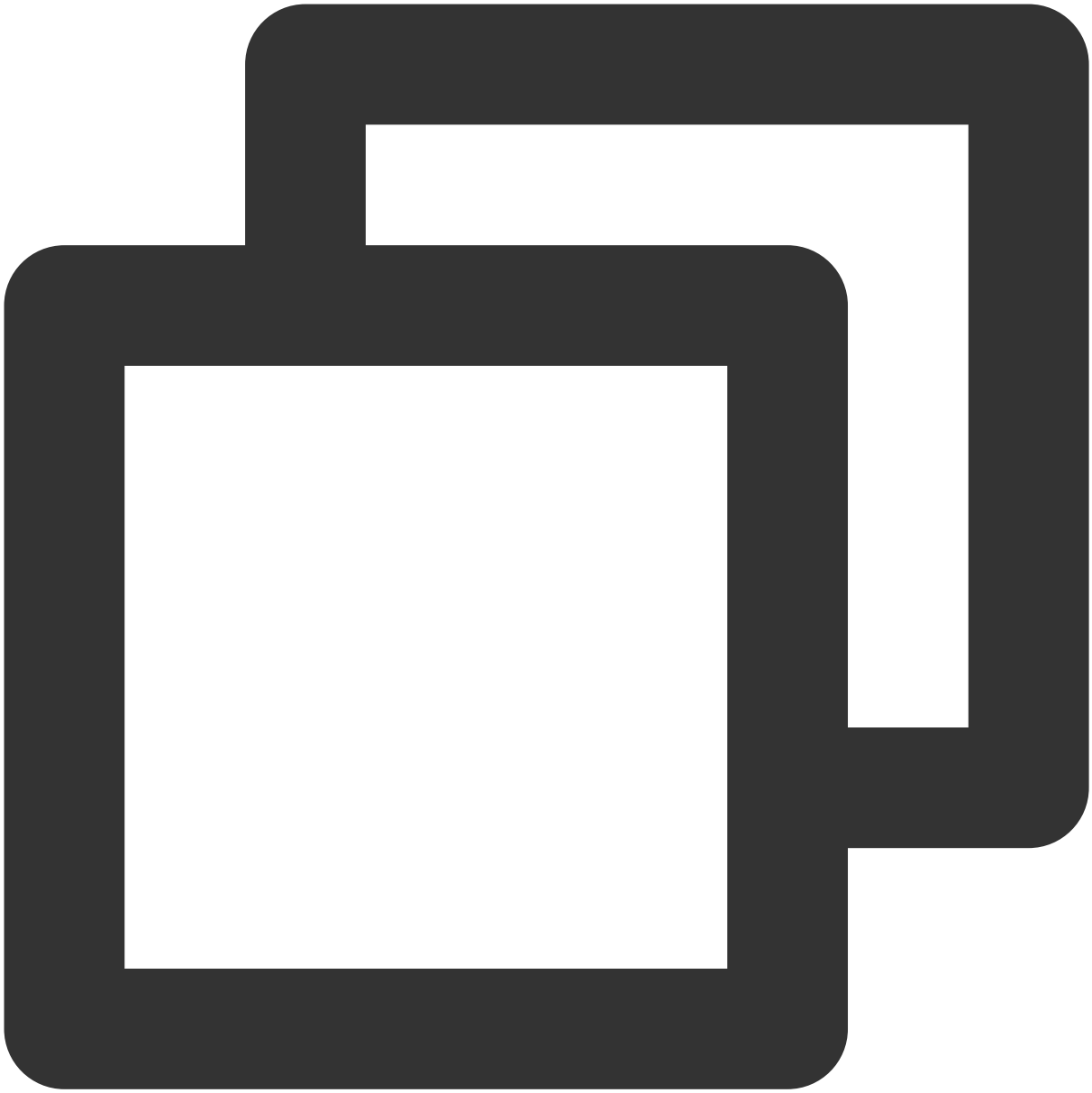
字段名称	解释
m_secretKey	原始的 SecretKey，即 `Gu5t9xGAR**********EXAMPLE` 。
credDate	即 Credential 中的 Date 字段信息。此示例取值为 `2019-02-25` 。
serviceName	即 Credential 中的 Service 字段信息。此示例取值为 `cvm` 。
signStr	代签名字符串。

2）计算签名：

此示例计算结果是 `72e494ea8*****************************************a96525168` 。

4. 拼接 Authorization

按如下格式拼接 Authorization：



```
Authorization =
    Algorithm + ' ' +
    'Credential=' + SecretId + '/' + CredentialScope + ', ' +
    'SignedHeaders=' + SignedHeaders + ', ' +
    'Signature=' + Signature
```

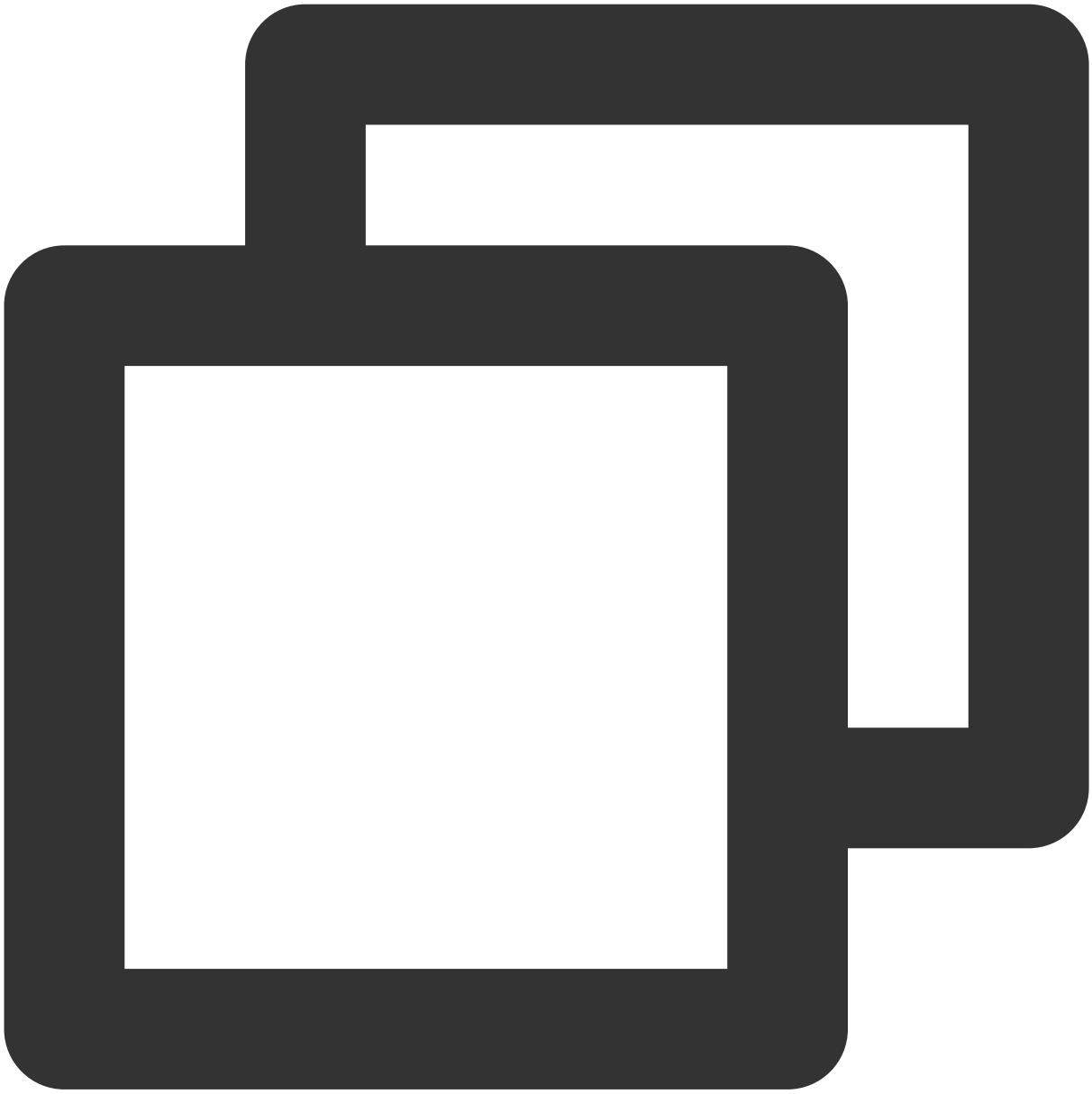
字段名称	解释
Algorithm	签名方法，固定为 `TC3-HMAC-SHA256` 。
SecretId	

	密钥对中的 SecretId，即 `AKIDz8krbsJ5**********mLPx3EXAMPLE` 。
CredentialScope	见上文，凭证范围。此示例计算结果是 `2019-02-25/cvm/tc3_request` 。
SignedHeaders	见上文，参与签名的头部信息。此示例取值为 `content-type;host` 。
Signature	签名值。此示例计算结果是 `72e494ea8****************************************a96525168` 。

根据以上规则，示例中得到的值为：

```
TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/2019-02-25/cvm/tc3_re
```

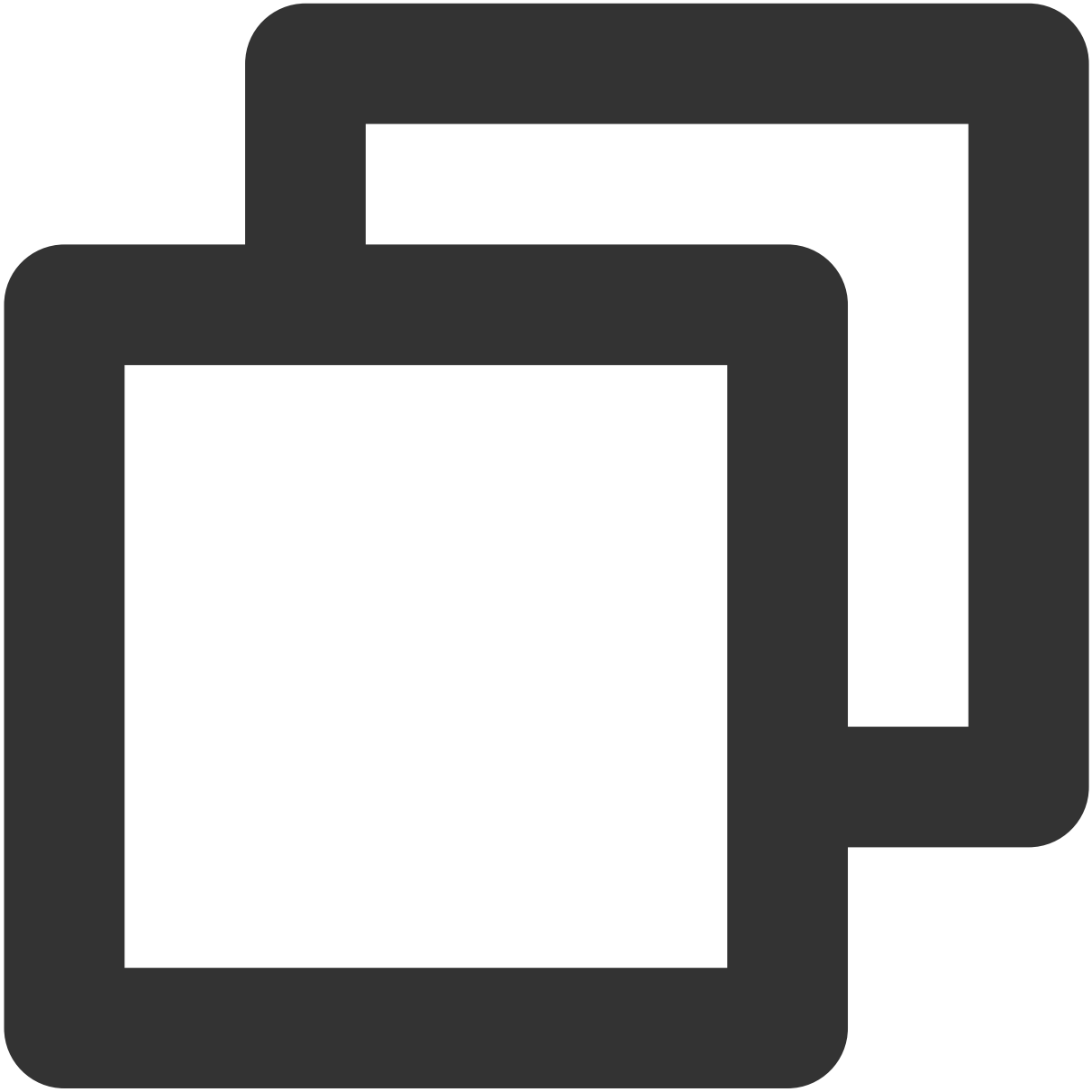
最终完整的调用信息如下：



```
POST https://cvm.tencentcloudapi.com/
Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/2019-0
Content-Type: application/json; charset=utf-8
Host: cvm.tencentcloudapi.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1551113065
```

```
X-TC-Region: ap-guangzhou

{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-na
```

5. API 3.0签名 V3示例



```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>
#include <stdio.h>
```

```
#include <time.h>
#include <openssl/sha.h>
#include <openssl/hmac.h>

using namespace std;

string get_data(int64_t &timestamp)
{
    string utcDate;
    char buff[20] = {0};
    // time_t timenow;
    struct tm sttime;
    sttime = *gmtime(&timestamp);
    strftime(buff, sizeof(buff), "%Y-%m-%d", &sttime);
    utcDate = string(buff);
    return utcDate;
}
string int2str(int64_t n)
{
    std::stringstream ss;
    ss << n;
    return ss.str();
}
string sha256Hex(const string &str)
{
    char buf[3];
    unsigned char hash[SHA256_DIGEST_LENGTH];
    SHA256_CTX sha256;
    SHA256_Init(&sha256);
    SHA256_Update(&sha256, str.c_str(), str.size());
    SHA256_Final(hash, &sha256);
    std::string NewString = "";
    for(int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        snprintf(buf, sizeof(buf), "%02x", hash[i]);
        NewString = NewString + buf;
    }
    return NewString;
}
string HmacSha256(const string &key, const string &input)
{
    unsigned char hash[32];

    HMAC_CTX *h;
#if OPENSSL_VERSION_NUMBER < 0x10100000L
    HMAC_CTX hmac;
    HMAC_CTX_init(&hmac);
```

```
    h = &hmac;
#else
    h = HMAC_CTX_new();
#endif

    HMAC_Init_ex(h, &key[0], key.length(), EVP_sha256(), NULL);
    HMAC_Update(h, ( unsigned char* )&input[0], input.length());
    unsigned int len = 32;
    HMAC_Final(h, hash, &len);

#if OPENSSL_VERSION_NUMBER < 0x10100000L
    HMAC_CTX_cleanup(h);
#else
    HMAC_CTX_free(h);
#endif

    std::stringstream ss;
    ss << std::setfill('0');
    for (int i = 0; i < len; i++)
    {
        ss  << hash[i];
    }

    return (ss.str());
}
string HexEncode(const string &input)
{
    static const char* const lut = "0123456789abcdef";
    size_t len = input.length();

    string output;
    output.reserve(2 * len);
    for (size_t i = 0; i < len; ++i)
    {
        const unsigned char c = input[i];
        output.push_back(lut[c >> 4]);
        output.push_back(lut[c & 15]);
    }
    return output;
}

int main()
{
    // 密钥参数
    string SECRET_ID = "AKIDz8krbsJ5**********mLPx3EXAMPLE";
    string SECRET_KEY = "Gu5t9xGAR***********EXAMPLE";
```

```
    string service = "cvm";
    string host = "cvm.tencentcloudapi.com";
    string region = "ap-guangzhou";
    string action = "DescribeInstances";
    string version = "2017-03-12";
    int64_t timestamp = 1551113065;
    string date = get_data(timestamp);

    // ************* 步骤 1：拼接规范请求串 *************
    string httpRequestMethod = "POST";
    string canonicalUri = "/";
    string canonicalQueryString = "";
    string canonicalHeaders = "content-type:application/json; charset=utf-8\\nhost:
    string signedHeaders = "content-type;host";
    string payload = "{\\"Limit\\": 1, \\"Filters\\": [{\\"Values\\": [\\"\\\\u672a
    string hashedRequestPayload = sha256Hex(payload);
    string canonicalRequest = httpRequestMethod + "\\n" + canonicalUri + "\\n" + ca
            + canonicalHeaders + "\\n" + signedHeaders + "\\n" + hashedRequestPaylo
    cout << canonicalRequest << endl;
    cout << "-----------------------" << endl;

    // ************* 步骤 2：拼接待签名字符串 *************
    string algorithm = "TC3-HMAC-SHA256";
    string RequestTimestamp = int2str(timestamp);
    string credentialScope = date + "/" + service + "/" + "tc3_request";
    string hashedCanonicalRequest = sha256Hex(canonicalRequest);
    string stringToSign = algorithm + "\\n" + RequestTimestamp + "\\n" + credential
    cout << stringToSign << endl;
    cout << "-----------------------" << endl;

    // ************* 步骤 3：计算签名 ***************
    string kKey = "TC3" + SECRET_KEY;
    string kDate = HmacSha256(kKey, date);
    string kService = HmacSha256(kDate, service);
    string kSigning = HmacSha256(kService, "tc3_request");
    string signature = HexEncode(HmacSha256(kSigning, stringToSign));
    cout << signature << endl;
    cout << "-----------------------" << endl;

    // ************* 步骤 4：拼接 Authorization *************
    string authorization = algorithm + " " + "Credential=" + SECRET_ID + "/" + cred
            + "SignedHeaders=" + signedHeaders + ", " + "Signature=" + signature;
    cout << authorization << endl;
    cout << "-----------------------" << endl;

    string headers = "curl -X POST https://" + host + "\\n"
                   + " -H \\"Authorization: " + authorization + "\\n"
```

```
                    + " -H \\"Content-Type: application/json; charset=utf-8\\"" + "\
                    + " -H \\"Host: " + host + "\\n"
                    + " -H \\"X-TC-Action: " + action + "\\n"
                    + " -H \\"X-TC-Timestamp: " + RequestTimestamp + "\\n"
                    + " -H \\"X-TC-Version: " + version + "\\n"
                    + " -H \\"X-TC-Region: " + region + "\\n"
                    + " -d '" + payload;
    cout << headers << endl;
    return 0;
};
```

API 2.0 签名

此签名现已不在维护，建议使用性能更优的 **API 3.0 签名**，如需使用，请直接在 API Explorer 的**签名串生成 > 选择 API 2.0 签名版本**进行操作。

签名失败

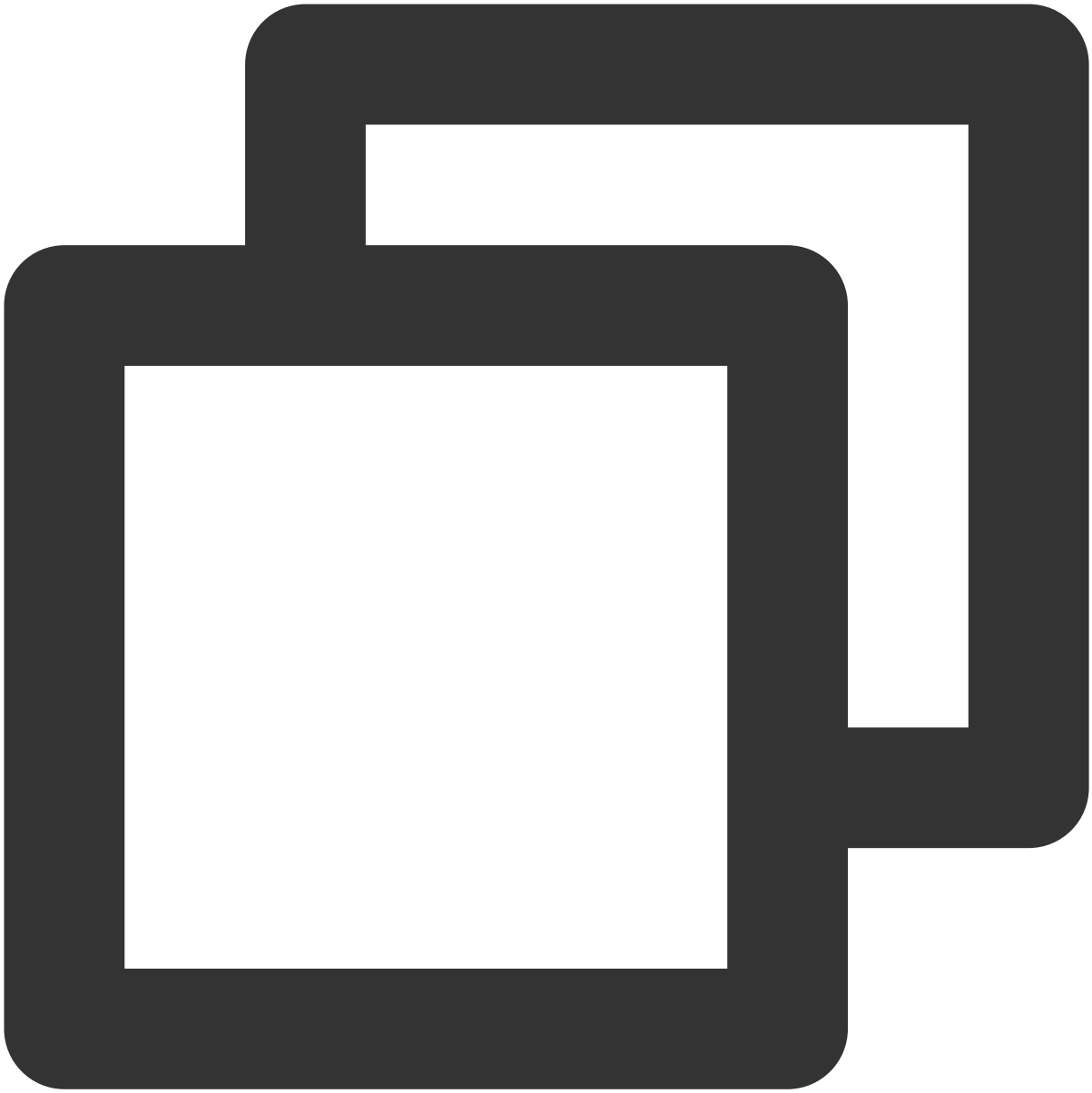
存在以下签名失败的错误码，请根据实际情况处理。

错误码	错误描述
AuthFailure.SignatureExpire	签名过期。Timestamp 与服务器接收到请求的时间相差不得超过五分钟。
AuthFailure.SecretIdNotFound	密钥不存在。请到控制台查看密钥是否被禁用，是否少复制了字符或者多了字符。
AuthFailure.SignatureFailure	签名错误。可能是签名计算错误，或者签名与实际发送的内容不相符合，也有可能是密钥 SecretKey 错误导致的。
AuthFailure.TokenFailure	临时证书 Token 错误。
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。

返回结果

正确返回结果

以云服务器的接口查看实例状态列表（DescribeInstancesStatus）2017-03-12 版本为例，若调用成功，其可能的返回如下为：



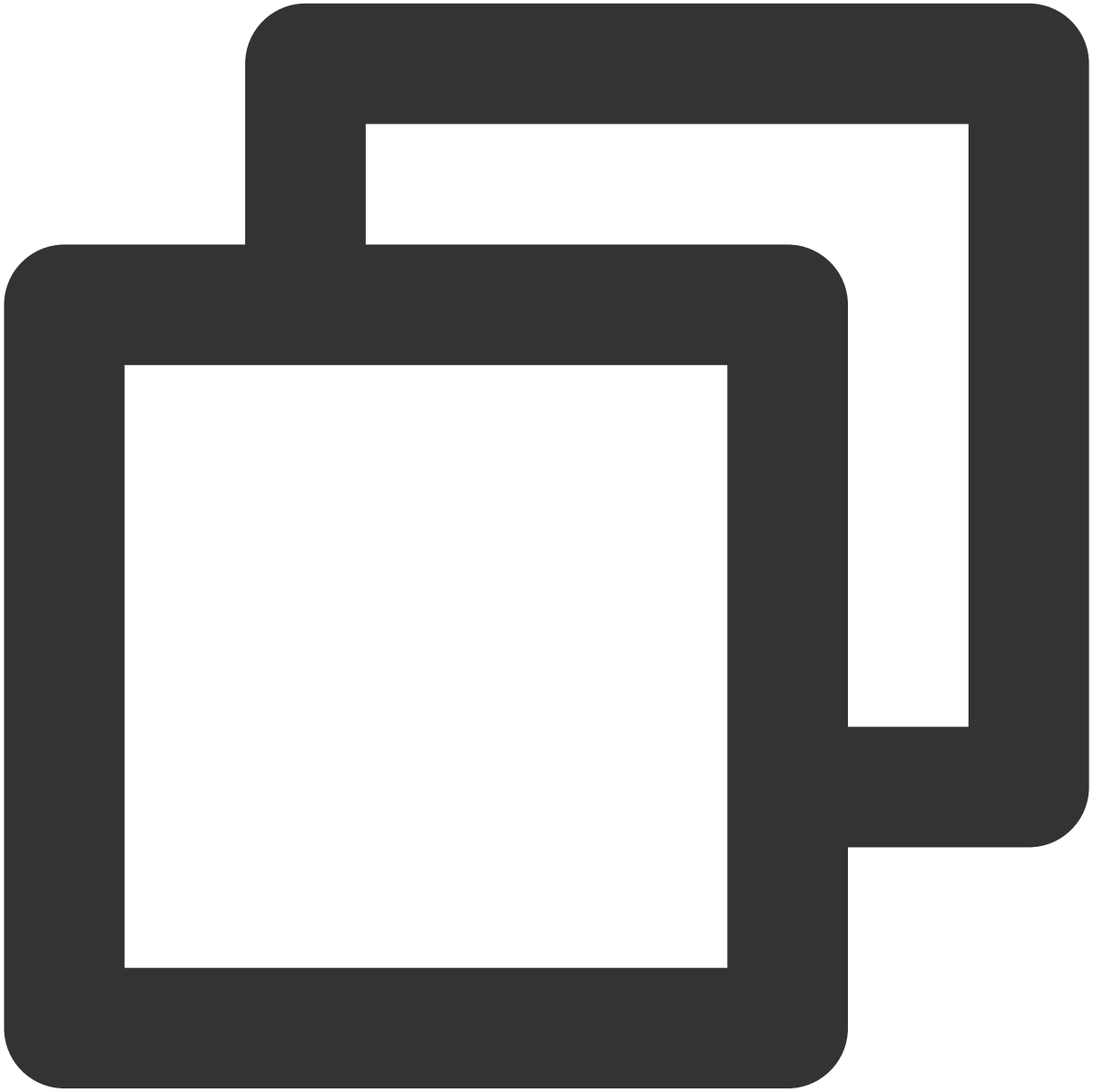
```
{
    "Response": {
        "TotalCount": 0,
        "InstanceStatusSet": [],
        "RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
    }
}
```

`Response` 及其内部的 `RequestId` 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 `TotalCount` 和 `InstanceStatusSet` 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 `TotalCount` 在此情况下的返回值为0， `InstanceStatusSet` 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：



```
{
    "Response": {
        "Error": {
```

```
            "Code": "AuthFailure.SignatureFailure",
            "Message": "The provided credentials could not be validated. Please che
        },
        "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
    }
}
```

`Error` 的出现代表着该请求调用失败。 `Error` 字段连同其内部的 `Code` 和 `Message` 字段在调用失败时是必定返回的。

`Code` 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。

`Message` 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。

`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。 Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。

LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	HTTPS 请求方法错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

版权所有：腾讯云计算（北京）有限责任公司

.NET API

最近更新时间：2023-03-07 18:16:40

腾讯云 API 全新升级3.0，该版本进行了性能优化且全地域部署、支持就近和按地域接入、访问时延下降显著，接口描述更加详细、错误码描述更加全面、SDK 增加接口级注释，让您更加方便快捷的使用腾讯云产品。这里针对 .NET API 调用方式进行简单说明。
现已支持云服务器（CVM）、云硬盘（CBS）、私有网络（VPC）、云数据库（TencentDB）等 腾讯云产品，后续会支持其他的云产品接入，敬请期待。

了解请求结构

1. 服务地址（endpoint）

API 支持就近地域接入（例如：cvm 产品域名为 `cvm.tencentcloudapi.com` ），也支持指定地域域名访问（例如：广州地域的域名为 `cvm.ap-guangzhou.tencentcloudapi.com` ），各地域参数见下文公共参数中的地域列表，详情请参见各产品的“请求结构”文档说明判断是否支持该地域。
注意：
对时延敏感的业务，建议指定带地域的域名。

2. 通信协议

腾讯云 API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法：
POST（推荐）
GET
POST 请求支持的 Content-Type 类型：
application/json（推荐），必须使用签名方法 v3（TC3-HMAC-SHA256）。
application/x-www-form-urlencoded，必须使用签名方法 v1（HmacSHA1 或 HmacSHA256）。
multipart/form-data（仅部分接口支持），必须使用签名方法 v3（TC3-HMAC-SHA256）。
GET 请求的请求包大小不得超过32KB。POST 请求使用签名方法 v1（HmacSHA1、HmacSHA256）时不得超过1MB。POST 请求使用签名方法 v3（TC3-HMAC-SHA256）时支持10MB。

4. 字符编码

均使用 UTF-8 编码。

版权所有：腾讯云计算（北京）有限责任公司

公共参数

说明：
公共参数是用于标识用户和接口签名的参数，每次请求均需要携带这些参数，才能正常发起请求。

签名方法 V3公共参数

签名方法 v3（有时也称作 TC3-HMAC-SHA256）相比签名方法 v1 （部分文档简称为“签名方法”），更安全，支持更大的请求包，支持 POST JSON 格式，性能有一定提升，推荐使用该签名方法计算签名。使用方法见下文 “签名方法介绍” 。

参数名称	类型	必选	描述
X-TC-Action	String	**是**	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
X-TC-Timestamp	Integer	**是**	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702。**注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。**
X-TC-Version	String	**是**	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本2017-03-12。
Authorization	String	**是**	HTTP 标准身份认证头部字段，例如： TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=72e494ea8**********a96525168 其中： TC3-HMAC-SHA256：签名方法，目前固定取该值。 Credential：签名凭证，AKIDEXAMPLE 是 SecretId。 Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致。 service 为产品名，通常为域名前缀，例如域名 cvm.tencentcloudapi.com 意味着产品名是 cvm。本产品取值为 cvm。 SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部。 Signature：签名摘要，计算过程详见下文。
X-TC-Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

签名方法 V1 公共参数

使用签名方法 v1（有时会称作 HmacSHA256 和 HmacSHA1），公共参数需要统一放到请求串中。

参数名称	类型	必选	描述
Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702，如果与当前时间相差过大，会引起签名过期错误。
Nonce	Integer	是	随机正整数，与 Timestamp 联合起来，用于防止重放攻击。
SecretId	String	是	在 云API密钥 上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey，而 SecretKey 会用来生成请求签名 Signature。
Signature	String	是	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。具体计算方法参见下文 “签名方法介绍” 。
Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本2017-03-12。
SignatureMethod	String	否	签名方式，目前支持 HmacSHA256和 HmacSHA1。只有指定此参数为 HmacSHA256时，才使用 HmacSHA256算法验证签名，其他情况均使用 HmacSHA1验证签名。
Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token 。

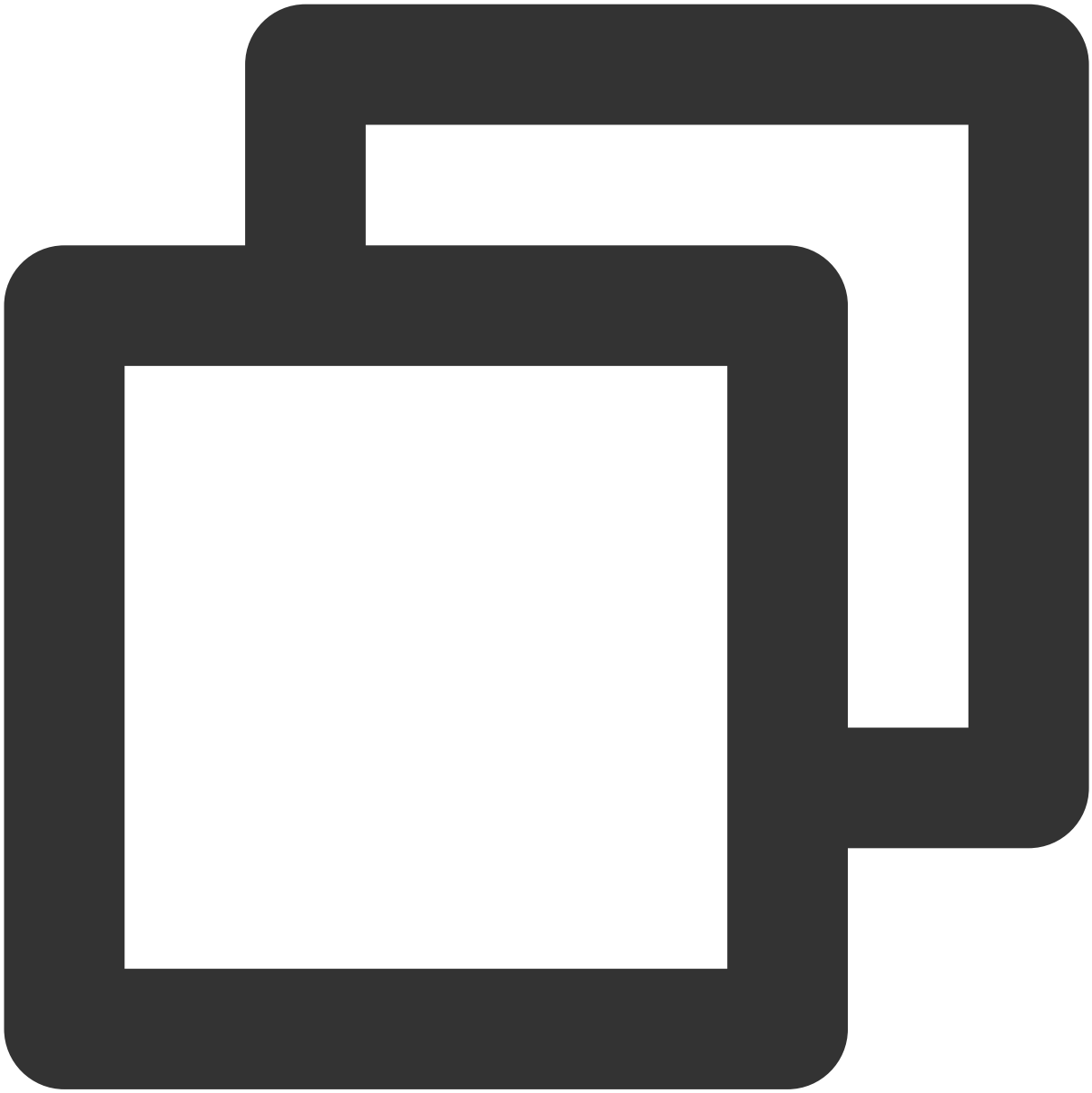
地域列表

由于各个产品支持地域不同，具体详情请参考各产品文档中的地域列表。
例如，您可以参考云服务器的 地域列表。

.NET API 调用方式

腾讯云 API 会对每个请求进行身份验证，用户需要使用安全凭证，经过特定的步骤对请求进行签名（Signature），每个请求都需要在公共请求参数中指定该签名结果并以指定的方式和格式发送请求。

假设用户的 SecretId 和 SecretKey 分别是： `AKIDz8krbsJ5*********mLPx3EXAMPLE` 和 `Gu5t9xGAR**********EXAMPLE` 。用户想查看广州区云服务器名为“未命名”的主机状态，只返回一条数据。则请求可能为：



```
curl -X POST https://cvm.tencentcloudapi.com \\
-H "Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5*********mLPx3EXAMPLE/20
-H "Content-Type: application/json; charset=utf-8" \\
-H "Host: cvm.tencentcloudapi.com" \\
-H "X-TC-Action: DescribeInstances" \\
-H "X-TC-Timestamp: 1551113065" \\
-H "X-TC-Version: 2017-03-12" \\
```

```
-H "X-TC-Region: ap-guangzhou" \\
-d '{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instanc
```

步骤1：申请安全凭证

本文使用的安全凭证为密钥，密钥包括 SecretId 和 SecretKey。每个用户最多可以拥有两对密钥。

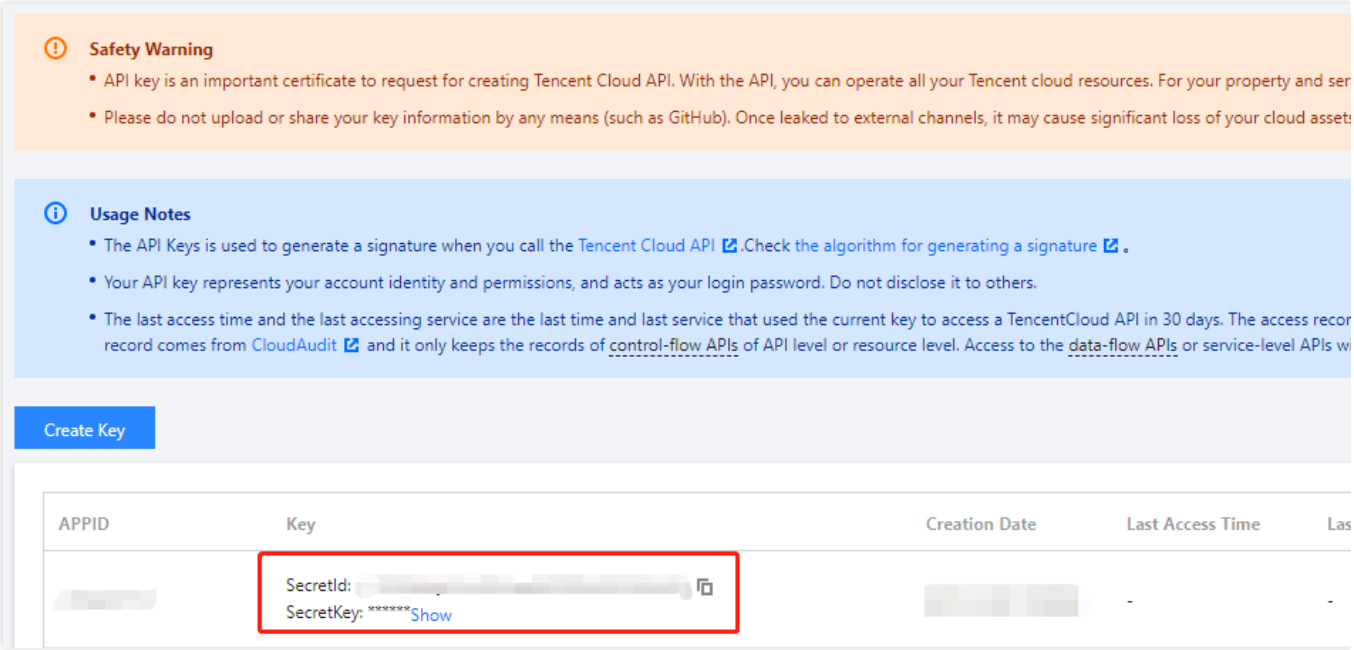
SecretId：用于标识 API 调用者身份，可以简单类比为用户名。

SecretKey：用于验证 API 调用者的身份，可以简单类比为密码。

注意：

用户必须严格保管安全凭证，避免泄露，否则将危及财产安全。如已泄漏，请立刻禁用该安全凭证。

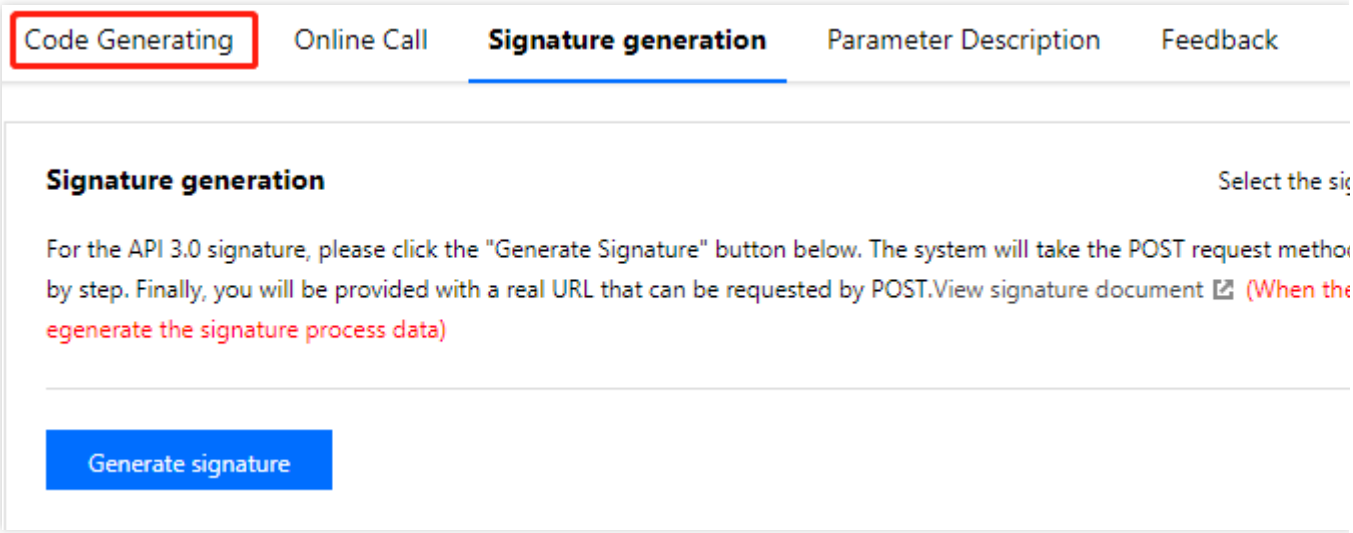
前往 API密钥管理 页面，即可进行获取。如下图所示：



步骤2：

1. 获取 API 3.0 V3 版本签名

签名方法 v3（TC3-HMAC-SHA256）功能上覆盖了以前的签名方法 v1，而且更安全，支持更大的请求，支持 json 格式，性能有一定提升，推荐使用该签名方法计算签名。如下图所示：



说明：

首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“API 3.0签名 v3”，可以生成签名过程进行验证，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的7种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

云 API 支持 GET 和 POST 请求。

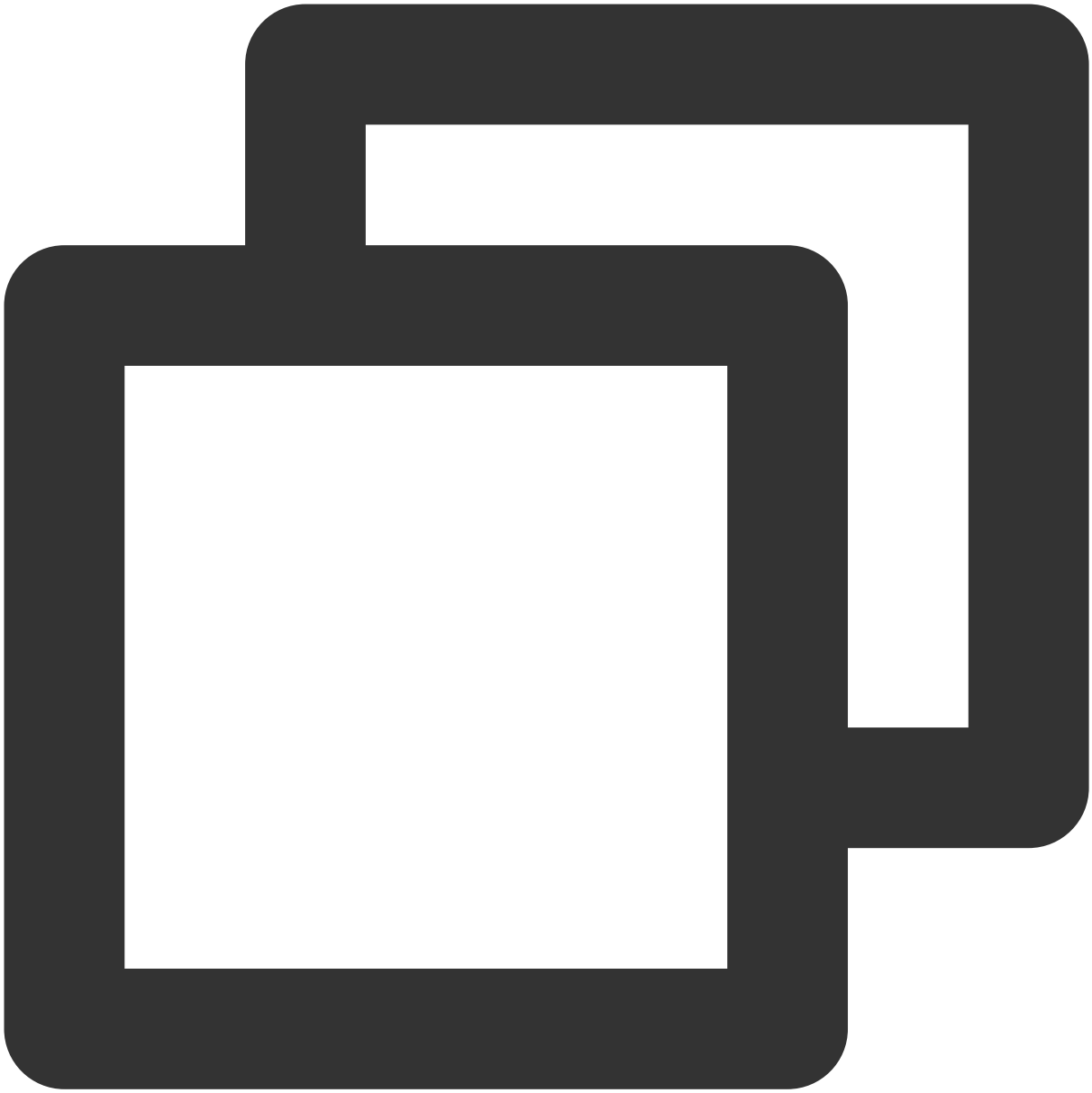
对于GET方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。

对于POST方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式绝大多数接口均支持，multipart 格式只有特定接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。推荐使用 POST 请求，因为两者的结果并无差异，但 GET 请求只支持32KB 以内的请求包。

下面以 查看实例列表 接口为例，分步骤介绍签名的计算过程。我们选择该接口是因为：

1. 云服务器默认已开通，该接口很常用；
2. 该接口是只读的，不会改变现有资源的状态；
3. 接口覆盖的参数种类较全，可以演示包含数据结构的数组如何使用。

1. 拼接规范请求串

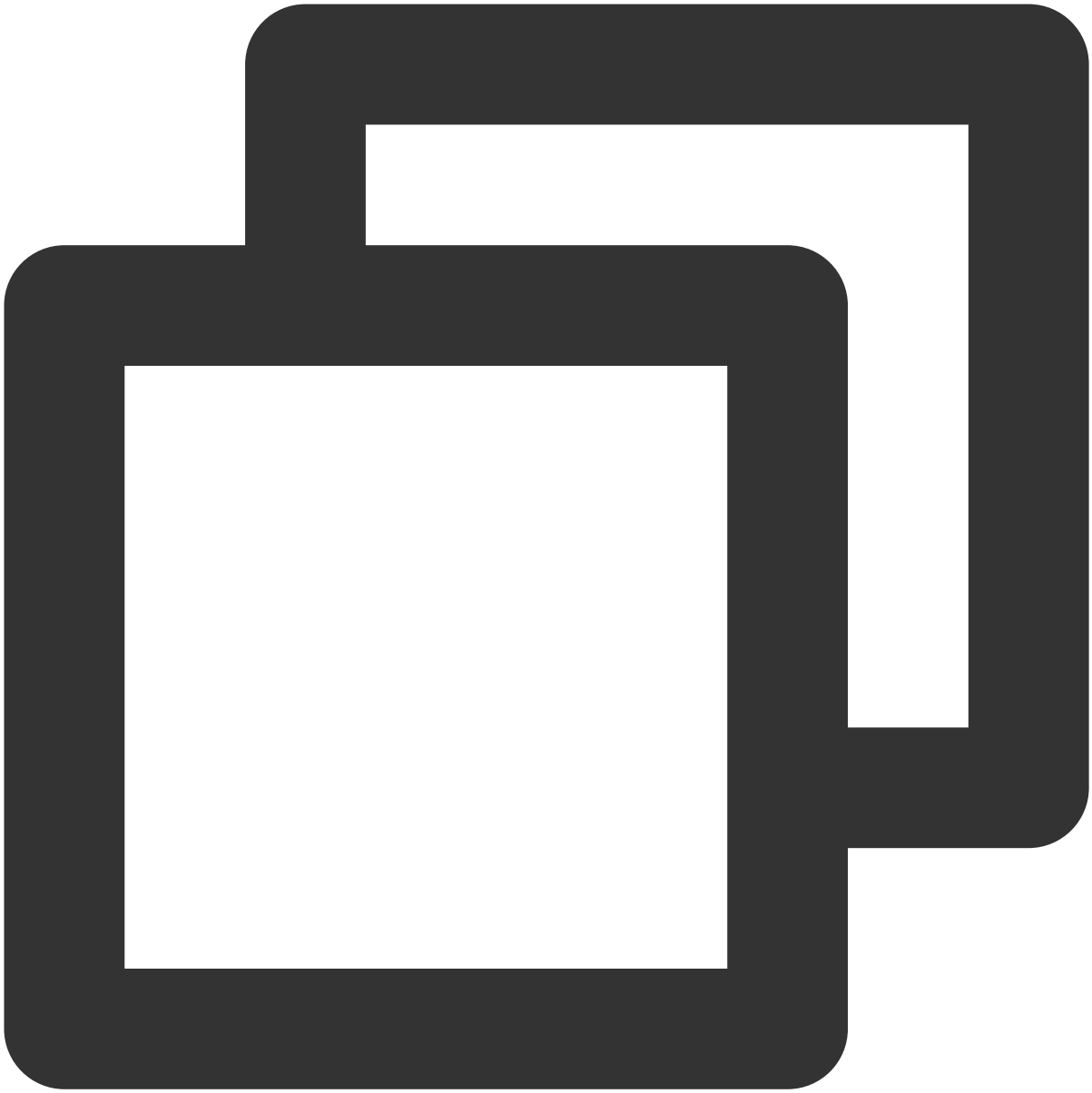


```
CanonicalRequest =
    HTTPRequestMethod + '\\n' +
    CanonicalURI + '\\n' +
    CanonicalQueryString + '\\n' +
    CanonicalHeaders + '\\n' +
    SignedHeaders + '\\n' +
    HashedRequestPayload
```

字段名称	解释

HTTPRequestMethod	HTTP 请求方法（GET、POST ）。此示例取值为 `POST` 。
CanonicalURI	URI 参数，API 3.0 固定为正斜杠（/）。
CanonicalQueryString	发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串""，对于 G 求，则为 URL 中问号（?）后面的字符串内容，例如：Limit=10&Offset=0。 注意：CanonicalQueryString 需要参考 RFC3986 进行 URLEncode，字符集 UTF8，推荐使用 语言标准库，所有特殊字符均需编码，大写形式。
CanonicalHeaders	参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部 签名以提高自身请求的唯一性和安全性。 拼接规则：头部 key 和 value 统一转成小写，掉首尾空格，按照 key:value\\n 格式拼接；多个头部，按照头部 key（小写）的 ASCII 行拼接。此示例计算结果是 `content-type:application/json; charset=ut 8\\nhost:cvm.tencentcloudapi.com\\n` 。 注意：content-type 必须和实际发 符合，有些编程语言网络库即使未指定也会自动添加 charset 值，如果签名时和发送时 致，服务器会返回签名校验失败。
SignedHeaders	参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包 部内容是一一对应的。content-type 和 host 为必选头部。 拼接规则：头部 key 统一转 写；多个头部 key（小写）按照 ASCII 升序进行拼接，并且以分号（;）分隔。此示例 content-type;host
HashedRequestPayload	请求正文（Requestpayload，即 body，此示例为 `{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-name"` 的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(RequestPayload))), HTTP 请求正文做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。 GET 请求，RequestPayload 固定为空字符串。此示例计算结果是 `35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f`

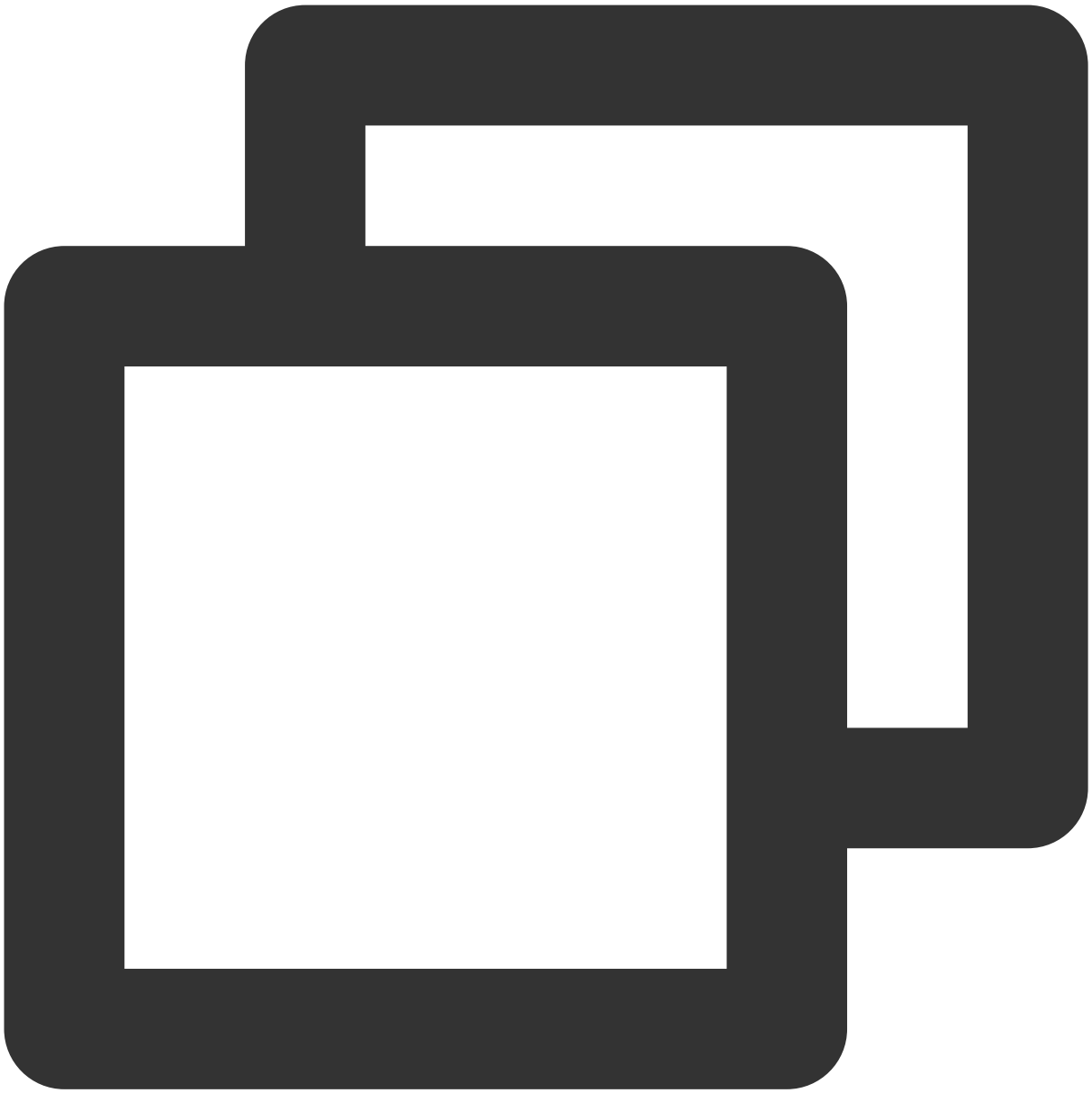
根据以上规则，示例中得到的规范请求串如下：



```
POST
/

content-type:application/json; charset=utf-8
host:cvm.tencentcloudapi.com

content-type;host
35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f064
```

2. 拼接待签名字符串

按如下格式拼接待签名字符串：



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

字段名称	解释
Algorithm	签名算法，目前固定为 `TC3-HMAC-SHA256` 。

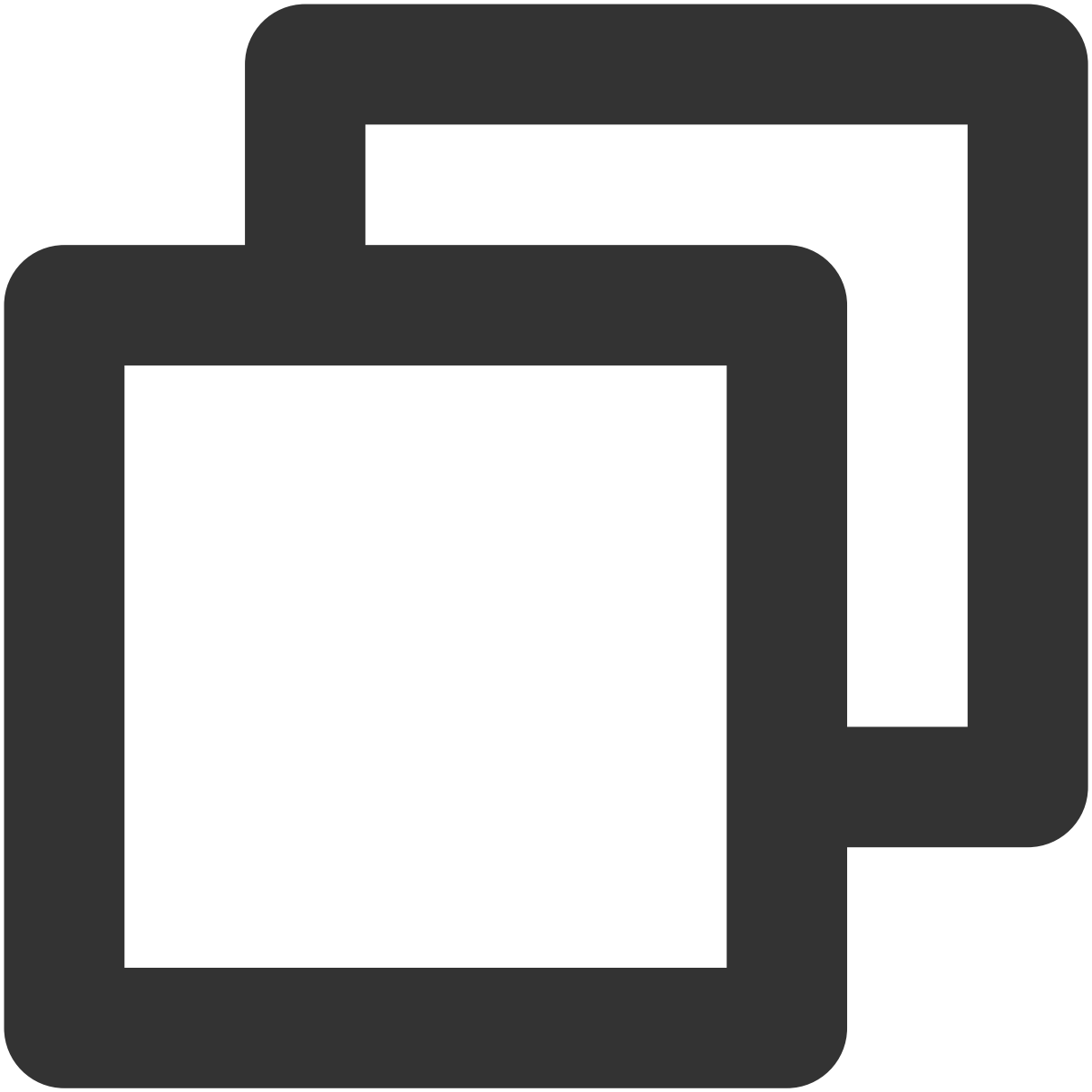
RequestTimestamp	请求时间戳，即请求头部的公共参数 X-TC-Timestamp 取值，取当前时间 UNIX 时间确到秒。此示例取值为 `1551113065` 。
CredentialScope	凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符串（tc3_request）。**Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Time 换算的 UTC 标准时间日期一致**；service 为产品名，必须与调用的产品域名一致。此算结果是 2019-02-25/cvm/tc3_request。
HashedCanonicalRequest	前述步骤拼接所得规范请求串的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))。此示例计算结果是 `5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d`

注意：

Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为 1551113065，在东八区的时间是 2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为 2019-02-25，而不是 2019-02-26。

Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败，返回签名过期错误。

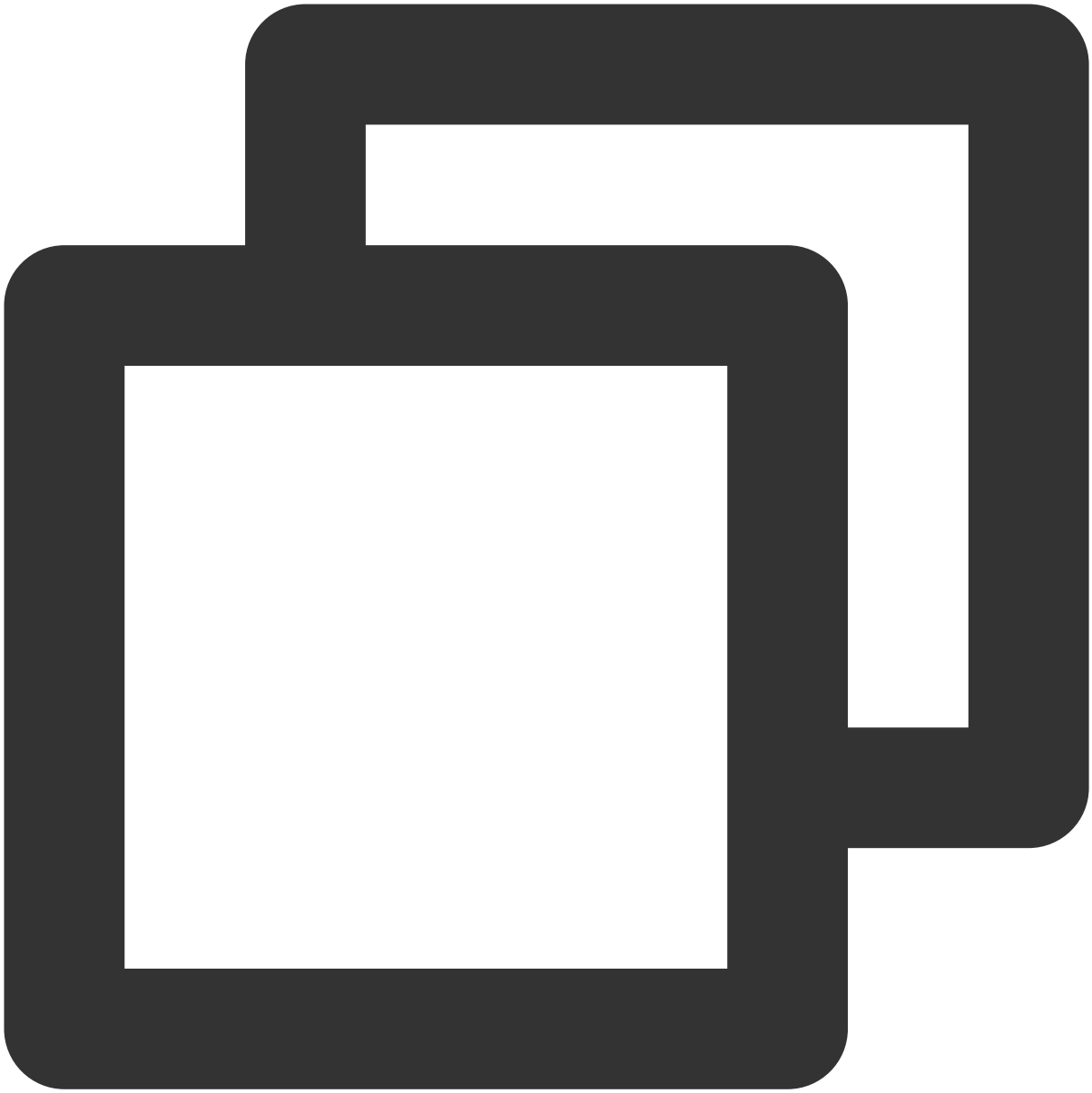
根据以上规则，示例中得到的待签名字符串如下：



```
TC3-HMAC-SHA256
1551113065
2019-02-25/cvm/tc3_request
5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d7031
```

3.计算签名(伪代码)

实际请参考下面示例：

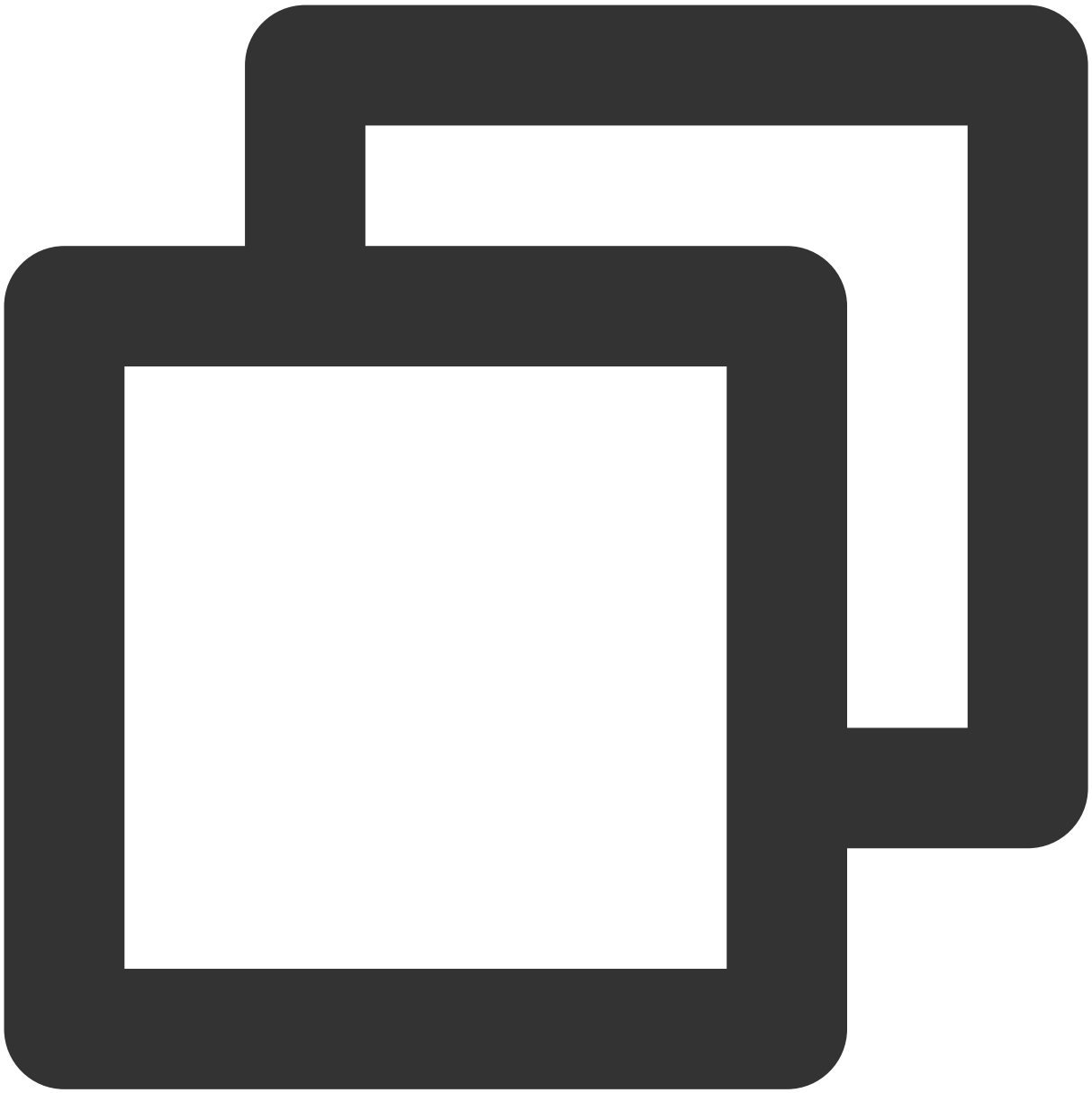


```
byte[] tc3SecretKey = Encoding.UTF8.GetBytes("TC3" + SECRET_KEY);
byte[] secretDate = HmacSHA256(tc3SecretKey, Encoding.UTF8.GetBytes(date));
byte[] secretService = HmacSHA256(secretDate, Encoding.UTF8.GetBytes(service));
byte[] secretSigning = HmacSHA256(secretService, Encoding.UTF8.GetBytes("tc3_reques
byte[] signatureBytes = HmacSHA256(secretSigning, Encoding.UTF8.GetBytes(stringToSi
string signature = BitConverter.ToString(signatureBytes).Replace("-", "").ToLower()
```

此示例计算结果是 `72e494ea8*****************************************a96525168` 。

4. 拼接 Authorization

按如下格式拼接 Authorization：



```
Authorization =
    Algorithm + ' ' +
    'Credential=' + SecretId + '/' + CredentialScope + ', ' +
    'SignedHeaders=' + SignedHeaders + ', ' +
    'Signature=' + Signature
```

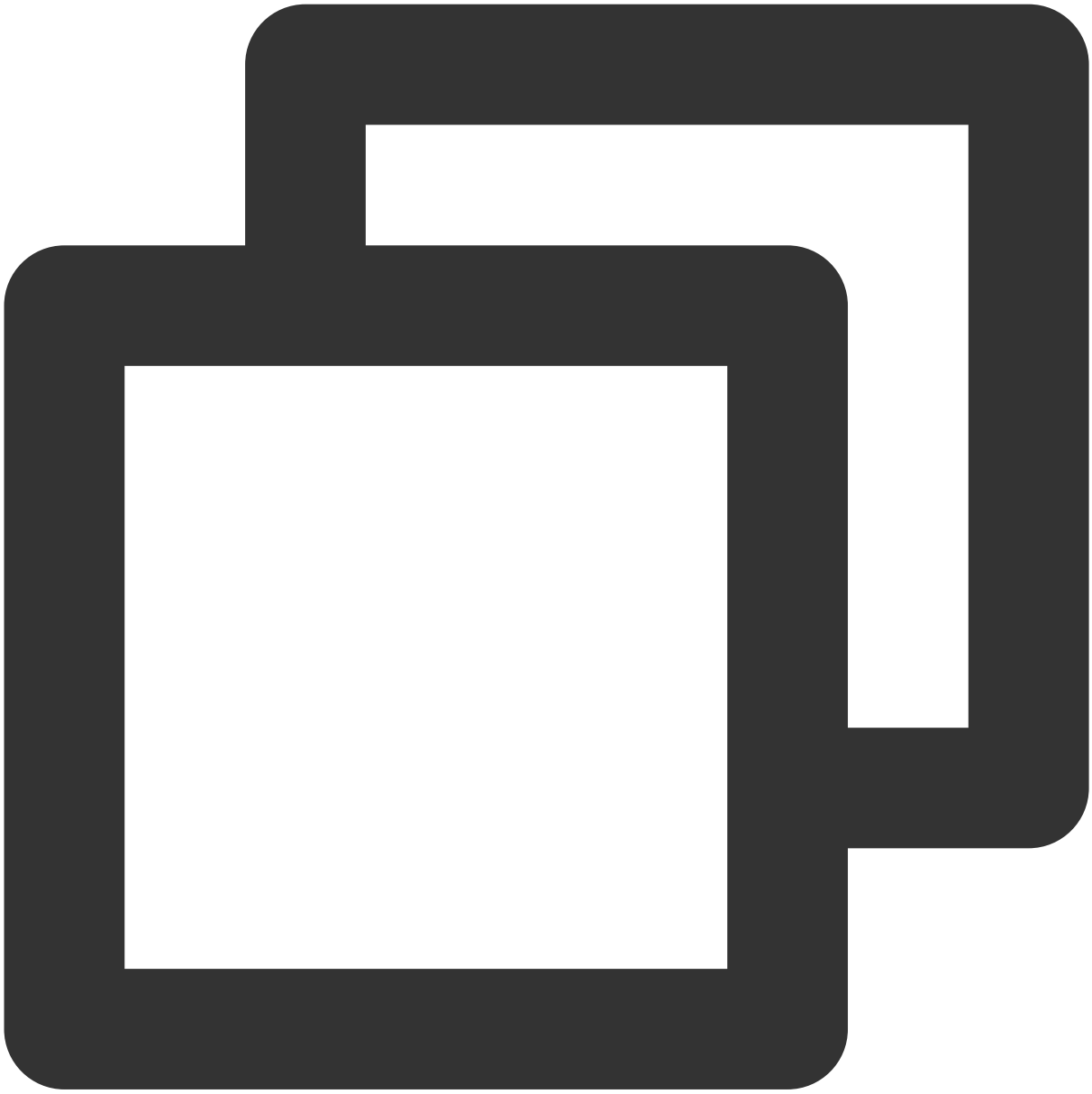
字段名称	解释
Algorithm	签名方法，固定为 `TC3-HMAC-SHA256` 。

SecretId	密钥对中的 SecretId，即 `AKIDz8krbsJ5**********mLPx3EXAMPLE` 。
CredentialScope	见上文，凭证范围。此示例计算结果是 `2019-02-25/cvm/tc3_request` 。
SignedHeaders	见上文，参与签名的头部信息。此示例取值为 `content-type;host` 。
Signature	签名值。此示例计算结果是 `72e494ea8****************************************a96525168` 。

根据以上规则，示例中得到的值为：

```
TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/2019-02-25/cvm/tc3_re
```

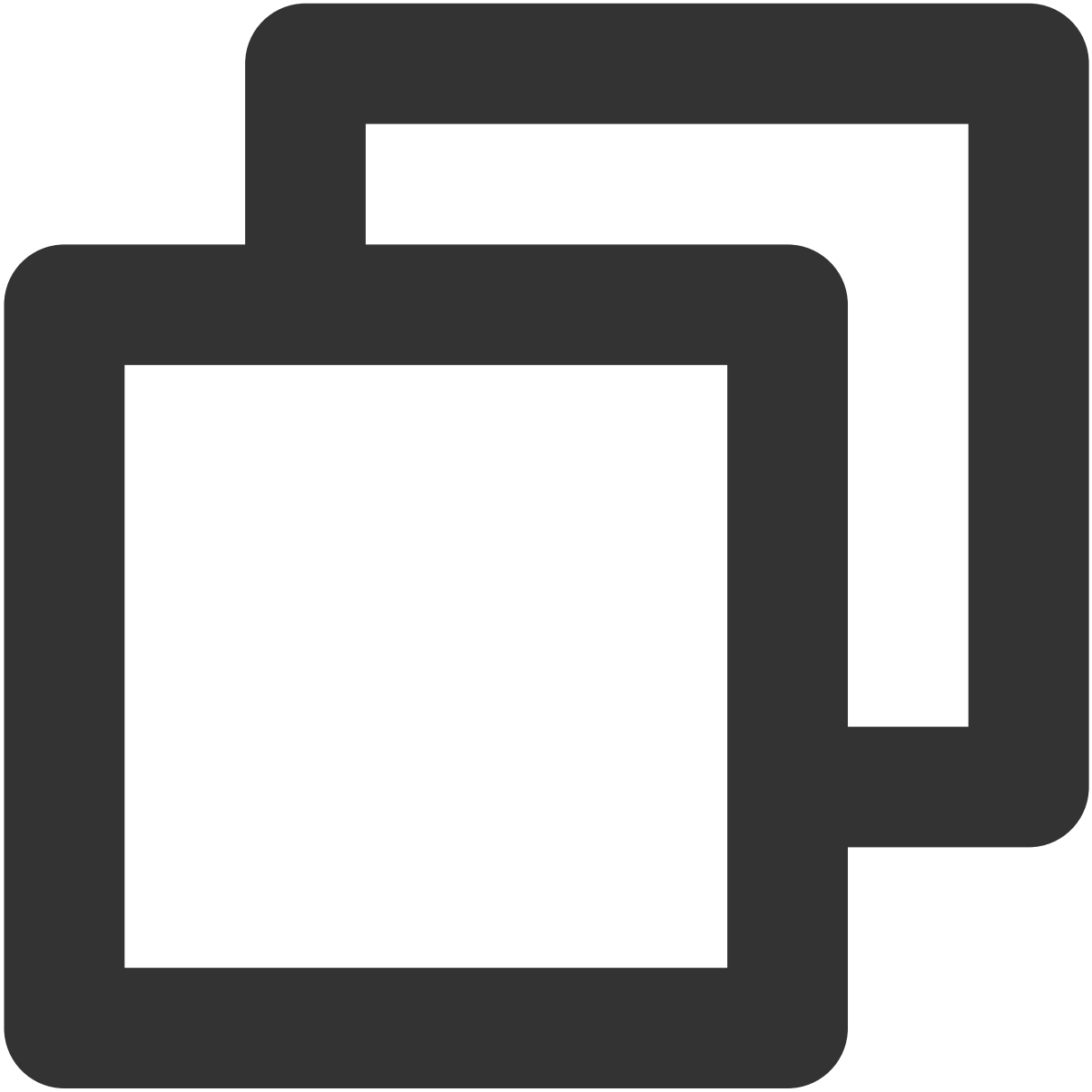
最终完整的调用信息如下：



```
POST https://cvm.tencentcloudapi.com/
Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/2019-0
Content-Type: application/json; charset=utf-8
Host: cvm.tencentcloudapi.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1551113065
```

```
X-TC-Region: ap-guangzhou

{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-na
```

5. API 3.0签名 V3示例



```
using System;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Text;
```

```
public class Application {
    public static string SHA256Hex(string s)
    {
        using (SHA256 algo = SHA256.Create())
        {
            byte[] hashbytes = algo.ComputeHash(Encoding.UTF8.GetBytes(s));
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < hashbytes.Length; ++i)
            {
                builder.Append(hashbytes[i].ToString("x2"));
            }
            return builder.ToString();
        }
    }
    public static byte[] HmacSHA256(byte[] key, byte[] msg)
    {
        using (HMACSHA256 mac = new HMACSHA256(key))
        {
            return mac.ComputeHash(msg);
        }
    }
    public static void Main(string[] args)
    {

        // 密钥参数
        string SECRET_ID = "AKIDz8krbsJ5**********mLPx3EXAMPLE";
        string SECRET_KEY = "Gu5t9xGAR***********EXAMPLE";


        string service = "cvm";
        string endpoint = "cvm.tencentcloudapi.com";
        string region = "ap-guangzhou";
        string action = "DescribeInstances";
        string version = "2017-03-12";
        string algorithm = "TC3-HMAC-SHA256";
        string contentType = "application/json";
        double RequestTimestamp = 1551113065;  // 时间戳 2019-02-26 00:44:25,此参数作
        // long timestamp = ToTimestamp() / 1000;
        // string requestTimestamp = timestamp.ToString();
        string date = new DateTime(1970, 1, 1, 0, 0, 0, 0, DateTimeKind.Utc).AddSec
        // 注意时区，否则容易出错

        // ************* 步骤 1：拼接规范请求串 *************
        string httpRequestMethod = "POST";
        string canonicalUri = "/";
        string canonicalQueryString = "";
        string canonicalHeaders = "content-type:" + contentType +"; charset=utf-8\\
```

```
        string signedHeaders = "content-type;host";
        string requestPayload = "{\\"Limit\\": 1, \\"Filters\\": [{\\"Values\\": [\
        string hashedRequestPayload = SHA256Hex(requestPayload);
        string canonicalRequest = httpRequestMethod + "\\n"
            + canonicalUri + "\\n"
            + canonicalQueryString + "\\n"
            + canonicalHeaders + "\\n"
            + signedHeaders + "\\n"
            + hashedRequestPayload;
        Console.WriteLine(canonicalRequest);
        Console.WriteLine("-----------------------------------");

        // ************* 步骤 2：拼接待签名字符串 *************
        string credentialScope = date + "/" + service + "/" + "tc3_request";
        string hashedCanonicalRequest = SHA256Hex(canonicalRequest);
        string stringToSign = algorithm + "\\n" + RequestTimestamp + "\\n" + creden
        Console.WriteLine(stringToSign);
        Console.WriteLine("-----------------------------------");

        // ************* 步骤 3：计算签名 *************
        byte[] tc3SecretKey = Encoding.UTF8.GetBytes("TC3" + SECRET_KEY);
        byte[] secretDate = HmacSHA256(tc3SecretKey, Encoding.UTF8.GetBytes(date));
        byte[] secretService = HmacSHA256(secretDate, Encoding.UTF8.GetBytes(servic
        byte[] secretSigning = HmacSHA256(secretService, Encoding.UTF8.GetBytes("tc
        byte[] signatureBytes = HmacSHA256(secretSigning, Encoding.UTF8.GetBytes(st
        string signature = BitConverter.ToString(signatureBytes).Replace("-", "").T
        Console.WriteLine(signature);
        Console.WriteLine("-----------------------------------");

        // ************* 步骤 4：拼接 Authorization *************
        string authorization = algorithm + " "
            + "Credential=" + SECRET_ID + "/" + credentialScope + ", "
            + "SignedHeaders=" + signedHeaders + ", "
            + "Signature=" + signature;
        Console.WriteLine(authorization);
        Console.WriteLine("-----------------------------------");

        Dictionary<string, string> headers = new Dictionary<string, string>();
        headers.Add("Authorization", authorization);
        headers.Add("Host", endpoint);
        headers.Add("Content-Type", contentType + "; charset=utf-8");
        headers.Add("X-TC-Timestamp", RequestTimestamp.ToString());
        headers.Add("X-TC-Version", version);
        headers.Add("X-TC-Action", action);
        headers.Add("X-TC-Region", region);
        Console.WriteLine("POST https://cvm.tencentcloudapi.com");
        foreach (KeyValuePair<string, string> kv in headers)
```

```
        {
            Console.WriteLine(kv.Key + ": " + kv.Value);
        }
        Console.WriteLine();
        Console.WriteLine(requestPayload);
    }
}
```

2. 获取 API 3.0 V1 版本签名

签名方法 v1（HmacSHA1、HmacSHA256）简单易用，但是功能和安全性都不如签名方法 v3，推荐使用签名方法 v3。

首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“API 3.0签名 v1”，可以生成签名过程进行验证，并提供了部分编程语言的签名示例，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的 7 种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

以云服务器查看实例列表（DescribeInstances）请求为例，当用户调用这一接口时，其请求参数可能如下:

参数名称	中文	参数值
Action	方法名	DescribeInstances
SecretId	密钥 ID	`AKIDz8krbsJ5*********mLPx3EXAMPLE`
Timestamp	当前时间戳	1465185768
Nonce	随机正整数	11886
Region	实例所在区域	ap-guangzhou
InstanceIds.0	待查询的实例 ID	ins-09dx96dg
Offset	偏移量	0
Limit	最大允许输出	20
Version	接口版本号	2017-03-12

1. 对参数排序

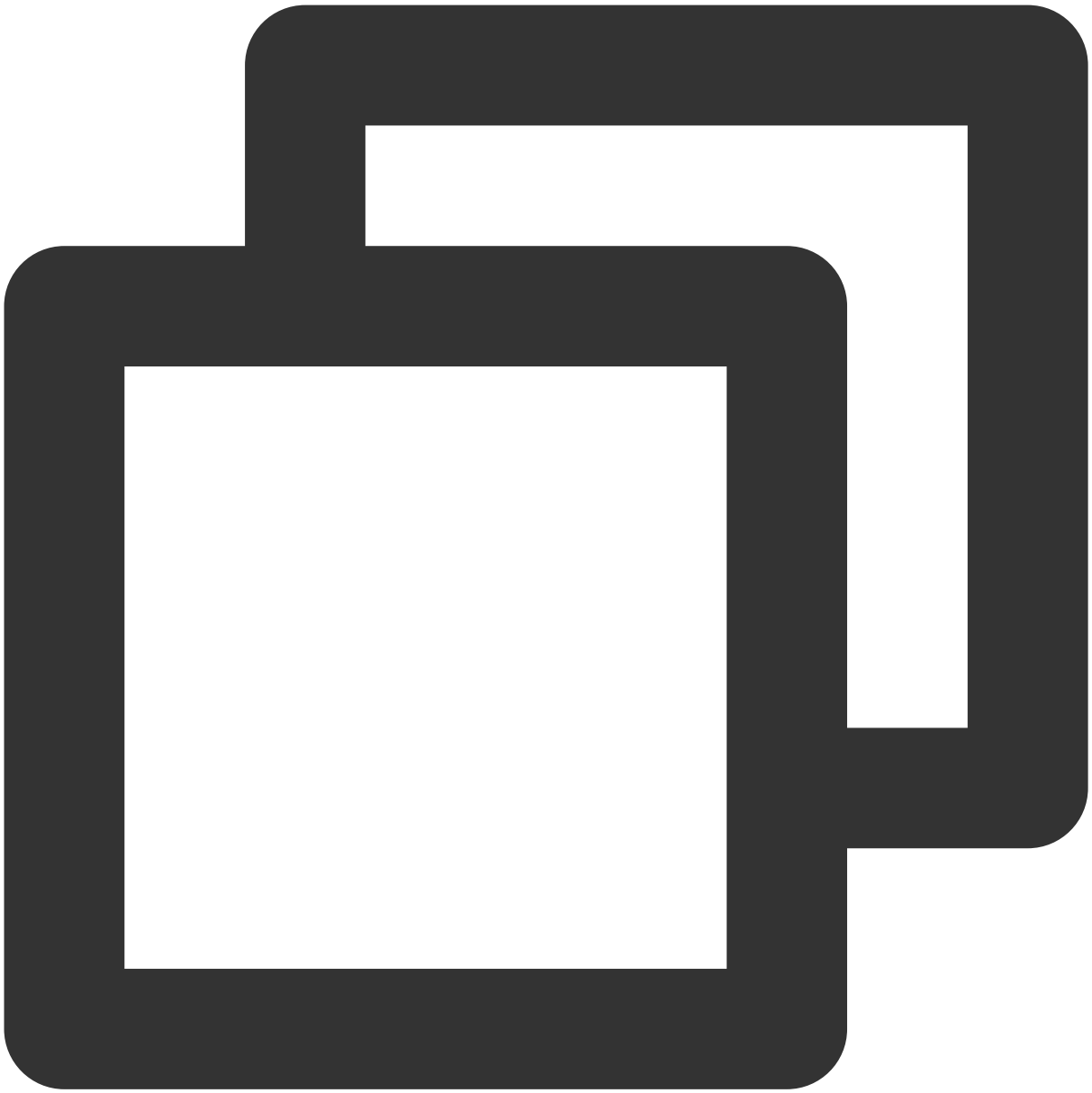
首先对所有请求参数按参数名的字典序（ASCII 码）升序排序。

注意：

只按参数名进行排序，参数值保持对应即可，不参与比大小。

按 ASCII 码比大小，如 InstanceIds.2要排在 InstanceIds.12后面，不是按字母表，也不是按数值。用户可以借助编程语言中的相关排序函数来实现这一功能，如 PHP 中的 ksort 函数。

上述示例参数的排序结果如下：



```
{
    'Action' : 'DescribeInstances',
    'InstanceIds.0' : 'ins-09dx96dg',
    'Limit' : 20,
    'Nonce' : 11886,
    'Offset' : 0,
    'Region' : 'ap-guangzhou',
    'SecretId' : 'AKIDz8krbsJ5**********mLPx3EXAMPLE',
    'Timestamp' : 1465185768,
```

```
    'Version': '2017-03-12',
}
```

使用程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

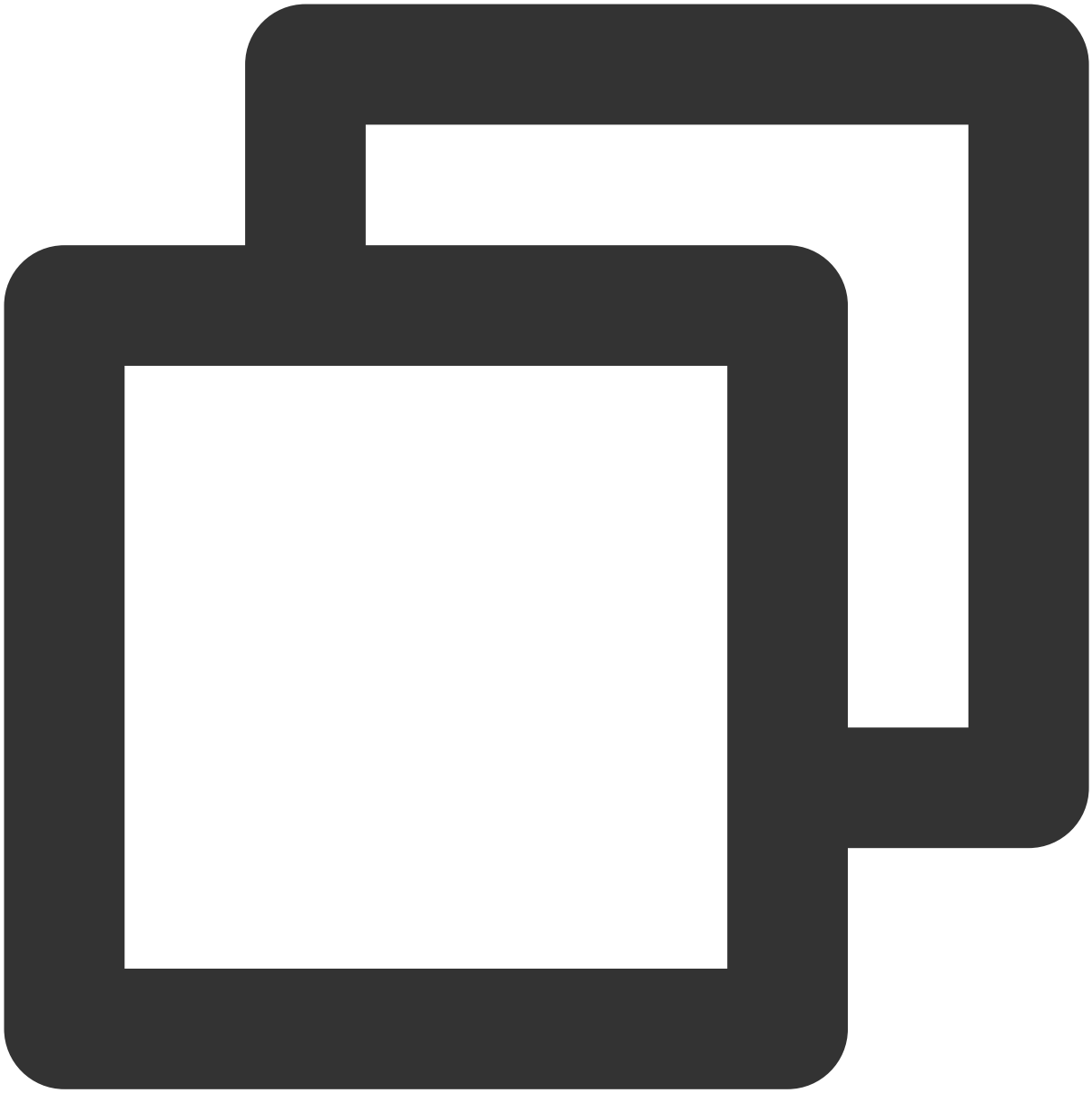
2. 拼接规范请求串

此步骤生成请求字符串。 将把上一步排序好的请求参数格式化成“参数名称=参数值”的形式，如对 Action 参数，其参数名称为 "Action"，参数值为 "DescribeInstances"，因此格式化后就为 Action=DescribeInstances。

注意：

“参数值”为原始值而非 url 编码后的值。

然后将格式化后的各个参数用"&"拼接在一起，最终生成的请求字符串为:



```
Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&R
```

3. 拼接待签名字符串

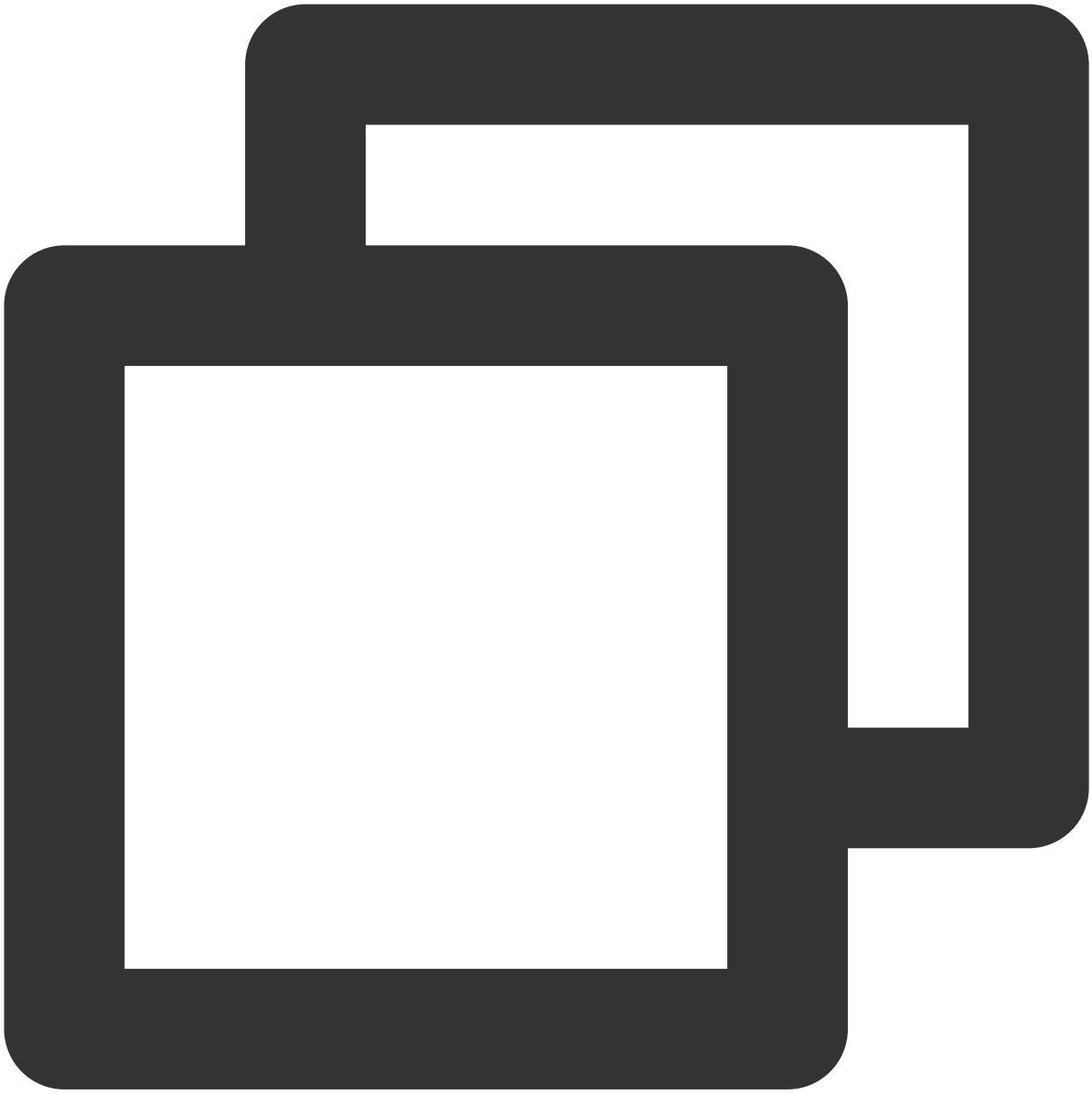
此步骤生成签名原文字符串。 签名原文字符串由以下几个参数构成：

1. 请求方法: 支持 POST 和 GET 方式，这里使用 GET 请求，注意方法为全大写。
2. 请求主机:查看实例列表（DescribeInstances）的请求域名为： `cvm.tencentcloudapi.com` 。实际的请求域名根据接口所属模块的不同而不同，详见各接口说明。
3. 请求路径: 当前版本云 API 的请求路径固定为 / 。

4. 请求字符串: 即上一步生成的请求字符串。

签名原文串的拼接规则为： `请求方法 + 请求主机 +请求路径 + ? + 请求字符串` 。

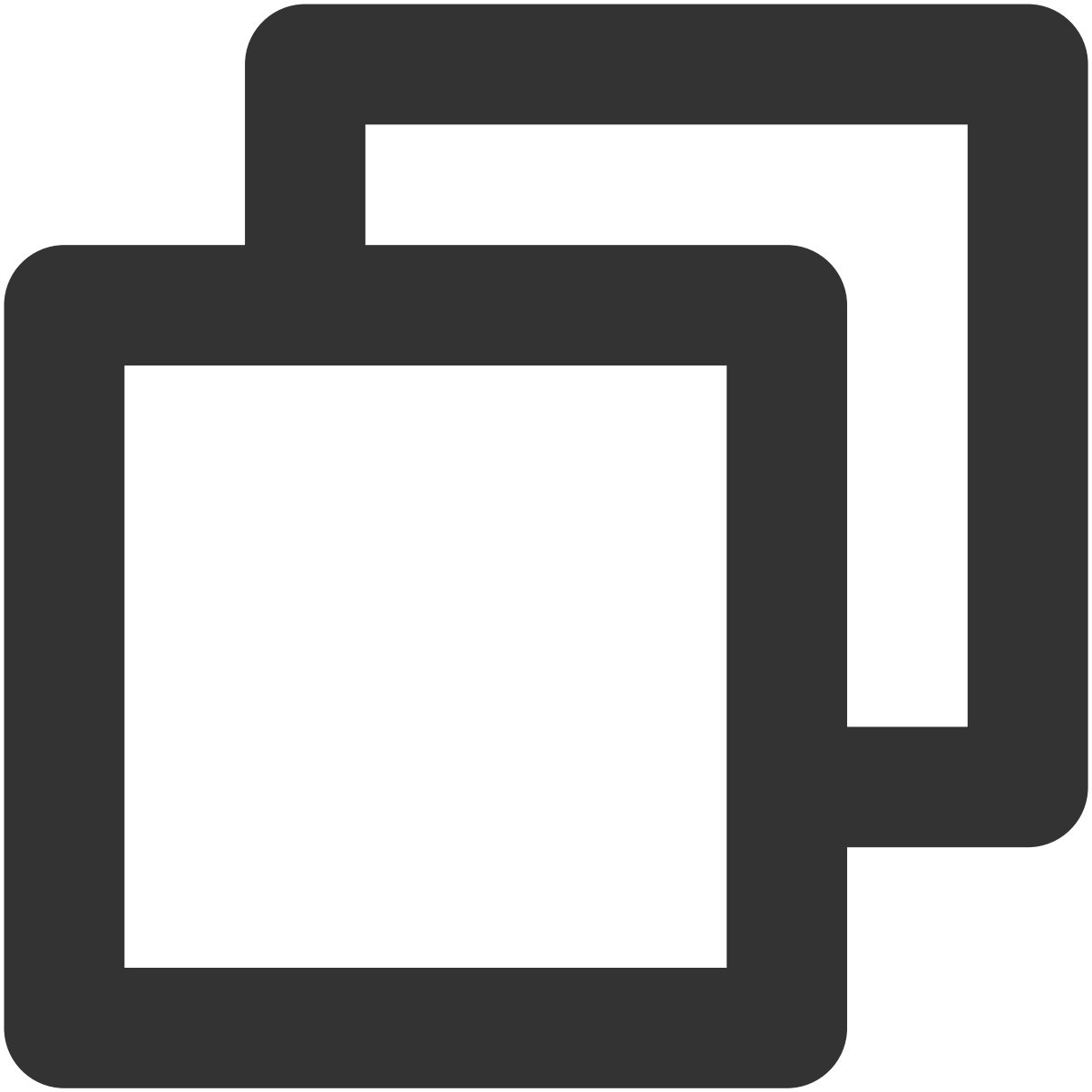
示例的拼接结果为：



```
GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Lim
```

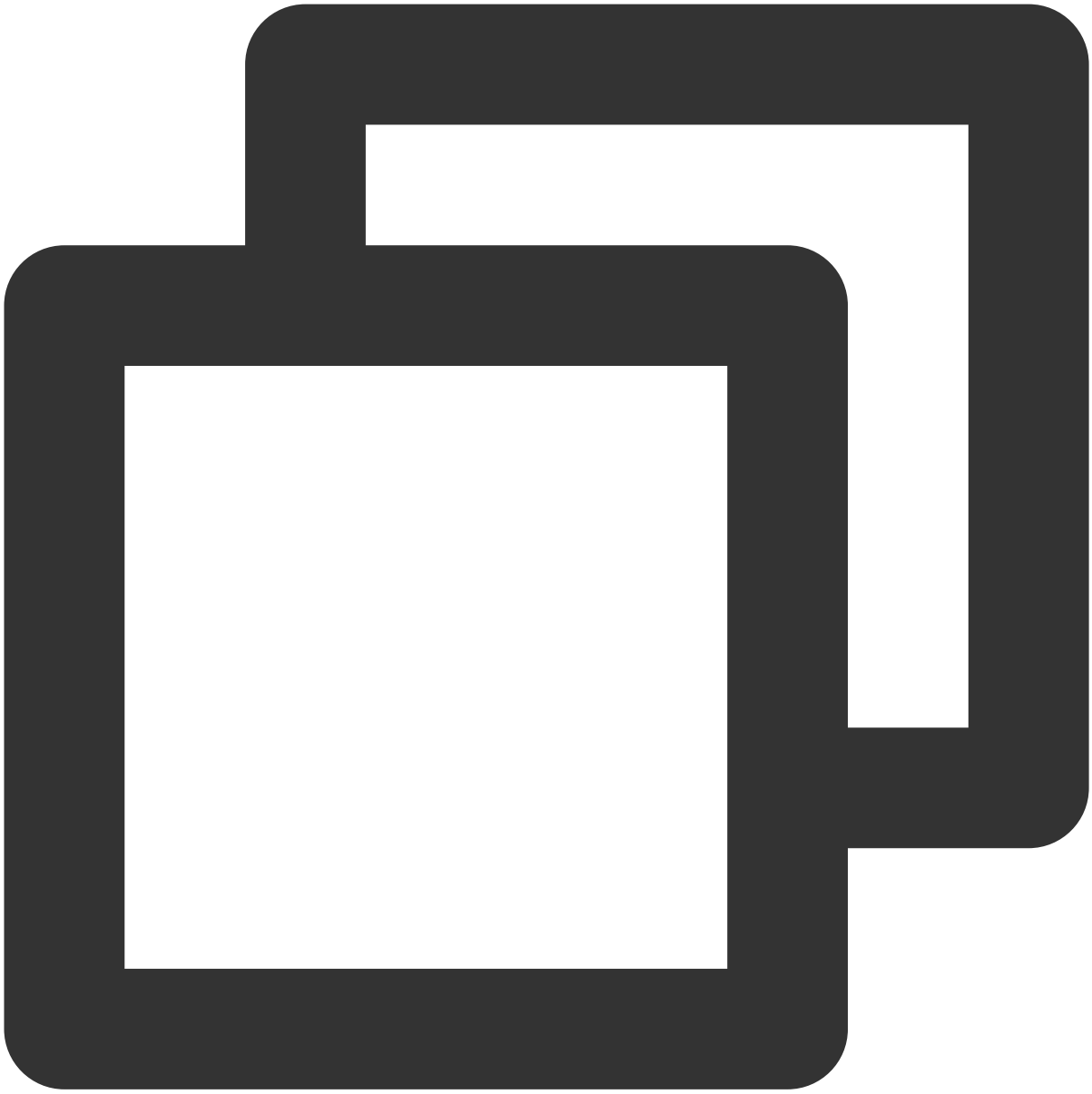
4. 计算签名（伪代码）

此步骤生成签名串。 首先使用 HMAC-SHA1 算法对上一步中获得的**签名原文字符串**进行签名，然后将生成的签名串使用 Base64进行编码，即可获得最终的签名串。

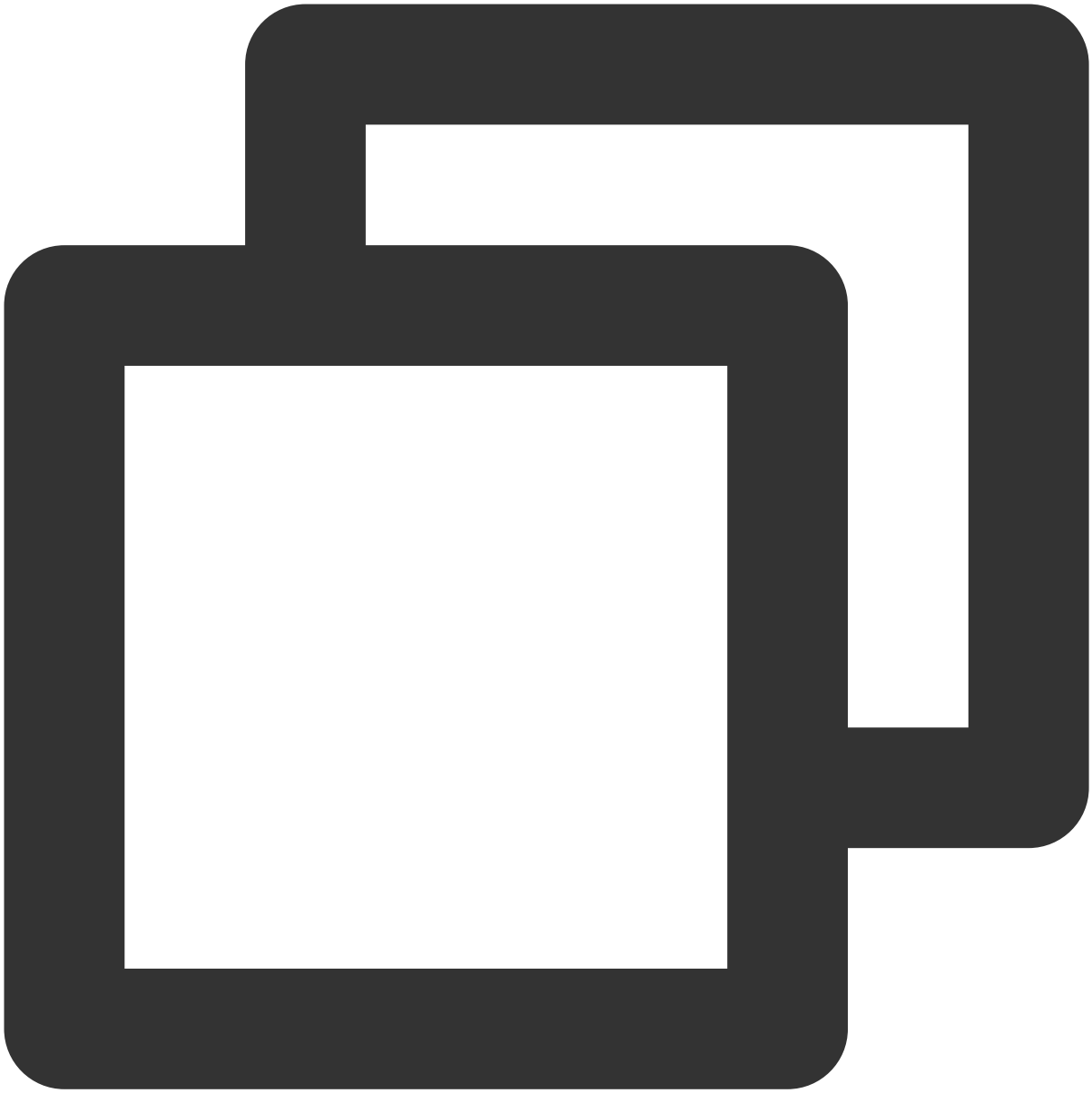


```
public static string Sign(string signKey, string secret, string SignatureMethod)
    {
        string signRet = string.Empty;
        using (HMACSHA1 mac = new HMACSHA1(Encoding.UTF8.GetBytes(signKey)))
        {
            byte[] hash = mac.ComputeHash(Encoding.UTF8.GetBytes(secret));
            signRet = Convert.ToBase64String(hash);
        }
        return signRet;
    }
```

5. 获取调用信息并发送请求



```
# 实际调用，成功后可能如果是消费接口会产生计费(此处以Python语言为例发送get请求)
resp = requests.get("https://" + endpoint, params=data)
print(resp.url)
```



最终得到的请求串为
`https://cvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96d`

字段名称	解释
endpoint	服务地址， 例如： `cvm.tencentcloudapi.com`
data	API 3.0签名 V1所举示例接口参数，**注意** 这里需要将计算的签名已键值对的形式加入 data 中

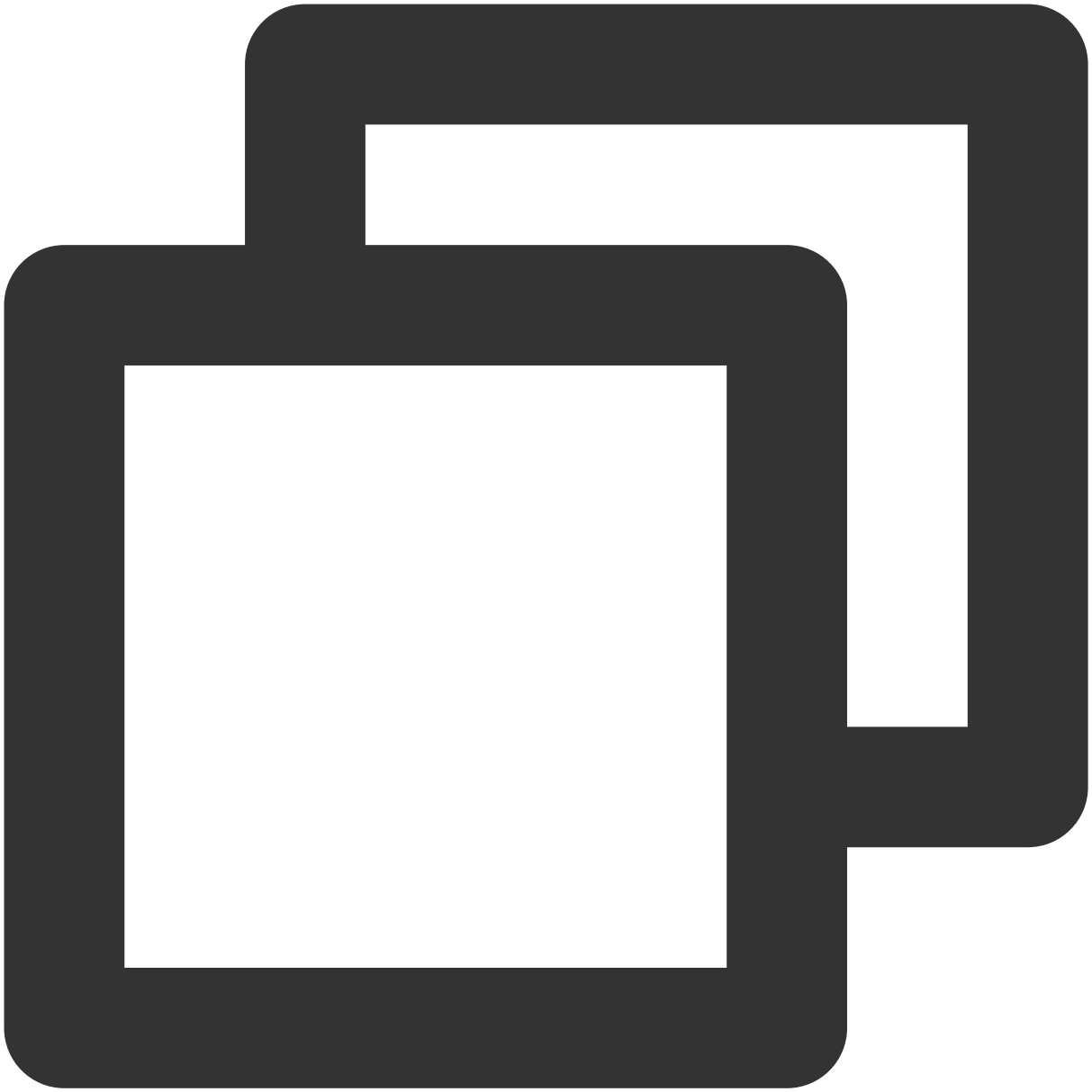
注意：

由于示例中的密钥是虚构的，时间戳也不是系统当前时间，因此如果将此 url 在浏览器中打开或者用 curl 等命令调用时会返回鉴权错误：签名过期。为了得到一个可以正常返回的 url ，需要修改示例中的 SecretId 和 SecretKey 为真实的密钥，并使用系统当前时间戳作为 Timestamp。
为了更清楚的解释签名过程，下面以 .NET 语言为例，将上述的签名过程具体实现。请求的域名、调用的接口和参数的取值都以上述签名过程为准，代码只为解释签名过程，并不具备通用性，实际开发请尽量使用 SDK。

6. 签名串编码

生成的签名串并不能直接作为请求参数，需要对其进行 URL 编码。
如上一步生成的签名串为 `Eli****************cGeI=` ，最终得到的签名串请求参数（Signature）为： `EliP**********************eI%3D` ，它将用于生成最终的请求 URL。
注意：
如果用户的请求方法是 GET，或者请求方法为 POST 同时 Content-Type 为 application/x-www-form-urlencoded，则发送请求时所有请求参数的值均需要做 URL 编码，参数键和=符号不需要编码。非 ASCII 字符在 URL 编码前需要先以 UTF-8 进行编码。
有些编程语言的网络库会自动为所有参数进行 urlencode，在这种情况下，就不需要对签名串进行 URL 编码了，否则两次 URL 编码会导致签名失败。
其他参数值也需要进行编码，编码采用 RFC 3986。使用 %XY 对特殊字符例如汉字进行百分比编码，其中“X”和“Y”为十六进制字符（0-9 和大写字母 A-F），使用小写将引发错误。

7. API 3.0签名 V1示例



```
using System;
using System.Collections.Generic;
using System.Net;
using System.Security.Cryptography;
using System.Text;

public class Application {
    public static string Sign(string signKey, string secret)
    {
        string signRet = string.Empty;
            using (HMACSHA1 mac = new HMACSHA1(Encoding.UTF8.GetBytes(signKey)))
```

```
            {
                byte[] hash = mac.ComputeHash(Encoding.UTF8.GetBytes(secret));
                signRet = Convert.ToBase64String(hash);
            }
        return signRet;
    }
    public static string MakeSignPlainText(SortedDictionary<string, string> request
    {
        string retStr = "";
        retStr += requestMethod;
        retStr += requestHost;
        retStr += requestPath;
        retStr += "?";
        string v = "";
        foreach (string key in requestParams.Keys)
        {
            v += string.Format("{0}={1}&", key, requestParams[key]);
        }
        retStr += v.TrimEnd('&');
        return retStr;
    }

    public static void Main(string[] args)
    {
        // 密钥参数
        string SECRET_ID = "AKIDz8krbsJ5**********mLPx3EXAMPLE";
        string SECRET_KEY = "Gu5t9xGAR***********EXAMPLE";

        string endpoint = "cvm.tencentcloudapi.com";
        string region = "ap-guangzhou";
        string action = "DescribeInstances";
        string version = "2017-03-12";
        double RequestTimestamp = 1465185768;  // 时间戳 2019-02-26 00:44:25,此参数作
        // long timestamp = ToTimestamp() / 1000;
        // string requestTimestamp = timestamp.ToString();
        Dictionary<string, string> param = new Dictionary<string, string>();
        param.Add("Limit", "20");
        param.Add("Offset", "0");
        param.Add("InstanceIds.0", "ins-09dx96dg");
        param.Add("Action", action);
        param.Add("Nonce", "11886");
        // param.Add("Nonce", Math.Abs(new Random().Next()).ToString());

        param.Add("Timestamp", RequestTimestamp.ToString());
        param.Add("Version", version);

        param.Add("SecretId", SECRET_ID);
```

```
        param.Add("Region", region);
        SortedDictionary<string, string> headers = new SortedDictionary<string, str
        string sigInParam = MakeSignPlainText(headers, "GET", endpoint, "/");
        Console.WriteLine(sigInParam);
        string sigOutParam = Sign(SECRET_KEY, sigInParam);


        Console.WriteLine("GET https://cvm.tencentcloudapi.com");
        foreach (KeyValuePair<string, string> kv in headers)
        {
            Console.WriteLine(kv.Key + ": " + kv.Value);
        }
        Console.WriteLine("Signature" + ": " + WebUtility.UrlEncode(sigOutParam));
        Console.WriteLine();

        string result = "https://cvm.tencentcloudapi.com/?";
        foreach (KeyValuePair<string, string> kv in headers)
        {
            result += WebUtility.UrlEncode(kv.Key) + "=" + WebUtility.UrlEncode(kv.
        }
        result += WebUtility.UrlEncode("Signature") + "=" + WebUtility.UrlEncode(si
        Console.WriteLine("GET " + result);
    }
}
```

API 2.0签名

此签名现已不在维护，建议使用性能更优的 **API 3.0签名**，如需使用，请直接在 API Explorer 的**签名串生成 > 选择 API 2.0签名** 版本进行操作。

签名失败

存在以下签名失败的错误码，请根据实际情况处理。

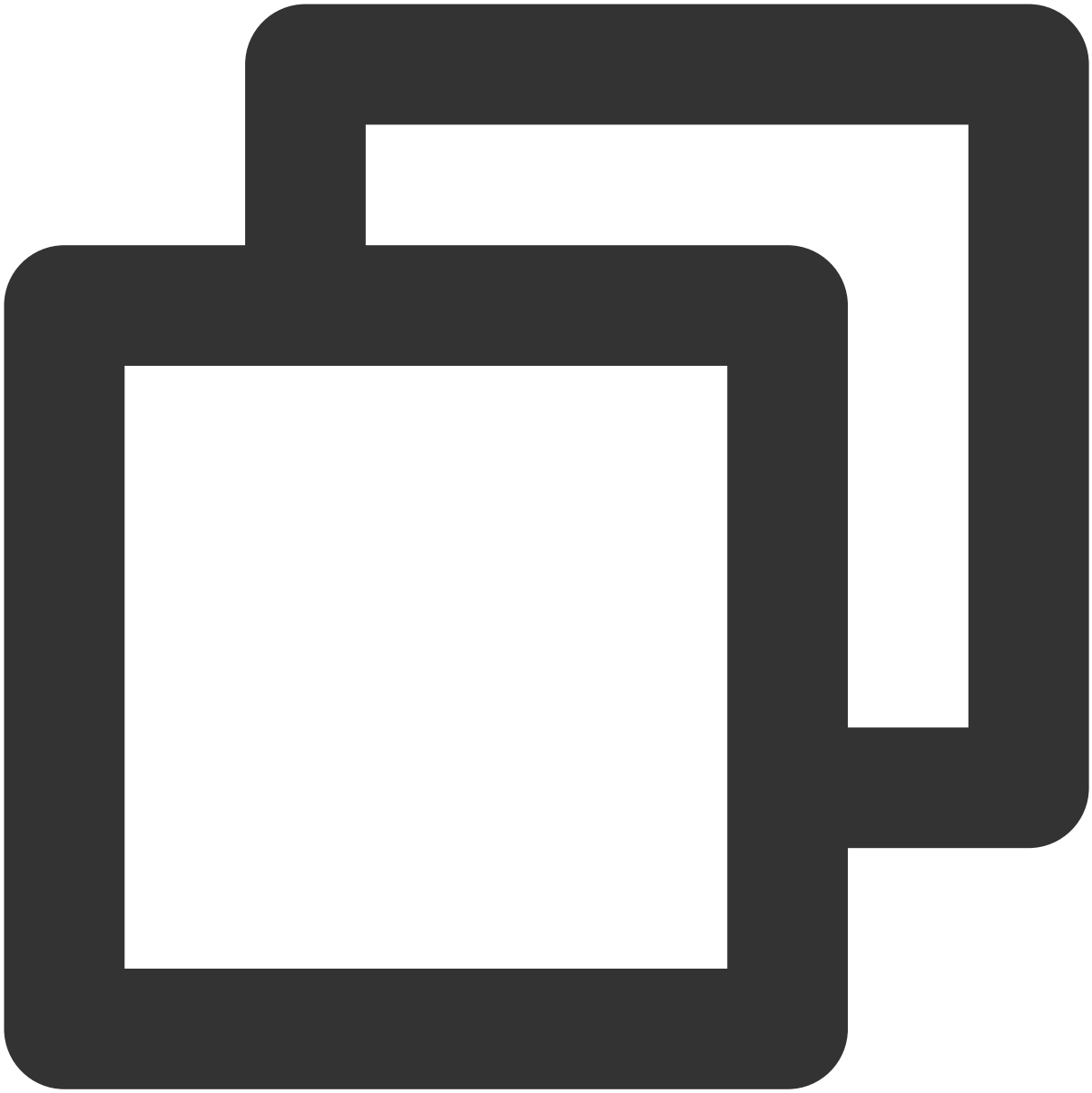
错误码	错误描述
AuthFailure.SignatureExpire	签名过期。Timestamp 与服务器接收到请求的时间相差不得超过五分钟。
AuthFailure.SecretIdNotFound	密钥不存在。请到控制台查看密钥是否被禁用，是否少复制了字符或者多了字符。
AuthFailure.SignatureFailure	签名错误。可能是签名计算错误，或者签名与实际发送的内容不相符合，也有可能是密钥 SecretKey 错误导致的。

AuthFailure.TokenFailure	临时证书 Token 错误。
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。

返回结果

正确返回结果

以云服务器的接口查看实例状态列表（DescribeInstancesStatus）2017-03-12版本为例，若调用成功，其可能的返回如下为：



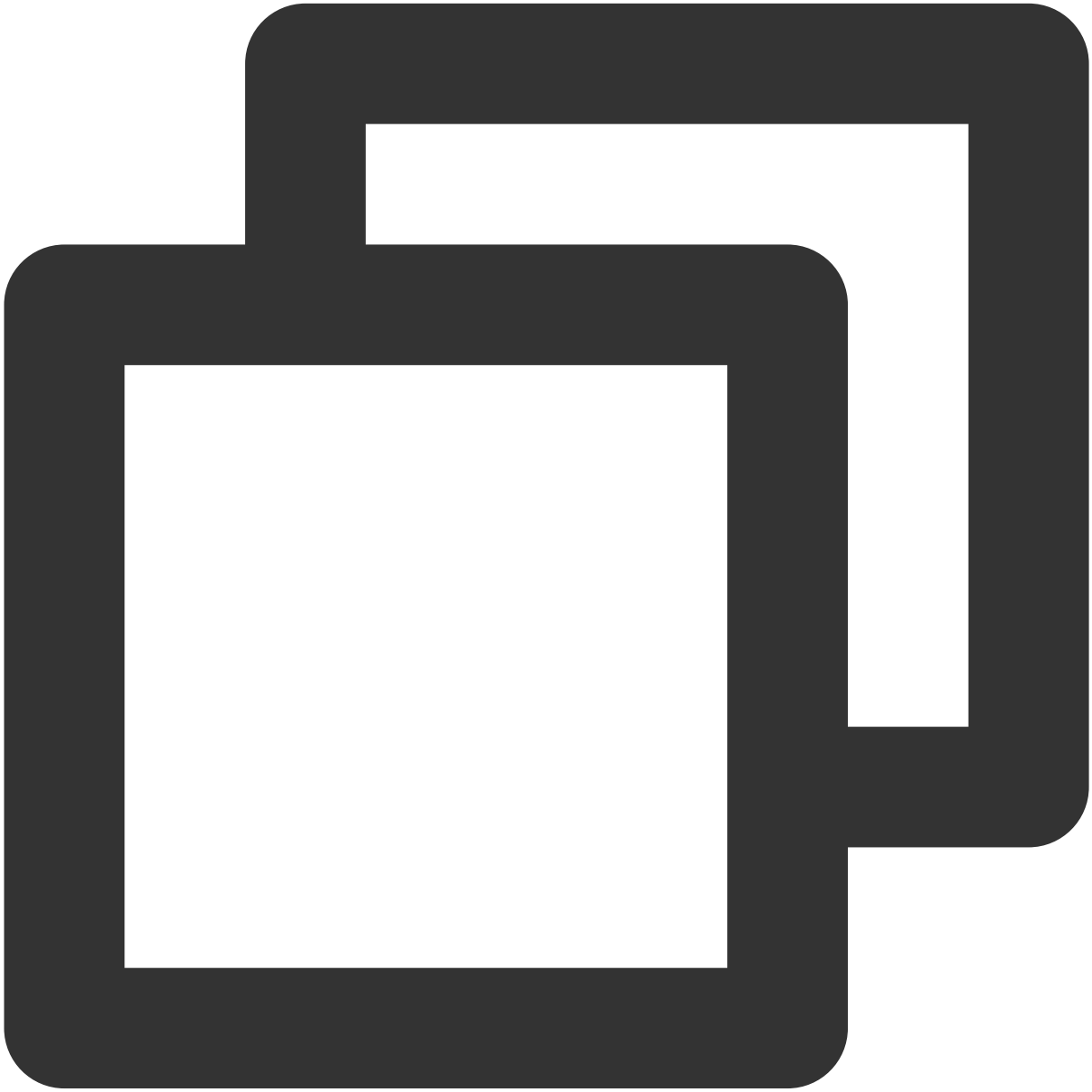
```
{
    "Response": {
        "TotalCount": 0,
        "InstanceStatusSet": [],
        "RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
    }
}
```

`Response` 及其内部的 `RequestId` 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 `TotalCount` 和 `InstanceStatusSet` 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 `TotalCount` 在此情况下的返回值为 0， `InstanceStatusSet` 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：



```
{
    "Response": {
        "Error": {
```

```
            "Code": "AuthFailure.SignatureFailure",
            "Message": "The provided credentials could not be validated. Please che
        },
        "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
    }
}
```

`Error` 的出现代表着该请求调用失败。 `Error` 字段连同其内部的 `Code` 和 `Message` 字段在调用失败时是必定返回的。

`Code` 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。

`Message` 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。

`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。 Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。

LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	HTTPS 请求方法错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

版权所有：腾讯云计算（北京）有限责任公司

GO API

最近更新时间：2023-03-07 18:16:40

腾讯云 API 全新升级3.0，该版本进行了性能优化且全地域部署、支持就近和按地域接入、访问时延下降显著，接口描述更加详细、错误码描述更加全面、SDK增加接口级注释，让您更加方便快捷的使用腾讯云产品。这里针对 GO API 调用方式进行简单说明。
现已支持云服务器（CVM）、云硬盘（CBS）、私有网络（VPC）、云数据库（TencentDB）等 腾讯云产品，后续会支持其他的云产品接入，敬请期待。

了解请求结构

1. 服务地址（endpoint）

API 支持就近地域接入（例如：cvm 产品域名为 `cvm.tencentcloudapi.com`），也支持指定地域域名访问（例如：广州地域的域名为 `cvm.ap-guangzhou.tencentcloudapi.com`），各地域参数见下文公共参数中的地域列表，详情请参见各产品的“请求结构”文档说明判断是否支持该地域。
注意：
对时延敏感的业务，建议指定带地域的域名。

2. 通信协议

腾讯云 API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法：
POST（推荐）
GET
POST 请求支持的 Content-Type 类型：
application/json（推荐），必须使用签名方法 v3（TC3-HMAC-SHA256）。
application/x-www-form-urlencoded，必须使用签名方法 v1（HmacSHA1 或 HmacSHA256）。
multipart/form-data（仅部分接口支持），必须使用签名方法 v3（TC3-HMAC-SHA256）。
GET 请求的请求包大小不得超过32KB。POST 请求使用签名方法 v1（HmacSHA1、HmacSHA256）时不得超过1MB。POST 请求使用签名方法 v3（TC3-HMAC-SHA256）时支持10MB。

4. 字符编码

均使用 UTF-8 编码。

版权所有：腾讯云计算（北京）有限责任公司

公共参数

说明：
公共参数是用于标识用户和接口签名的参数，每次请求均需要携带这些参数，才能正常发起请求。

签名方法 V3 公共参数

签名方法 v3 （有时也称作 TC3-HMAC-SHA256）相比签名方法 v1 （有些文档可能会简称签名方法），更安全，支持更大的请求包，支持 POST JSON 格式，性能有一定提升，推荐使用该签名方法计算签名。使用方法见下文 “签名方法介绍” 。

参数名称	类型	必选	描述
X-TC-Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
X-TC-Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如 1529223702。**注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。**
X-TC-Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
Authorization	String	是	HTTP 标准身份认证头部字段，例如： TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=72e494ea8**********a96525168 其中： TC3-HMAC-SHA256：签名方法，目前固定取该值。 Credential：签名凭证，AKIDEXAMPLE 是 SecretId。 Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致。 service 为产品名，通常为域名前缀，例如域名 cvm.tencentcloudapi.com 意味着产品名是 cvm。本产品取值为 cvm。 SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部。 Signature：签名摘要，计算过程详见下文。
X-TC-Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

签名方法 V1 公共参数

使用签名方法 v1（有时会称作 HmacSHA256 和 HmacSHA1），公共参数需要统一放到请求串中。

参数名称	类型	必选	描述
Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances 。
Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如 1529223702，如果与当前时间相差过大，会引起签名过期错误。
Nonce	Integer	是	随机正整数，与 Timestamp 联合起来，用于防止重放攻击。
SecretId	String	是	在 云API密钥 上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey，而 SecretKey 会用来生成请求签名 Signature。
Signature	String	是	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。具体计算方法参见下文“签名方法介绍” 。
Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
SignatureMethod	String	否	签名方式，目前支持 HmacSHA256 和 HmacSHA1。只有指定此参数为 HmacSHA256 时，才使用 HmacSHA256 算法验证签名，其他情况均使用 HmacSHA1 验证签名。
Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token 。

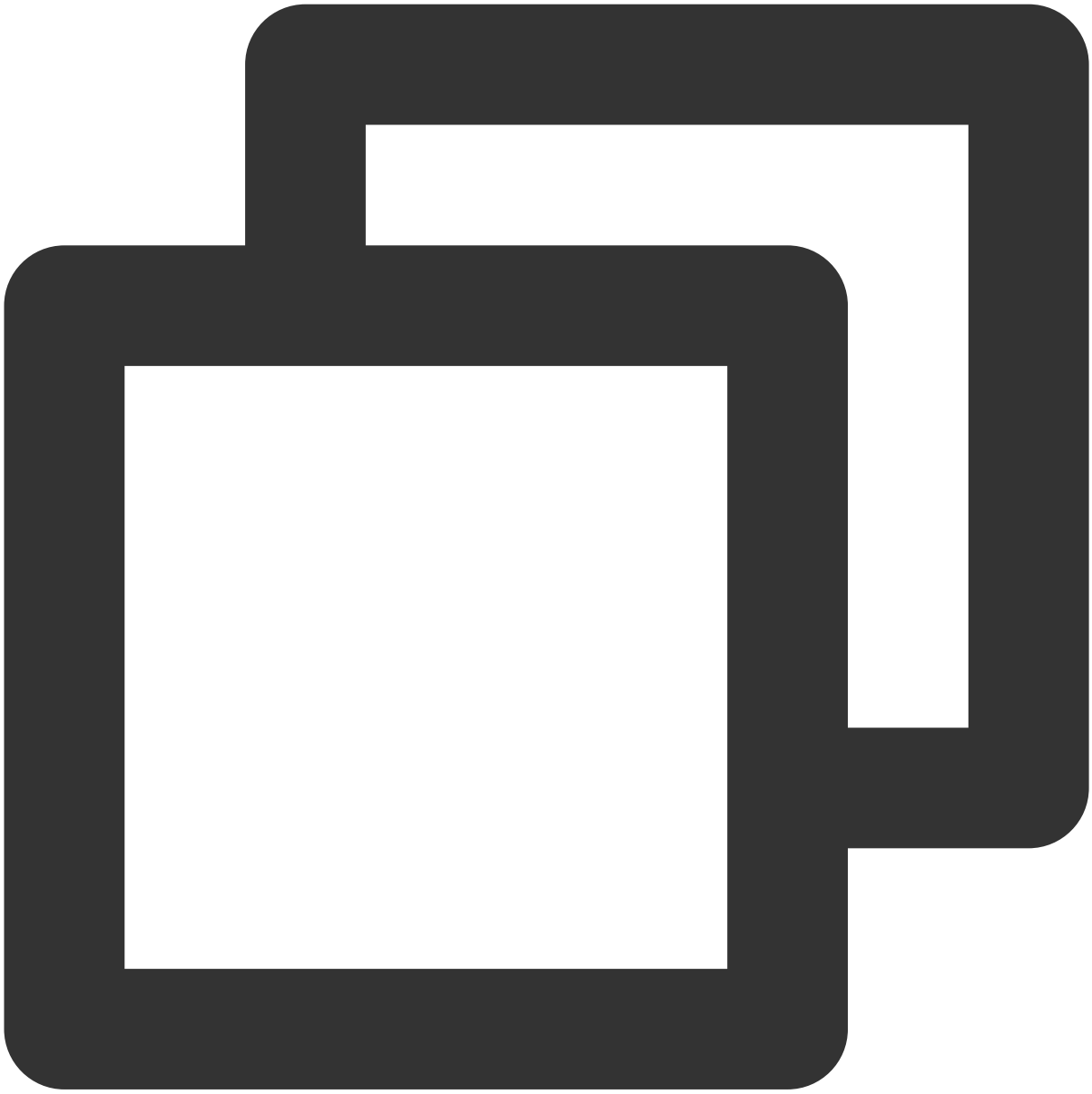
地域列表

由于各个产品支持地域不同，具体详情请参考各产品文档中的地域列表。
例如，您可以参考云服务器的 地域列表。

GO API 调用方式

腾讯云 API 会对每个请求进行身份验证，用户需要使用安全凭证，经过特定的步骤对请求进行签名（Signature），每个请求都需要在公共请求参数中指定该签名结果并以指定的方式和格式发送请求。

假设用户的 SecretId 和 SecretKey 分别是： `AKIDz8krbsJ5*********mLPx3EXAMPLE` 和 `Gu5t9xGAR**********EXAMPLE` 。用户想查看广州区云服务器名为“未命名”的主机状态，只返回一条数据。则请求可能为：



```
curl -X POST https://cvm.tencentcloudapi.com \\
-H "Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5*********mLPx3EXAMPLE/20
-H "Content-Type: application/json; charset=utf-8" \\
-H "Host: cvm.tencentcloudapi.com" \\
-H "X-TC-Action: DescribeInstances" \\
-H "X-TC-Timestamp: 1551113065" \\
-H "X-TC-Version: 2017-03-12" \\
```

```
-H "X-TC-Region: ap-guangzhou" \\
-d '{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instanc
```

步骤1：申请安全凭证

本文使用的安全凭证为密钥，密钥包括 SecretId 和 SecretKey。每个用户最多可以拥有两对密钥。

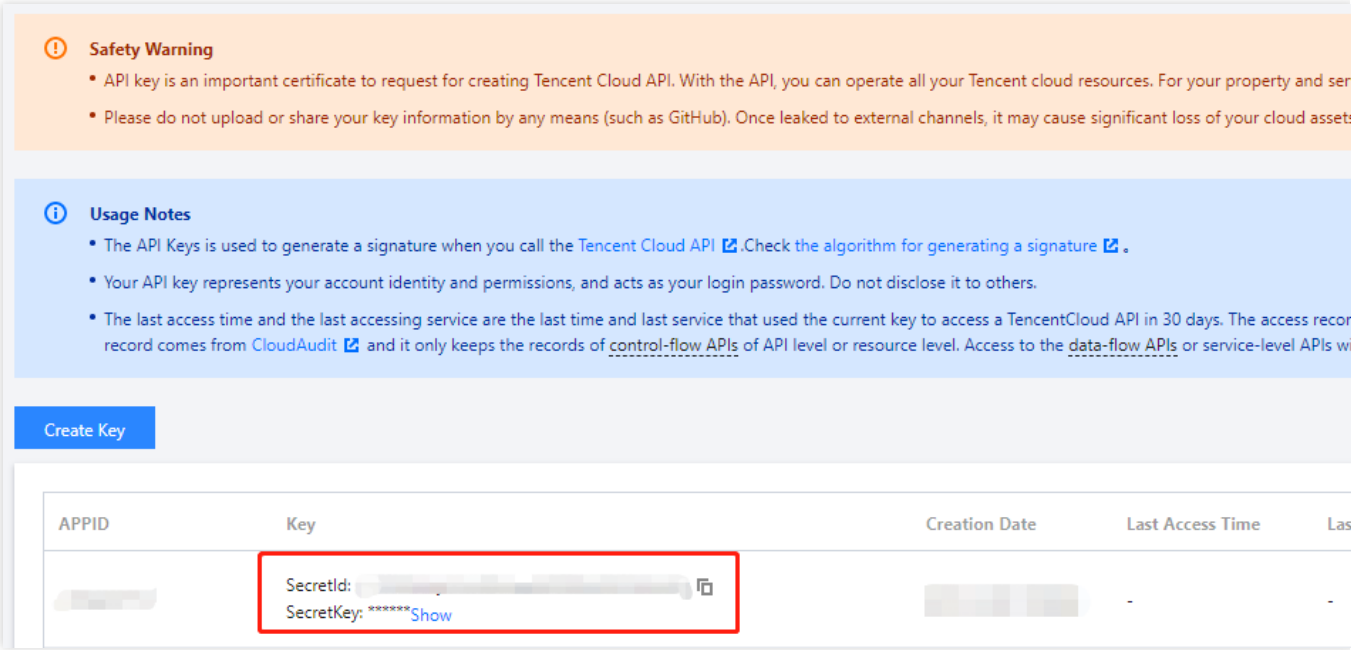
SecretId：用于标识 API 调用者身份，可以简单类比为用户名。

SecretKey：用于验证 API 调用者的身份，可以简单类比为密码。

注意：

用户必须严格保管安全凭证，避免泄露，否则将危及财产安全。如已泄漏，请立刻禁用该安全凭证。

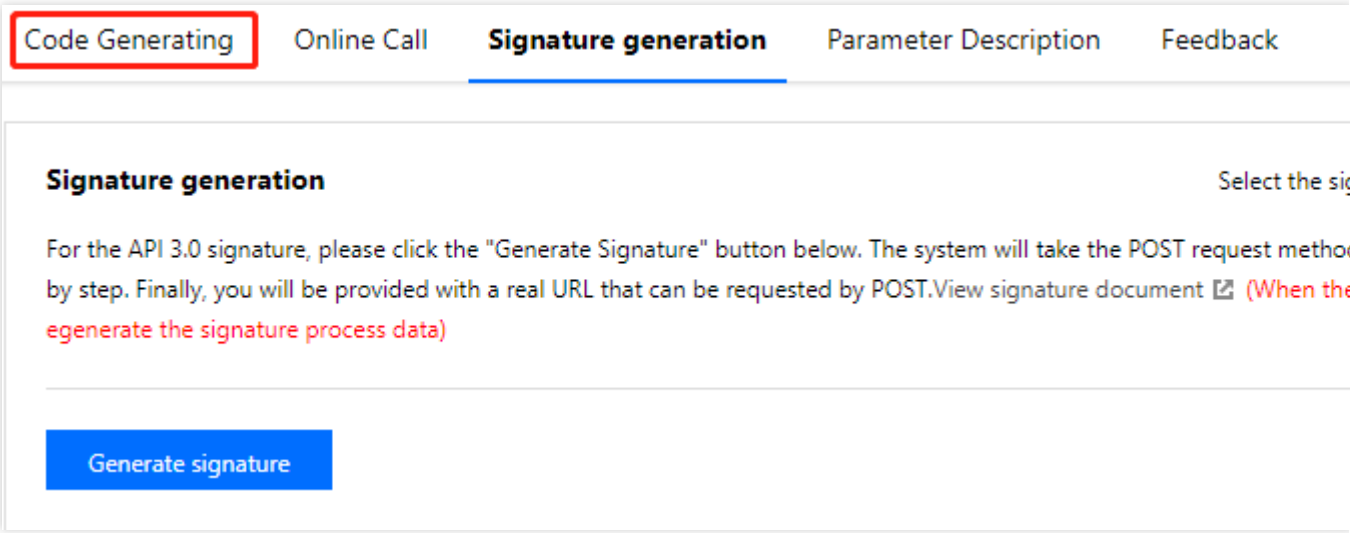
前往 API密钥管理 页面，即可进行获取。如下图所示：



步骤2：

1. 获取 API 3.0 V3 版本签名

签名方法 v3 （TC3-HMAC-SHA256）功能上覆盖了以前的签名方法 v1，而且更安全，支持更大的请求，支持 json 格式，性能有一定提升，推荐使用该签名方法计算签名。



注意：

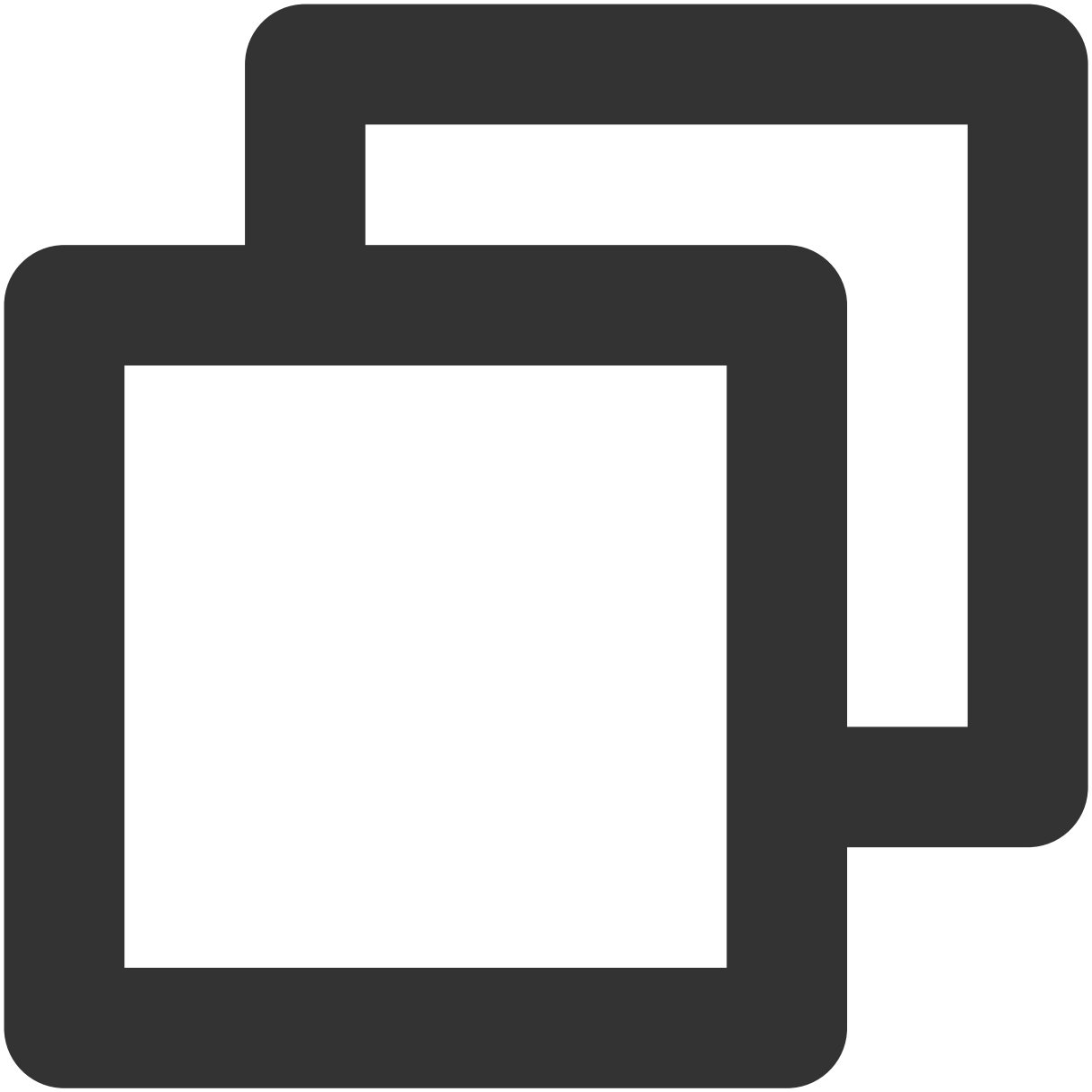
首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“ API 3.0 签名 v3 ”，可以生成签名过程进行验证，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的7种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

云 API 支持 GET 和 POST 请求。对于 GET 方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。对于POST方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式绝大多数接口均支持，multipart 格式只有特定接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。推荐使用 POST 请求，因为两者的结果并无差异，但 GET 请求只支持 32 KB 以内的请求包。

下面以 查看实例列表 接口为例，分步骤介绍签名的计算过程。我们选择该接口是因为：

1. 云服务器默认已开通，该接口很常用；
2. 该接口是只读的，不会改变现有资源的状态；
3. 接口覆盖的参数种类较全，可以演示包含数据结构的数组如何使用。

1. 拼接规范请求串

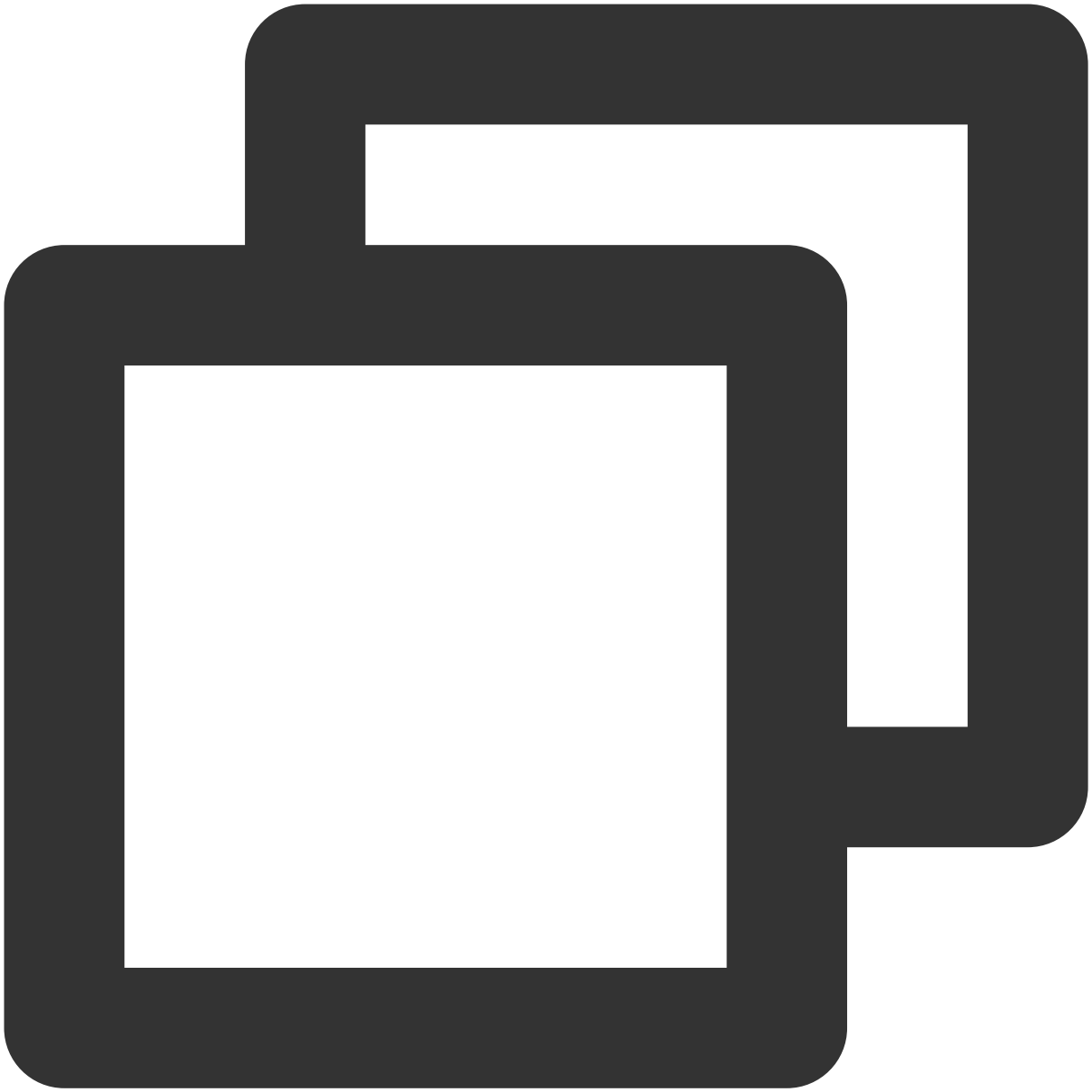


```
CanonicalRequest =
    HTTPRequestMethod + '\\n' +
    CanonicalURI + '\\n' +
    CanonicalQueryString + '\\n' +
    CanonicalHeaders + '\\n' +
    SignedHeaders + '\\n' +
    HashedRequestPayload
```

字段名称	解释

HTTPRequestMethod	HTTP 请求方法（GET、POST ）。此示例取值为 `POST` 。
CanonicalURI	URI 参数，API 3.0 固定为正斜杠（/）。
CanonicalQueryString	发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串""，对于 G 求，则为 URL 中问号（?）后面的字符串内容，例如：Limit=10&Offset=0。 注意：CanonicalQueryString 需要参考 RFC3986 进行 URLEncode，字符集 UTF8，推荐使用 语言标准库，所有特殊字符均需编码，大写形式。
CanonicalHeaders	参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部 签名以提高自身请求的唯一性和安全性。 拼接规则：头部 key 和 value 统一转成小写，掉首尾空格，按照 key:value\\n 格式拼接；多个头部，按照头部 key（小写）的 ASCII 行拼接。此示例计算结果是 `content-type:application/json; charset=ut 8\\nhost:cvm.tencentcloudapi.com\\n` 。 注意：content-type 必须和实际发 符合，有些编程语言网络库即使未指定也会自动添加 charset 值，如果签名时和发送时 致，服务器会返回签名校验失败。
SignedHeaders	参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包 部内容是一一对应的。content-type 和 host 为必选头部。 拼接规则：头部 key 统一转 写；多个头部 key（小写）按照 ASCII 升序进行拼接，并且以分号（;）分隔。此示例 content-type;host
HashedRequestPayload	请求正文（Requestpayload，即 body，此示例为 `{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-name"` 的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(RequestPayload))), HTTP 请求正文做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。 GET 请求，RequestPayload 固定为空字符串。此示例计算结果是 `35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f`

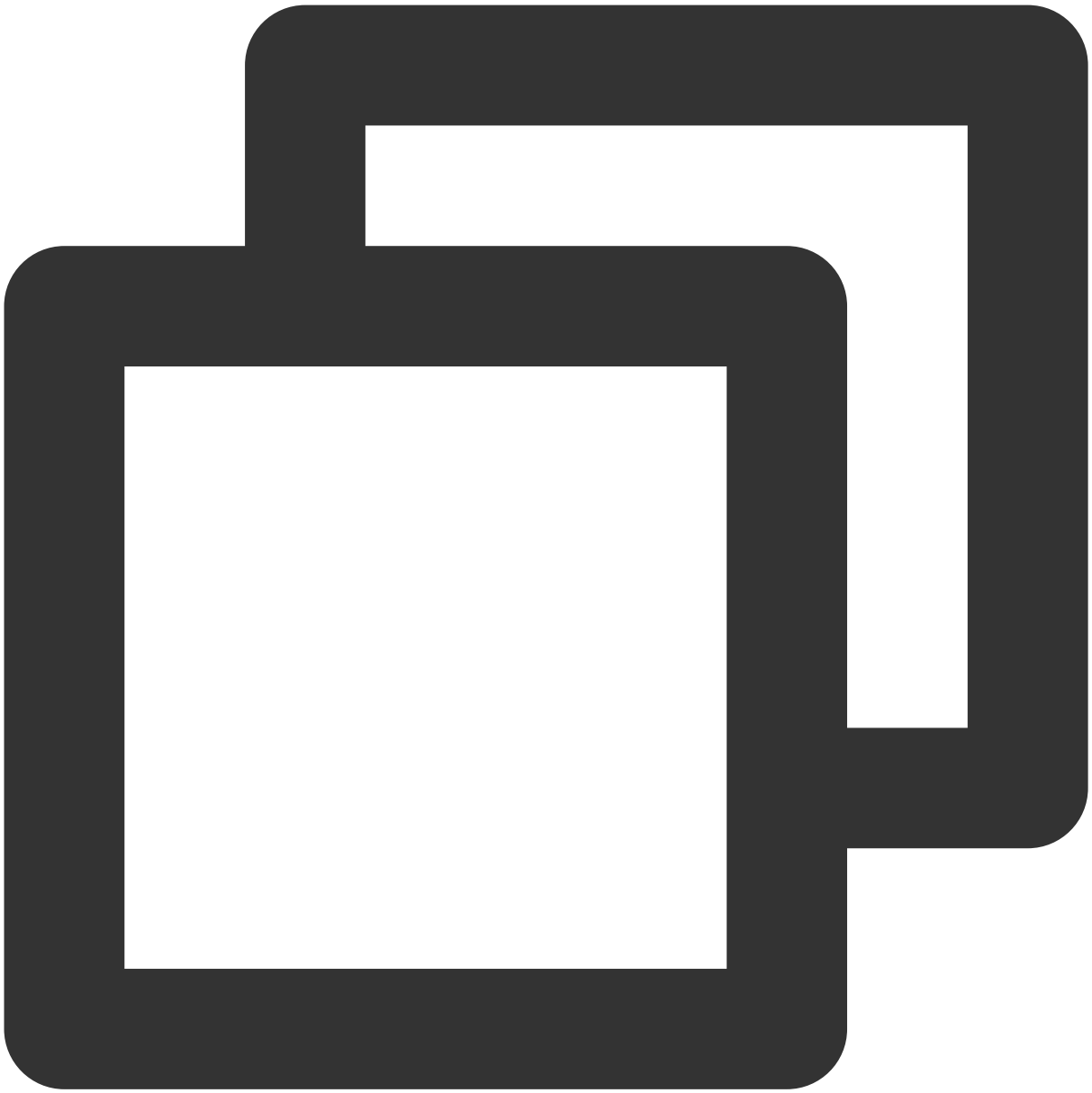
根据以上规则，示例中得到的规范请求串如下：



```
POST
/

content-type:application/json; charset=utf-8
host:cvm.tencentcloudapi.com

content-type;host
35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f064
```

2. 拼接待签名字符串

按如下格式拼接待签名字符串：



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

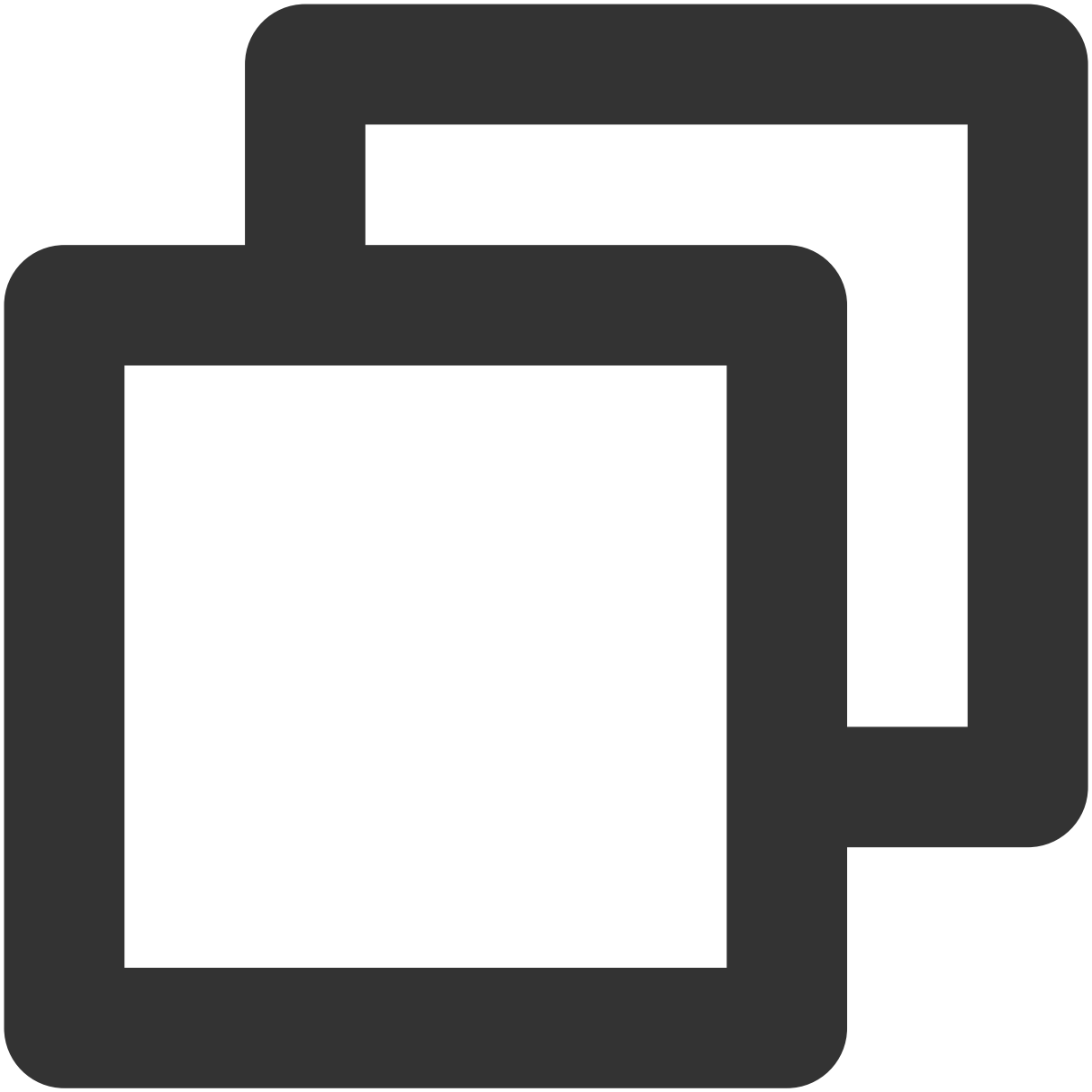
字段名称	解释
Algorithm	签名算法，目前固定为 `TC3-HMAC-SHA256` 。

RequestTimestamp	请求时间戳，即请求头部的公共参数 X-TC-Timestamp 取值，取当前时间 UNIX 时间确到秒。此示例取值为 `1551113065` 。
CredentialScope	凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符串（tc3_request）。**Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Time 换算的 UTC 标准时间日期一致**；service 为产品名，必须与调用的产品域名一致。此算结果是 2019-02-25/cvm/tc3_request。
HashedCanonicalRequest	前述步骤拼接所得规范请求串的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))。此示例计算结果是 `5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d`

注意：

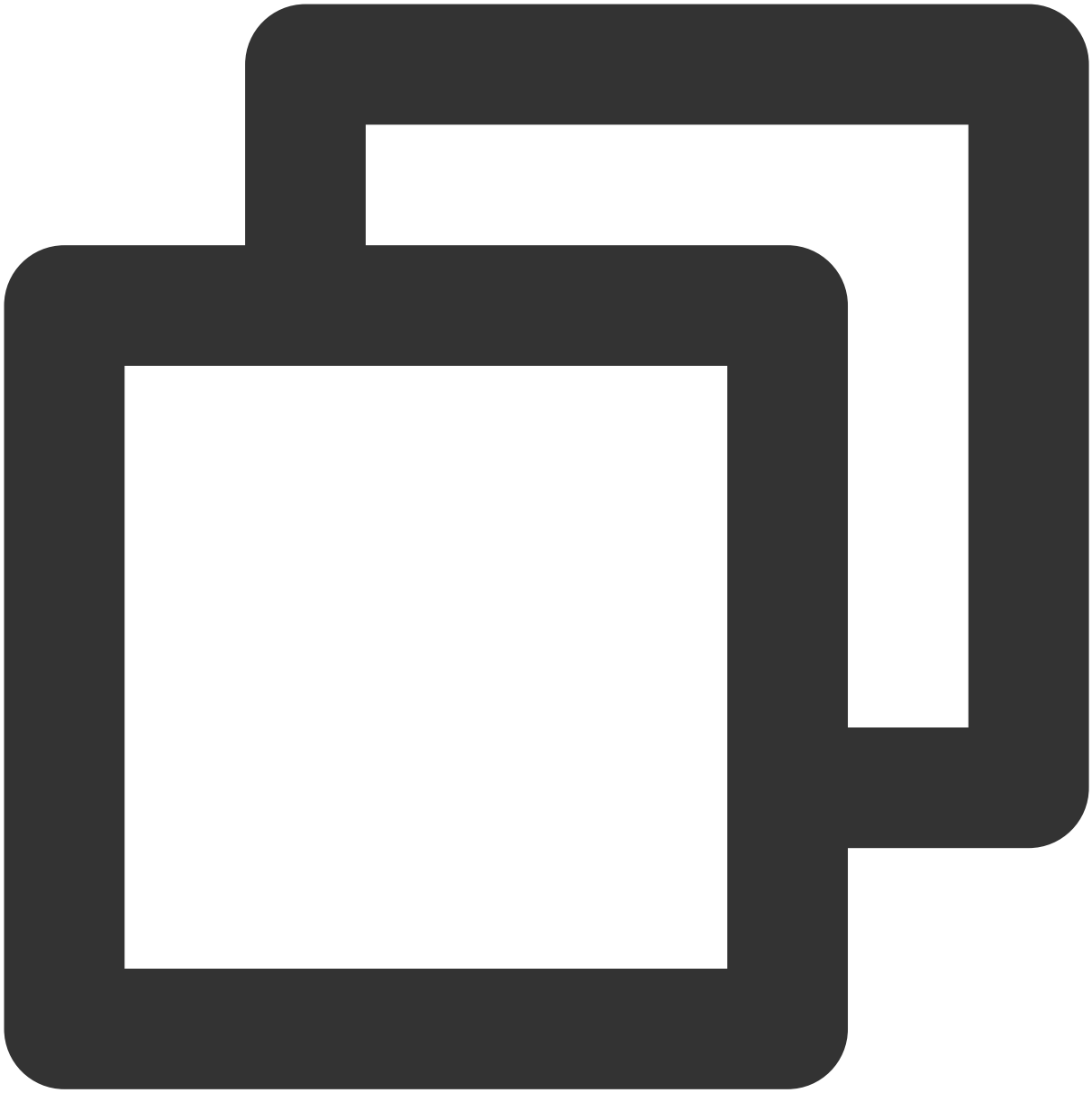
1. Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为 1551113065，在东八区的时间是 2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为 2019-02-25，而不是 2019-02-26。
2. Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败，返回签名过期错误。

根据以上规则，示例中得到的待签名字符串如下：



```
TC3-HMAC-SHA256
1551113065
2019-02-25/cvm/tc3_request
5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d7031
```

3.计算签名(伪代码)



```
secretDate := hmacsha256(date, "TC3"+secretKey)
secretService := hmacsha256(service, secretDate)
secretSigning := hmacsha256("tc3_request", secretService)
signature := hex.EncodeToString([]byte(hmacsha256(string2sign, secretSigning)))
fmt.Println(signature)
```

派生出的密钥 `SecretDate` 、 `SecretService` 和 `SecretSigning` 是二进制的数据，可能包含不可打印字符，此处不展示中间结果。

字段名称	解释

secretKey	原始的 secretKey，即 `Gu5t9xGAR**********EXAMPLE` 。
date	即 Credential 中的 Date 字段信息(标准时间)。此示例取值为 `2019-02-25` 。
service	即 Credential 中的 Service 字段信息。此示例取值为 `cvm` 。

此示例计算结果是 `72e494ea8****************************************a96525168` 。

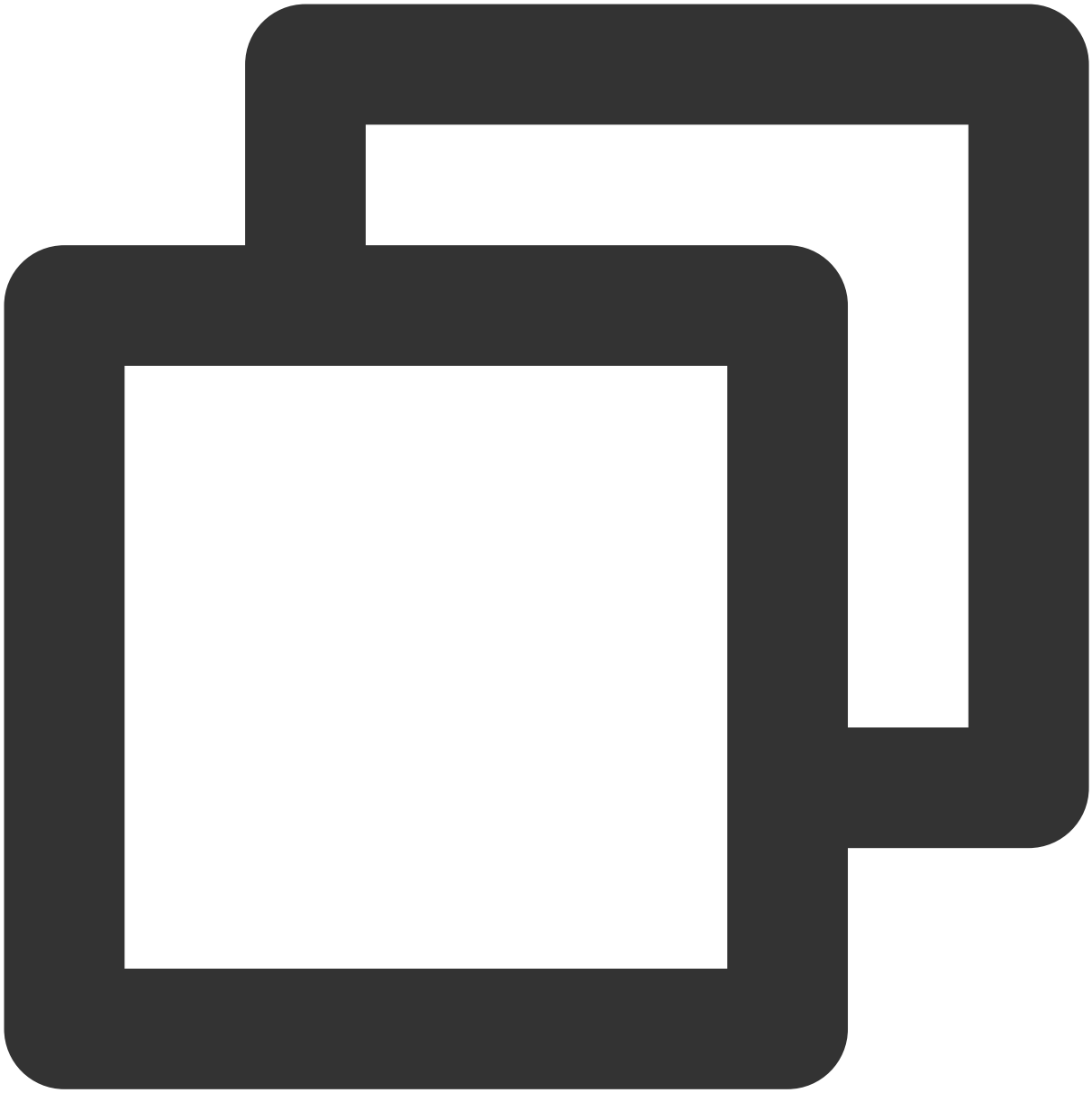
4. 拼接 Authorization

按如下格式拼接 Authorization：

```
    authorization := fmt.Sprintf("%s Credential=%s/%s, SignedHeaders=%s, Signature=
        algorithm,
        secretId,
        credentialScope,
        signedHeaders,
        signature)
    fmt.Println(authorization)
```

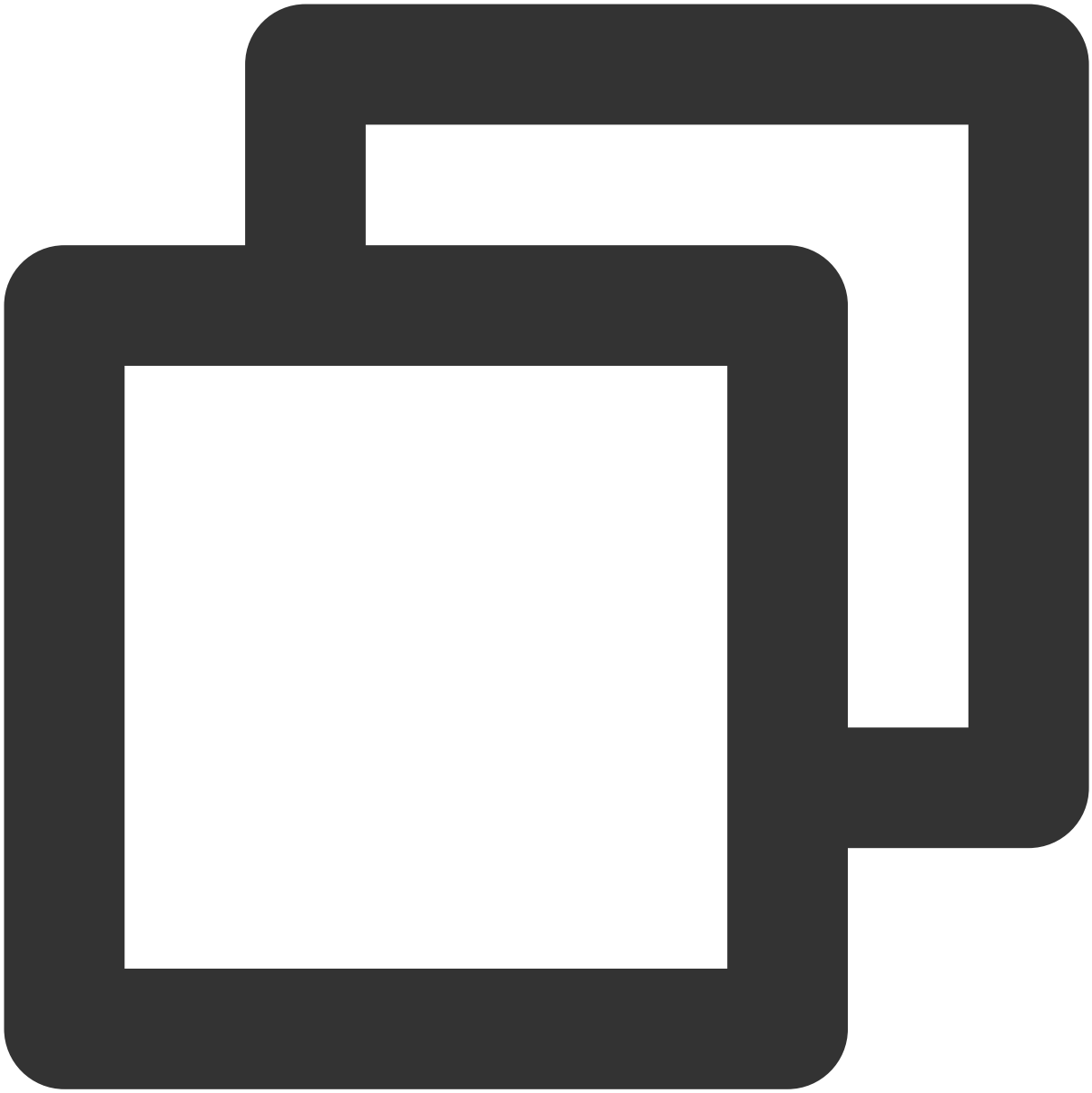
字段名称	解释
algorithm	签名方法，固定为 `TC3-HMAC-SHA256` 。
secretId	密钥对中的 SecretId，即 `AKIDz8krbsJ5**********mLPx3EXAMPLE` 。
credentialScope	见上文，凭证范围。此示例计算结果是 `2019-02-25/cvm/tc3_request` 。
signedHeaders	见上文，参与签名的头部信息。此示例取值为 `content-type;host` 。
signature	签名值。此示例计算结果是 `72e494ea8****************************************a96525168` 。

根据以上规则，示例中得到的值为：



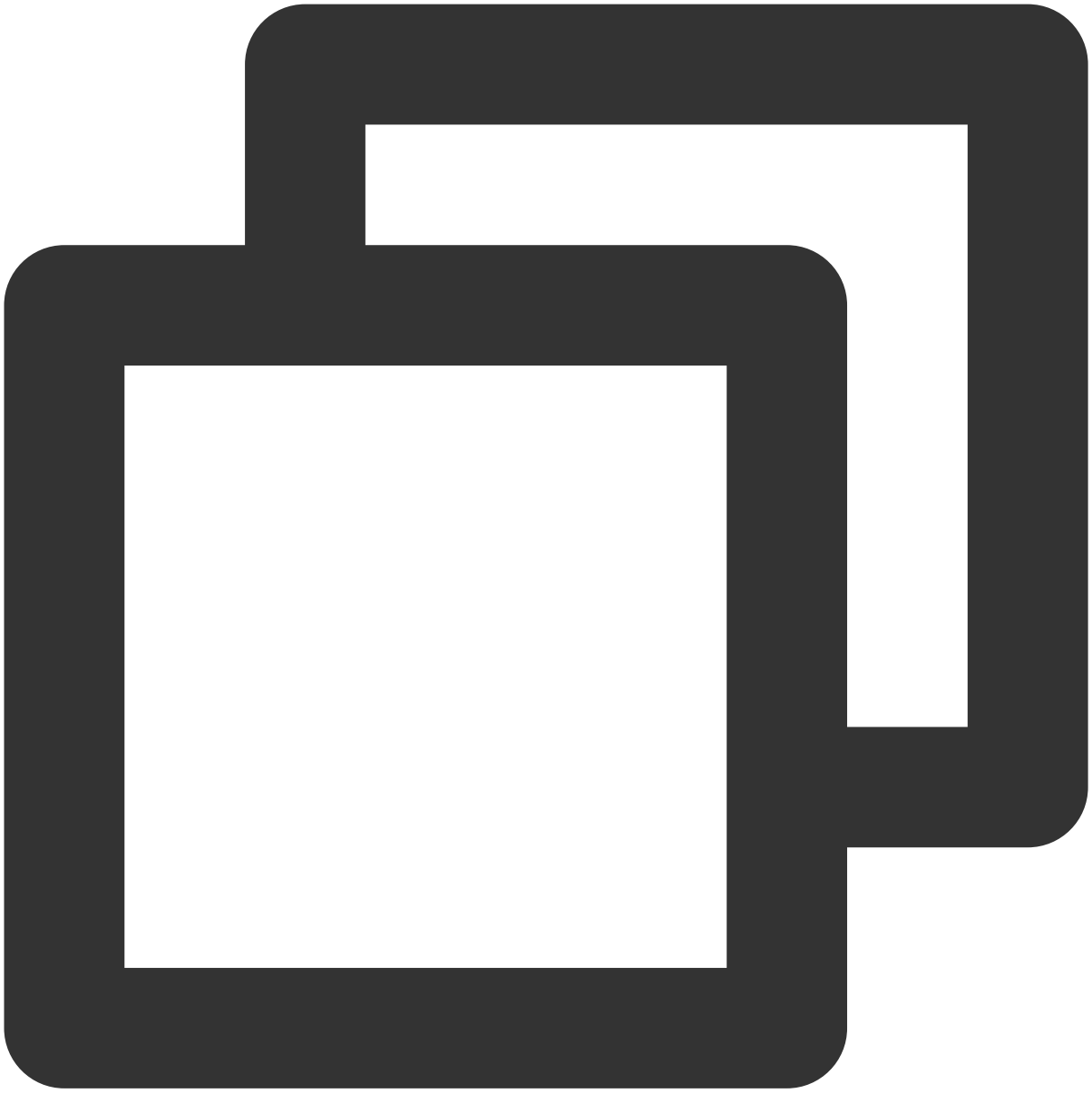
```
TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/2019-02-25/cvm/tc3_re
```

最终完整的调用信息如下：



```
POST https://cvm.tencentcloudapi.com/
Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/2019-0
Content-Type: application/json; charset=utf-8
Host: cvm.tencentcloudapi.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1551113065
X-TC-Region: ap-guangzhou

{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-na
```

5. API 3.0 签名 V3示例



```
package main

import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "fmt"
    "time"
)
```

```
func sha256hex(s string) string {
    b := sha256.Sum256([]byte(s))
    return hex.EncodeToString(b[:])
}

func hmacsha256(s, key string) string {
    hashed := hmac.New(sha256.New, []byte(key))
    hashed.Write([]byte(s))
    return string(hashed.Sum(nil))
}

func main() {
    secretId := "AKIDz8krbsJ5**********mLPx3EXAMPLE"
    secretKey := "Gu5t9xGAR***********EXAMPLE"
    host := "cvm.tencentcloudapi.com"
    algorithm := "TC3-HMAC-SHA256"
    service := "cvm"
    version := "2017-03-12"
    action := "DescribeInstances"
    region := "ap-guangzhou"
    //var timestamp int64 = time.Now().Unix()
    var timestamp int64 = 1551113065

    // step 1: build canonical request string
    httpRequestMethod := "POST"
    canonicalURI := "/"
    canonicalQueryString := ""
    canonicalHeaders := "content-type:application/json; charset=utf-8\\n" + "host:"
    signedHeaders := "content-type;host"
    payload := `{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Nam
    hashedRequestPayload := sha256hex(payload)
    canonicalRequest := fmt.Sprintf("%s\\n%s\\n%s\\n%s\\n%s\\n%s",
        httpRequestMethod,
        canonicalURI,
        canonicalQueryString,
        canonicalHeaders,
        signedHeaders,
        hashedRequestPayload)
    fmt.Println(canonicalRequest)

    // step 2: build string to sign
    date := time.Unix(timestamp, 0).UTC().Format("2006-01-02")
    credentialScope := fmt.Sprintf("%s/%s/tc3_request", date, service)
    hashedCanonicalRequest := sha256hex(canonicalRequest)
    string2sign := fmt.Sprintf("%s\\n%d\\n%s\\n%s",
        algorithm,
```

```
        timestamp,
        credentialScope,
        hashedCanonicalRequest)
    fmt.Println(string2sign)

    // step 3: sign string
    secretDate := hmacsha256(date, "TC3"+secretKey)
    secretService := hmacsha256(service, secretDate)
    secretSigning := hmacsha256("tc3_request", secretService)
    signature := hex.EncodeToString([]byte(hmacsha256(string2sign, secretSigning)))
    fmt.Println(signature)

    // step 4: build authorization
    authorization := fmt.Sprintf("%s Credential=%s/%s, SignedHeaders=%s, Signature=
        algorithm,
        secretId,
        credentialScope,
        signedHeaders,
        signature)
    fmt.Println(authorization)

    curl := fmt.Sprintf(`curl -X POST https://%s\\
 -H "Authorization: %s"\\
 -H "Content-Type: application/json; charset=utf-8"\\
 -H "Host: %s" -H "X-TC-Action: %s"\\
 -H "X-TC-Timestamp: %d"\\
 -H "X-TC-Version: %s"\\
 -H "X-TC-Region: %s"\\
 -d '%s'`, host, authorization, host, action, timestamp, version, region, payload)
    fmt.Println(curl)
}
```

2. 获取 API 3.0 V1 版本签名

签名方法 v1 简单易用，但是功能和安全性都不如签名方法 v3，推荐使用签名方法 v3。

注意：

首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“ API 3.0 签名 v1 ”，可以生成签名过程进行验证，并提供了部分编程语言的签名示例，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的 7 种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

以云服务器查看实例列表（DescribeInstances）请求为例，当用户调用这一接口时，其请求参数可能如下:

参数名称	中文	参数值
Action	方法名	DescribeInstances
SecretId	密钥 ID	`AKIDz8krbsJ5*********mLPx3EXAMPLE`

Timestamp	当前时间戳	1465185768
Nonce	随机正整数	11886
Region	实例所在区域	ap-guangzhou
InstanceIds.0	待查询的实例 ID	ins-09dx96dg
Offset	偏移量	0
Limit	最大允许输出	20
Version	接口版本号	2017-03-12

1. 对参数排序

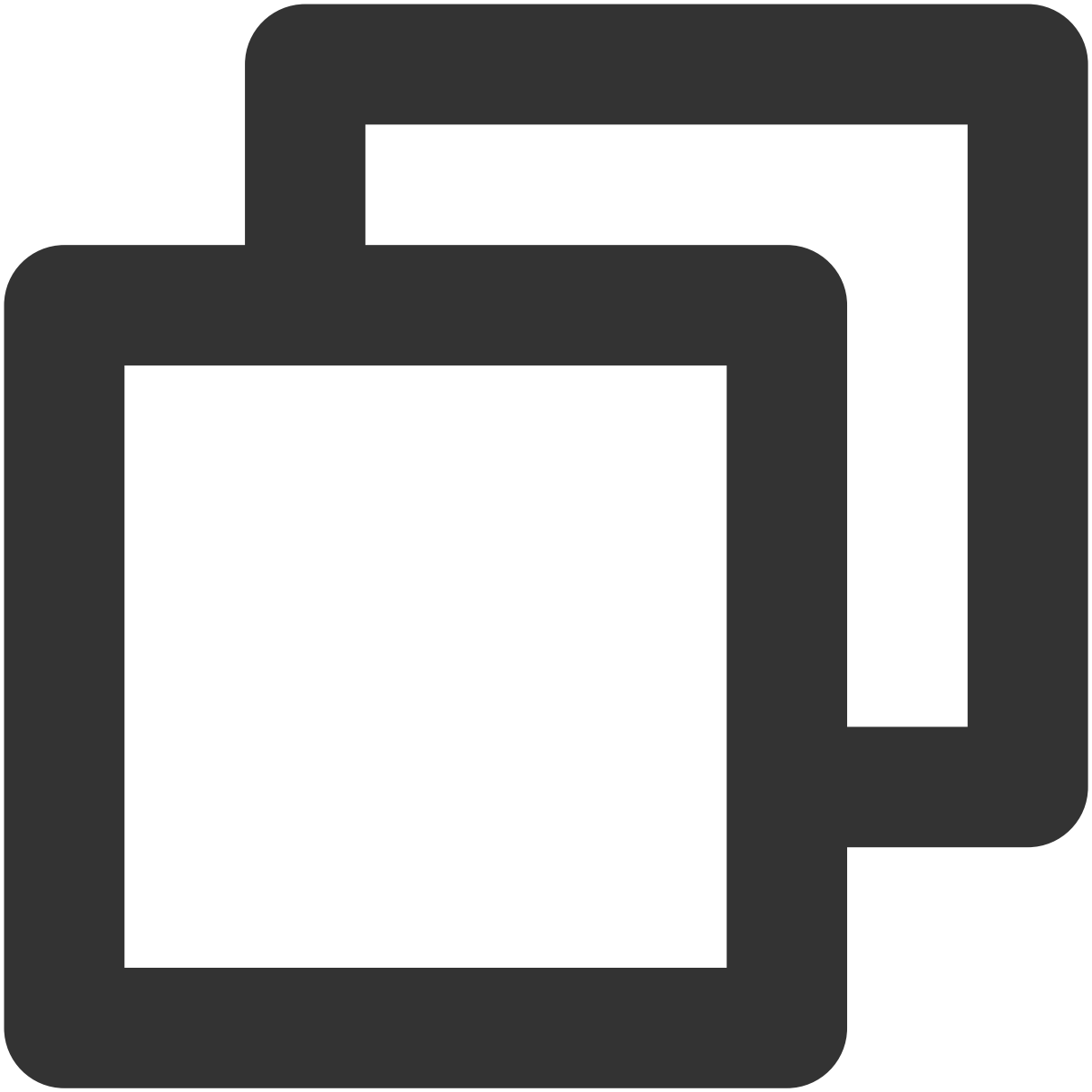
首先对所有请求参数按参数名的字典序（ASCII 码）升序排序。

注意：

只按参数名进行排序，参数值保持对应即可，不参与比大小。

按 ASCII 码比大小，如 InstanceIds.2 要排在 InstanceIds.12 后面，不是按字母表，也不是按数值。用户可以借助编程语言中的相关排序函数来实现这一功能，如 PHP 中的 ksort 函数。

上述示例参数的排序结果如下：



```
{
    'Action' : 'DescribeInstances',
    'InstanceIds.0' : 'ins-09dx96dg',
    'Limit' : 20,
    'Nonce' : 11886,
    'Offset' : 0,
    'Region' : 'ap-guangzhou',
    'SecretId' : 'AKIDz8krbsJ5**********mLPx3EXAMPLE',
    'Timestamp' : 1465185768,
    'Version': '2017-03-12',
}
```

使用程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

2. 拼接规范请求串

此步骤生成请求字符串。将把上一步排序好的请求参数格式化成“参数名称=参数值”的形式，如对 Action 参数，其参数名称为 "Action" ，参数值为 "DescribeInstances" ，因此格式化后就为 Action=DescribeInstances 。

注意：

“参数值”为原始值而非 url 编码后的值。

然后将格式化后的各个参数用"&"拼接在一起，最终生成的请求字符串为:

```
Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&R
```

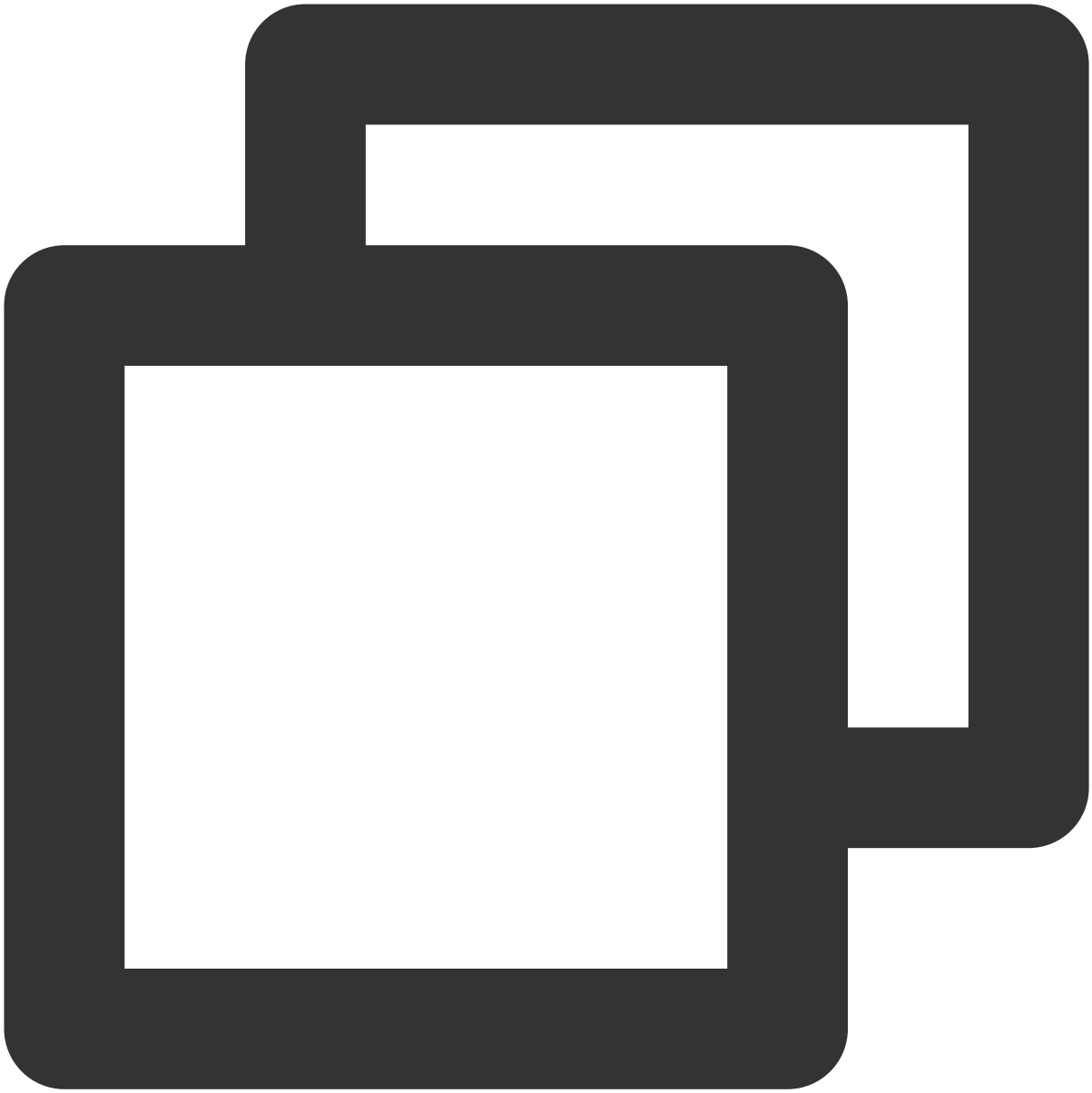
3. 拼接待签名字符串

此步骤生成签名原文字符串。 签名原文字符串由以下几个参数构成:

1. 请求方法: 支持 POST 和 GET 方式，这里使用 GET 请求，注意方法为全大写。
2. 请求主机:查看实例列表（DescribeInstances）的请求域名为： `cvm.tencentcloudapi.com` 。实际的请求域名根据接口所属模块的不同而不同，详见各接口说明。
3. 请求路径: 当前版本云API的请求路径固定为 / 。
4. 请求字符串: 即上一步生成的请求字符串。

签名原文串的拼接规则为： `请求方法 + 请求主机 +请求路径 + ? + 请求字符串` 。

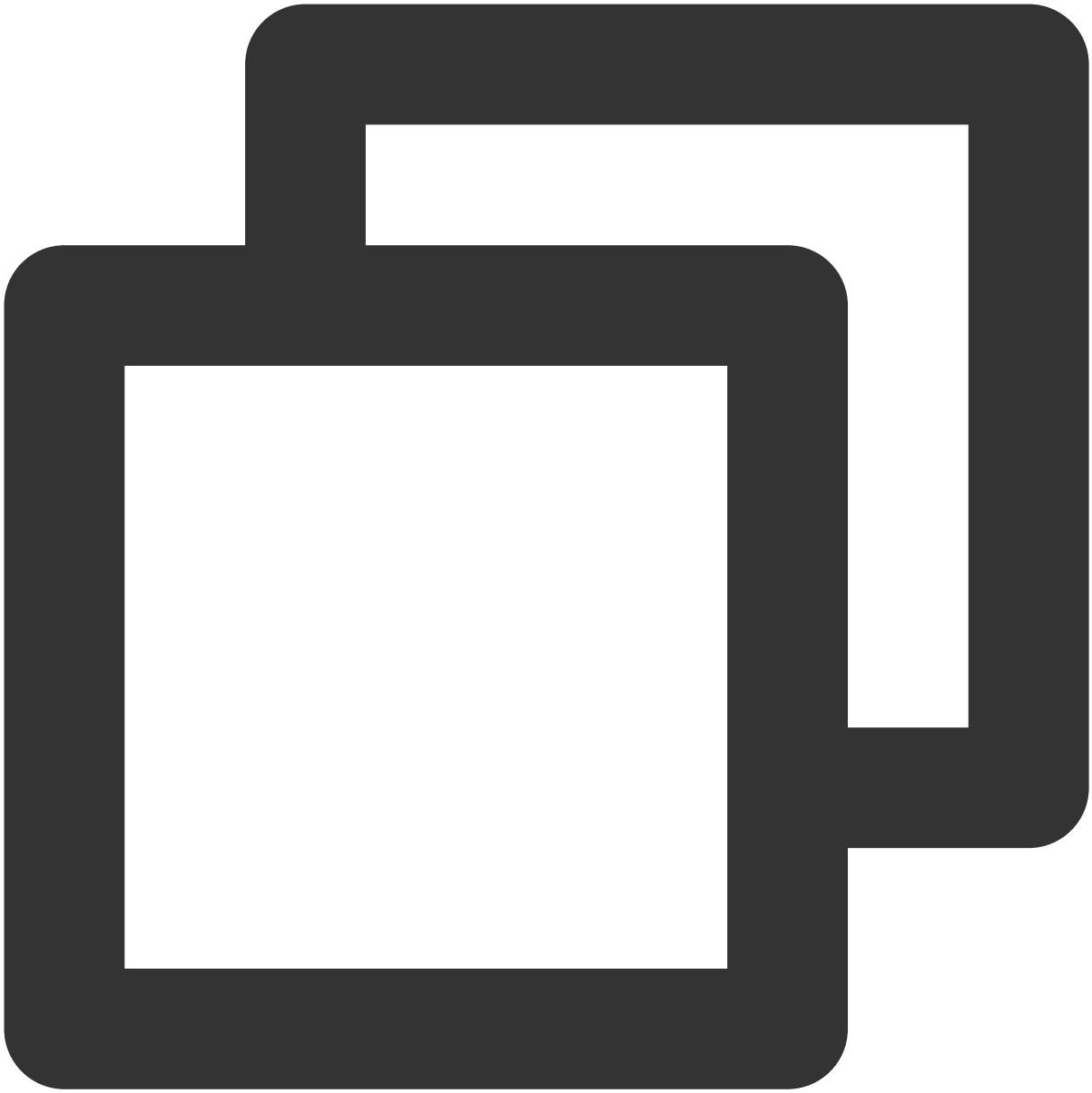
示例的拼接结果为：



```
GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Lim
```

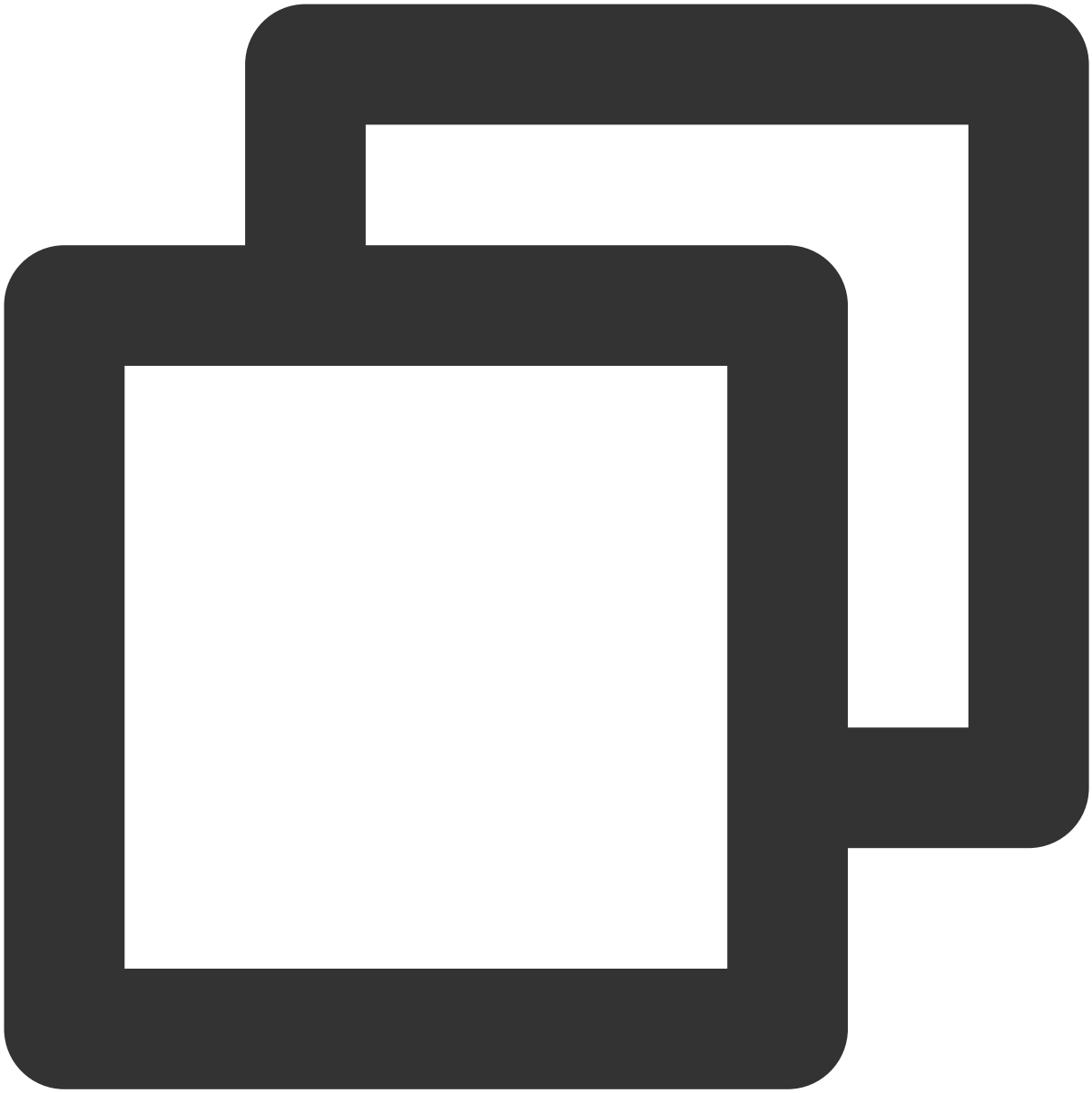
4. 计算签名（伪代码）

此步骤生成签名串。 首先使用 HMAC-SHA1 算法对上一步中获得的**签名原文字符串**进行签名，然后将生成的签名串使用 Base64 进行编码，即可获得签名串。

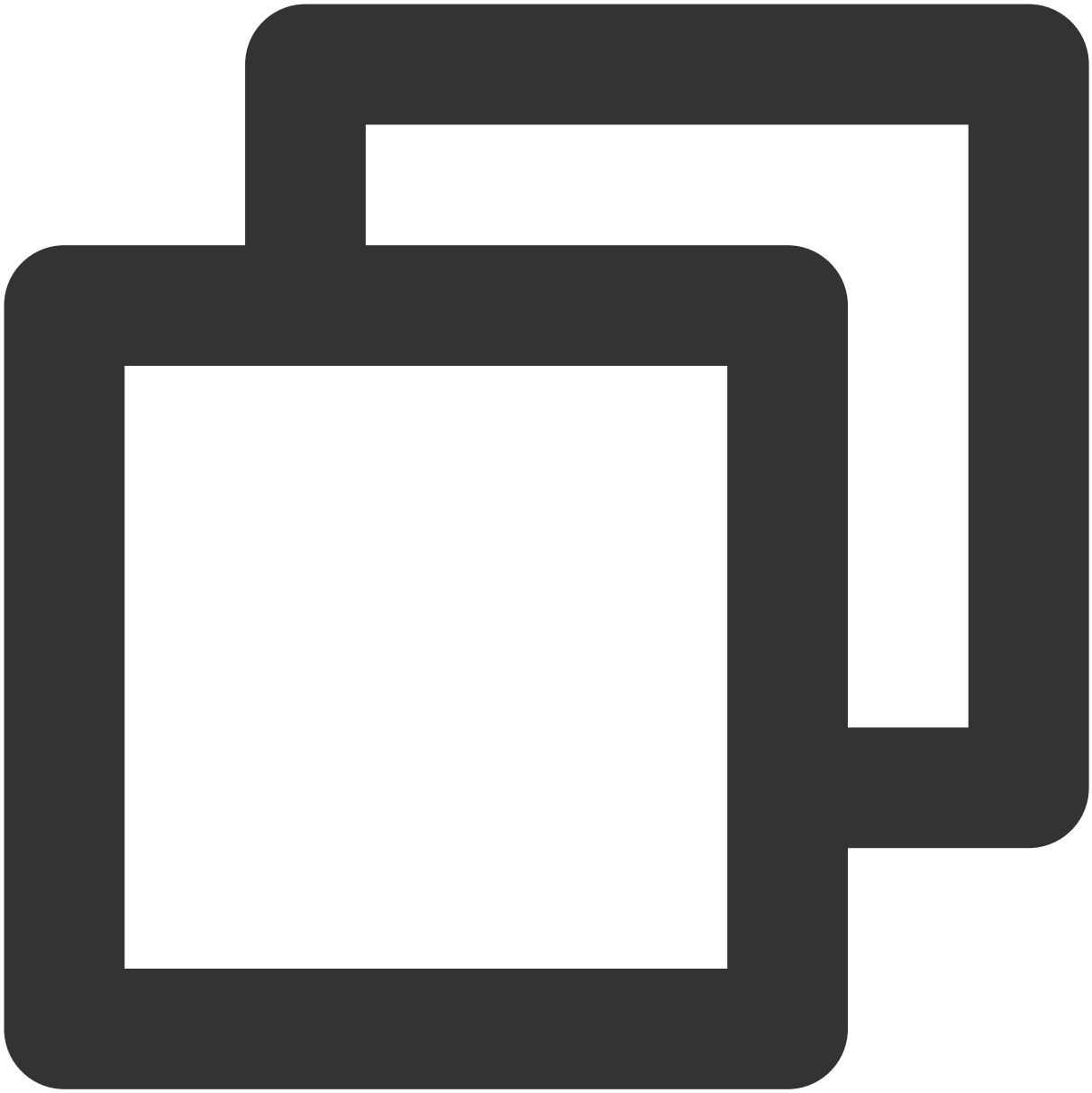


```
hashed := hmac.New(sha1.New, []byte(secretKey))
hashed.Write(buf.Bytes())

fmt.Println(base64.StdEncoding.EncodeToString(hashed.Sum(nil)))
```

5. 获取调用信息并发送请求



```
# 实际调用，成功后如果是消费接口会产生计费(此处以Python语言为例发送get请求)
resp = requests.get("https://" + endpoint, params=data)
print(resp.url)
```



最终得到的请求串为：
`https://cvm.tencentcloudapi.com/?Nonce=11886&SecretId=AKIDz8krbsJ5**********mLPx3EX`

字段名称	解释
endpoint	服务地址， 例如： `cvm.tencentcloudapi.com`
data	API 3.0 签名 V1 所举示例接口参数，**注意** 这里需要将计算的签名已键值对的形式加入data中

注意：

由于示例中的密钥是虚构的，时间戳也不是系统当前时间，因此如果将此 url 在浏览器中打开或者用 curl 等命令调用时会返回鉴权错误：签名过期。为了得到一个可以正常返回的 url ，需要修改示例中的 SecretId 和 SecretKey 为真实的密钥，并使用系统当前时间戳作为 Timestamp 。
为了更清楚的解释签名过程，下面以 Go 语言为例，将上述的签名过程具体实现。请求的域名、调用的接口和参数的取值都以上述签名过程为准，代码只为解释签名过程，并不具备通用性，实际开发请尽量使用 SDK 。

6. 签名串编码

生成的签名串并不能直接作为请求参数，需要对其进行 URL 编码。
如上一步生成的签名串为 `Eli****************cGeI=` ，最终得到的签名串请求参数（Signature）为： `EliP**********************eI%3D` ，它将用于生成最终的请求 URL。
注意：
如果用户的请求方法是 GET，或者请求方法为 POST 同时 Content-Type 为 application/x-www-form-urlencoded，则发送请求时所有请求参数的值均需要做 URL 编码，参数键和=符号不需要编码。非 ASCII 字符在 URL 编码前需要先以 UTF-8 进行编码。
有些编程语言的网络库会自动为所有参数进行 urlencode，在这种情况下，就不需要对签名串进行 URL 编码了，否则两次 URL 编码会导致签名失败。
其他参数值也需要进行编码，编码采用 RFC 3986。使用 `%XY` 对特殊字符例如汉字进行百分比编码，其中 `X` 和 `Y` 为十六进制字符（0-9 和大写字母 A-F），使用小写将引发错误。

7. API 3.0 签名 V1示例

注意：
此处只到计算签名部分，如需获得最终请求串，需要在参数 params 里加上下面所得到的签名键值。

```
package main

import (
    "bytes"
    "crypto/hmac"
    "crypto/sha1"
    "encoding/base64"
    "net/url"
    "fmt"
    "sort"
)
```

```
func main() {
    secretId := "AKIDz8krbsJ5**********mLPx3EXAMPLE"
    secretKey := "Gu5t9xGAR***********EXAMPLE"
    endpoint := "cvm.tencentcloudapi.com"
    params := map[string]string{
        "Nonce":          "11886",
        "Timestamp":      "1465185768",
        "Region":         "ap-guangzhou",
        "SecretId":       secretId,
        "Version":        "2017-03-12",
        "Action":         "DescribeInstances",
        "InstanceIds.0": "ins-09dx96dg",
        "Limit":          "20",
        "Offset":         "0",
    }

    var buf bytes.Buffer
    buf.WriteString("GET")
    buf.WriteString(endpoint)
    buf.WriteString("/")
    buf.WriteString("?")

    // sort keys by ascii asc order
    keys := make([]string, 0, len(params))
    for k, _ := range params {
        keys = append(keys, k)
    }
    sort.Strings(keys)

    for i := range keys {
        k := keys[i]
        buf.WriteString(k)
        buf.WriteString("=")
        buf.WriteString(params[k])
        buf.WriteString("&")
    }
    buf.Truncate(buf.Len() - 1)

    hashed := hmac.New(sha1.New, []byte(secretKey))
    hashed.Write(buf.Bytes())

    signature := base64.StdEncoding.EncodeToString(hashed.Sum(nil))
    fmt.Println(base64.StdEncoding.EncodeToString(hashed.Sum(nil)))
    final_signature := url.QueryEscape(signature)
    fmt.Println(final_signature)
}
```

签名失败

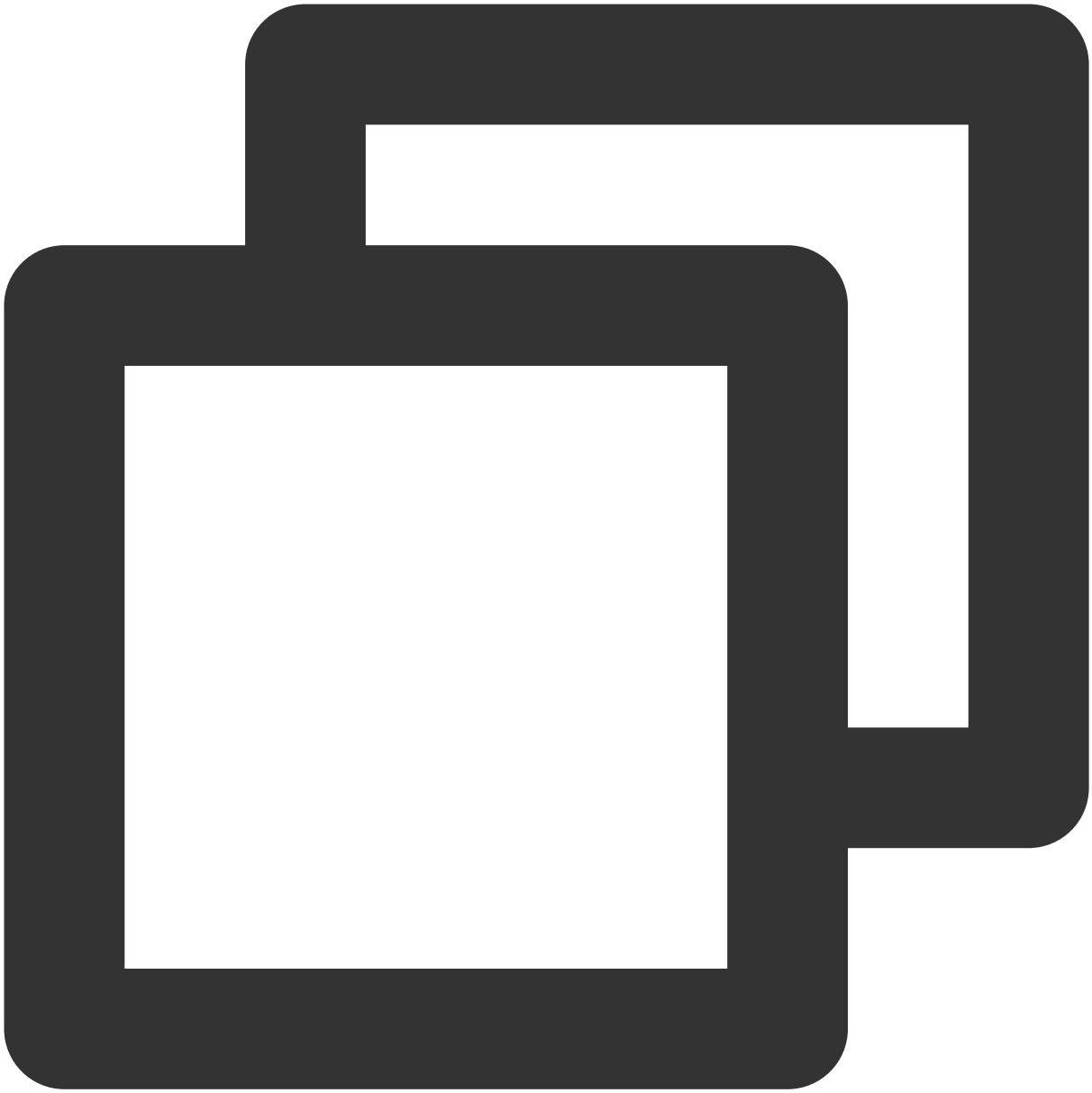
存在以下签名失败的错误码，请根据实际情况处理。

错误码	错误描述
AuthFailure.SignatureExpire	签名过期。Timestamp 与服务器接收到请求的时间相差不得超过五分钟。
AuthFailure.SecretIdNotFound	密钥不存在。请到控制台查看密钥是否被禁用，是否少复制了字符或者多了字符。
AuthFailure.SignatureFailure	签名错误。可能是签名计算错误，或者签名与实际发送的内容不相符合，也有可能是密钥 SecretKey 错误导致的。
AuthFailure.TokenFailure	临时证书 Token 错误。
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。

返回结果

正确返回结果

以云服务器的接口查看实例状态列表（DescribeInstancesStatus）2017-03-12 版本为例，若调用成功，其可能的返回如下为：



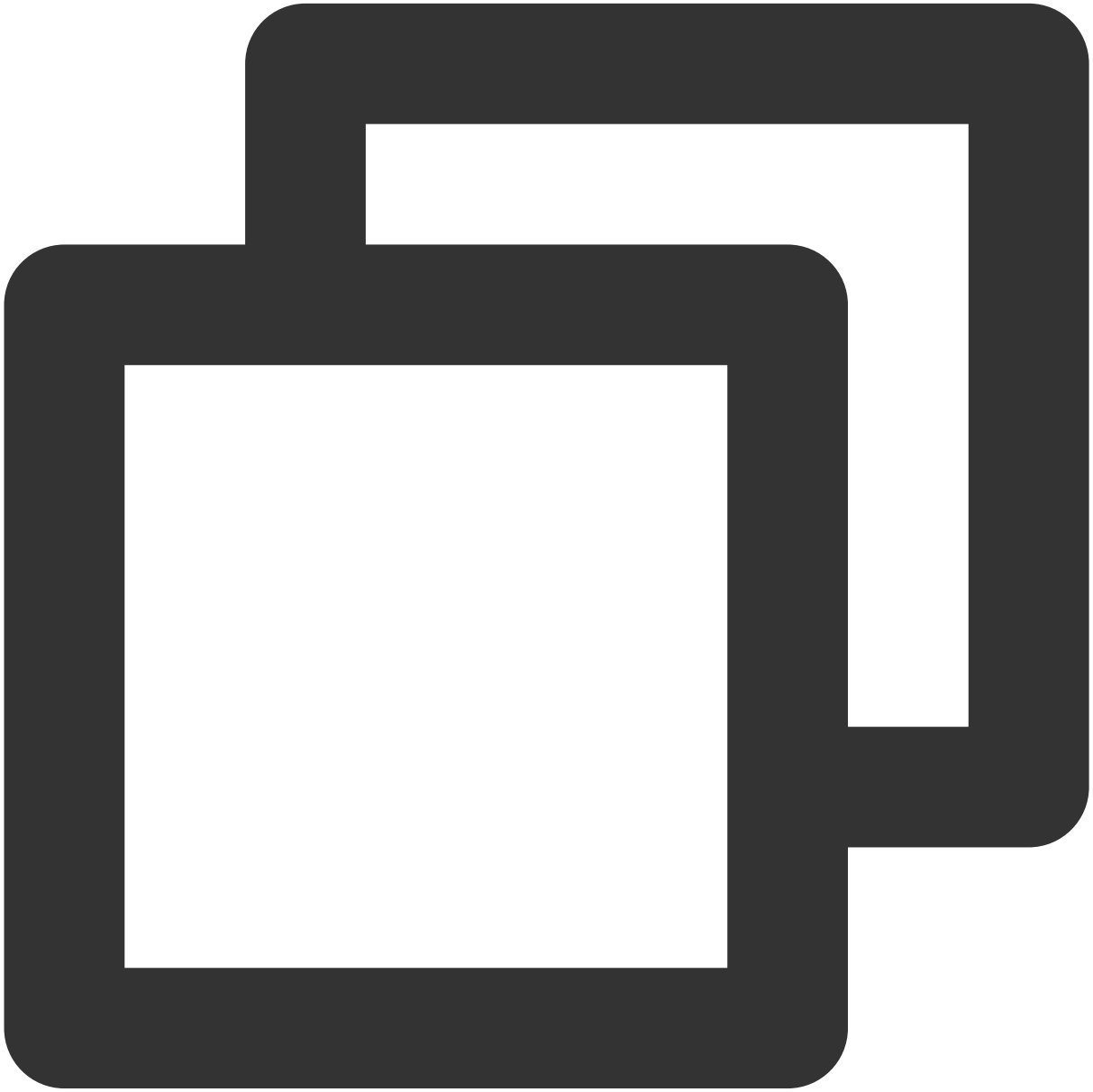
```
{
    "Response": {
        "TotalCount": 0,
        "InstanceStatusSet": [],
        "RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
    }
}
```

`Response` 及其内部的 `RequestId` 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 `TotalCount` 和 `InstanceStatusSet` 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 `TotalCount` 在此情况下的返回值为 0， `InstanceStatusSet` 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：



```
{
    "Response": {
        "Error": {
```

```
            "Code": "AuthFailure.SignatureFailure",
            "Message": "The provided credentials could not be validated. Please che
        },
        "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
    }
}
```

`Error` 的出现代表着该请求调用失败。 `Error` 字段连同其内部的 `Code` 和 `Message` 字段在调用失败时是必定返回的。

`Code` 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。

`Message` 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。

`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。 Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。

LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	HTTPS 请求方法错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

版权所有：腾讯云计算（北京）有限责任公司

Java API

最近更新时间：2023-03-07 18:16:40

腾讯云 API 全新升级3.0，该版本进行了性能优化且全地域部署、支持就近和按地域接入、访问时延下降显著，接口描述更加详细、错误码描述更加全面、SDK 增加接口级注释，让您更加方便快捷的使用腾讯云产品。这里针对 Java API 调用方式进行简单说明。
现已支持云服务器（CVM）、云硬盘（CBS）、私有网络（VPC）、云数据库（TencentDB）等 腾讯云产品，后续会支持其他的云产品接入，敬请期待。

了解请求结构

1. 服务地址（endpoint）

API 支持就近地域接入（例如：cvm 产品域名为 `cvm.tencentcloudapi.com` ），也支持指定地域域名访问（例如：广州地域的域名为 `cvm.ap-guangzhou.tencentcloudapi.com` ），各地域参数见下文公共参数中的地域列表，详情请参见各产品的“请求结构”文档说明判断是否支持该地域。

注意：

对时延敏感的业务，建议指定带地域的域名。

2. 通信协议

腾讯云 API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法：
POST（推荐）
GET
POST 请求支持的 Content-Type 类型：
application/json（推荐），必须使用签名方法 v3（TC3-HMAC-SHA256）。
application/x-www-form-urlencoded，必须使用签名方法 v1（HmacSHA1 或 HmacSHA256）。
multipart/form-data（仅部分接口支持），必须使用签名方法 v3（TC3-HMAC-SHA256）。
GET 请求的请求包大小不得超过32KB。POST 请求使用签名方法 v1（HmacSHA1、HmacSHA256）时不得超过1MB。POST 请求使用签名方法 v3（TC3-HMAC-SHA256）时支持10MB。

4. 字符编码

均使用 UTF-8 编码。

版权所有：腾讯云计算（北京）有限责任公司

公共参数

公共参数是用于标识用户和接口签名的参数，每次请求均需要携带这些参数，才能正常发起请求。

签名方法 V3 公共参数

签名方法 v3（有时也称作 TC3-HMAC-SHA256）相比签名方法 v1（有些文档可能会简称“签名方法”），更安全，支持更大的请求包，支持 POST JSON 格式，性能有一定提升，推荐使用该签名方法计算签名。使用方法见下文 “签名方法介绍” 。

参数名称	类型	必选	描述
X-TC-Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
X-TC-Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702。**注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。**
X-TC-Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本2017-03-12。
Authorization	String	是	HTTP 标准身份认证头部字段，例如：TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=72e494ea8**********a96525168 其中： TC3-HMAC-SHA256：签名方法，目前固定取该值。 Credential：签名凭证，AKIDEXAMPLE 是 SecretId。 Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致。 service 为产品名，通常为域名前缀，例如域名 cvm.tencentcloudapi.com 意味着产品名是 cvm 。本产品取值为 cvm。 SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部。 Signature：签名摘要，计算过程详见下文。
X-TC-Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

签名方法 V1 公共参数

使用签名方法 v1（有时会称作 HmacSHA256 和 HmacSHA1），公共参数需要统一放到请求串中。

参数名称	类型	必选	描述
Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702，如果与当前时间相差过大，会引起签名过期错误。
Nonce	Integer	是	随机正整数，与 Timestamp 联合起来，用于防止重放攻击。
SecretId	String	是	在 云API密钥 上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey，而 SecretKey 会用来生成请求签名 Signature。
Signature	String	是	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。具体计算方法参见下文 “签名方法介绍” 。
Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本2017-03-12。
SignatureMethod	String	否	签名方式，目前支持 HmacSHA256 和 HmacSHA1。只有指定此参数为 HmacSHA256 时，才使用 HmacSHA256 算法验证签名，其他情况均使用 HmacSHA1 验证签名。
Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token 。

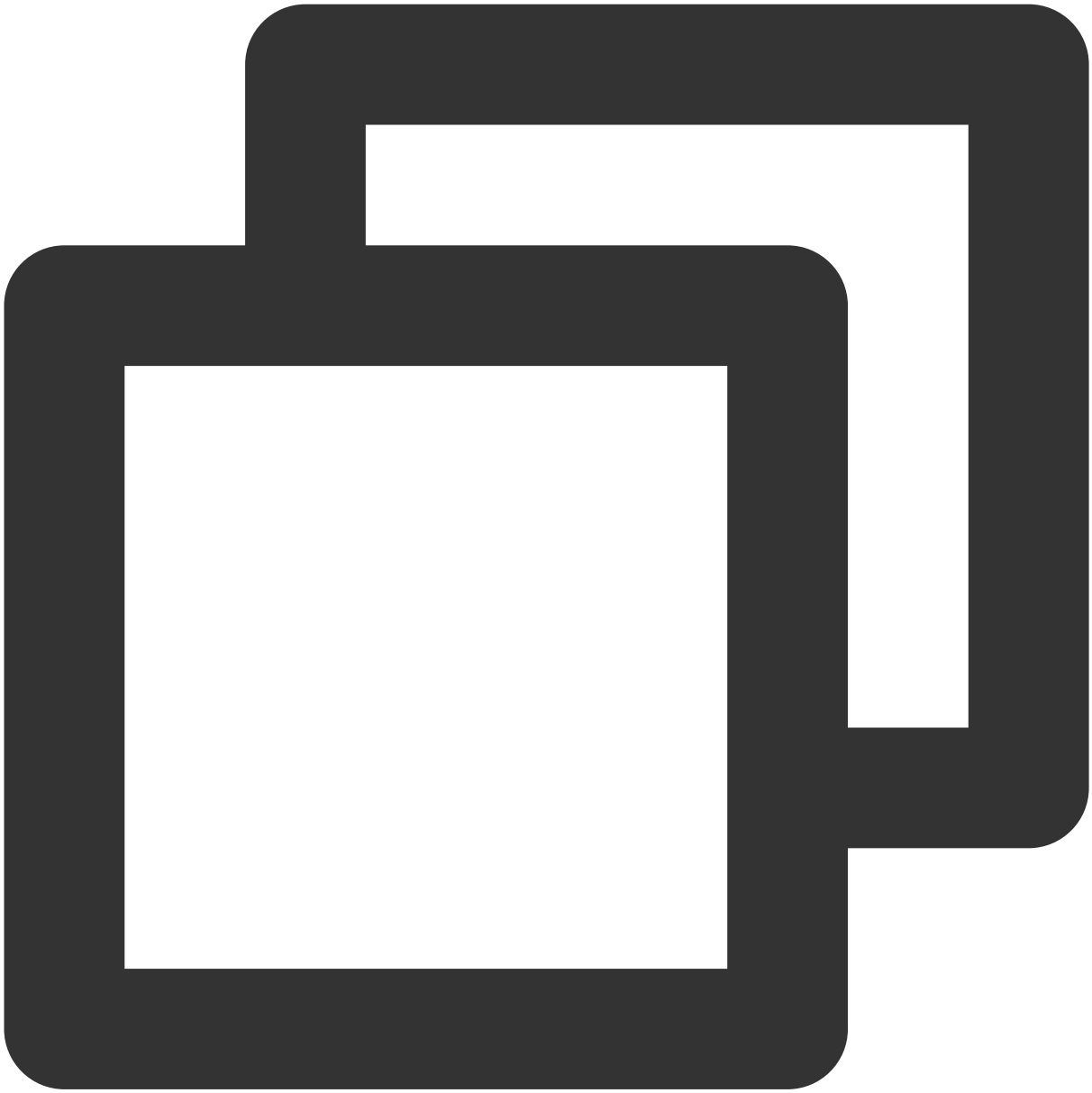
地域列表

由于各个产品支持地域不同，具体详情请参考各产品文档中的地域列表。
例如，您可以参考云服务器的 地域列表。

Java API 调用方式

腾讯云 API 会对每个请求进行身份验证，用户需要使用安全凭证，经过特定的步骤对请求进行签名（Signature），每个请求都需要在公共请求参数中指定该签名结果并以指定的方式和格式发送请求。

假设用户的 SecretId 和 SecretKey 分别是： `AKIDz8krbsJ5*********mLPx3EXAMPL` 和 `Gu5t9xGAR**********EXAMPLE` 。用户想查看广州区云服务器名为“未命名”的主机状态，只返回一条数据。则请求可能为：



```
curl -X POST https://cvm.tencentcloudapi.com \\
-H "Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5*********mLPx3EXAMPL/201
-H "Content-Type: application/json; charset=utf-8" \\
-H "Host: cvm.tencentcloudapi.com" \\
-H "X-TC-Action: DescribeInstances" \\
-H "X-TC-Timestamp: 1551113065" \\
-H "X-TC-Version: 2017-03-12" \\
```

```
-H "X-TC-Region: ap-guangzhou" \\
-d '{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instanc
```

步骤1：申请安全凭证

本文使用的安全凭证为密钥，密钥包括 SecretId 和 SecretKey。每个用户最多可以拥有两对密钥。

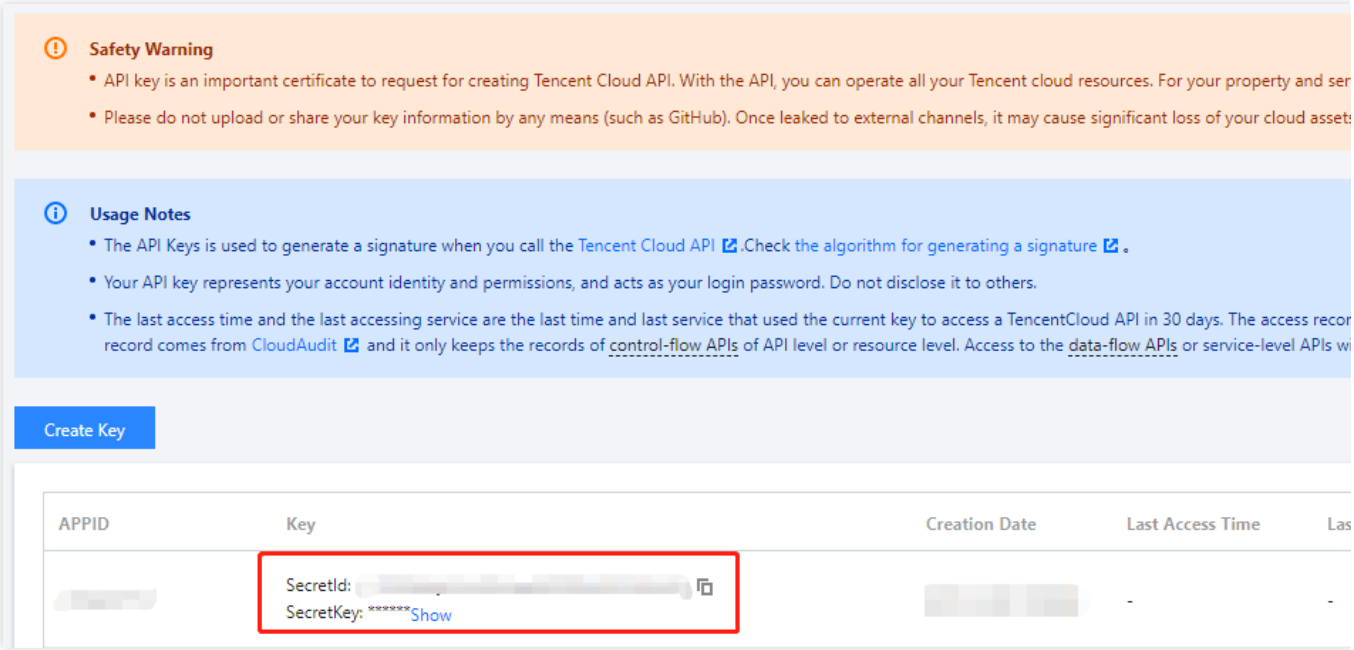
SecretId：用于标识 API 调用者身份，可以简单类比为用户名。

SecretKey：用于验证 API 调用者的身份，可以简单类比为密码。

注意：

用户必须严格保管安全凭证，避免泄露，否则将危及财产安全。如已泄漏，请立刻禁用该安全凭证。

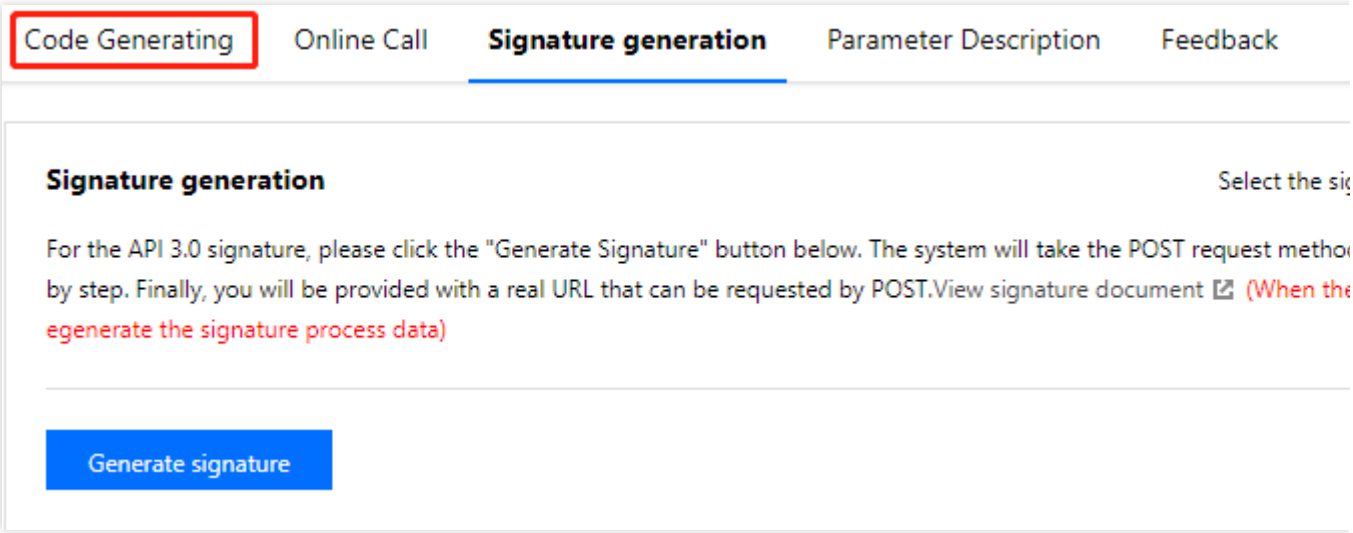
前往 API密钥管理 页面，即可进行获取。如下图所示：



步骤2：

1. 获取 API 3.0 V3 版本签名

签名方法 v3（TC3-HMAC-SHA256）功能上覆盖了以前的签名方法 v1 ，而且更安全，支持更大的请求，支持 json 格式，性能有一定提升，推荐使用该签名方法计算签名。



注意：

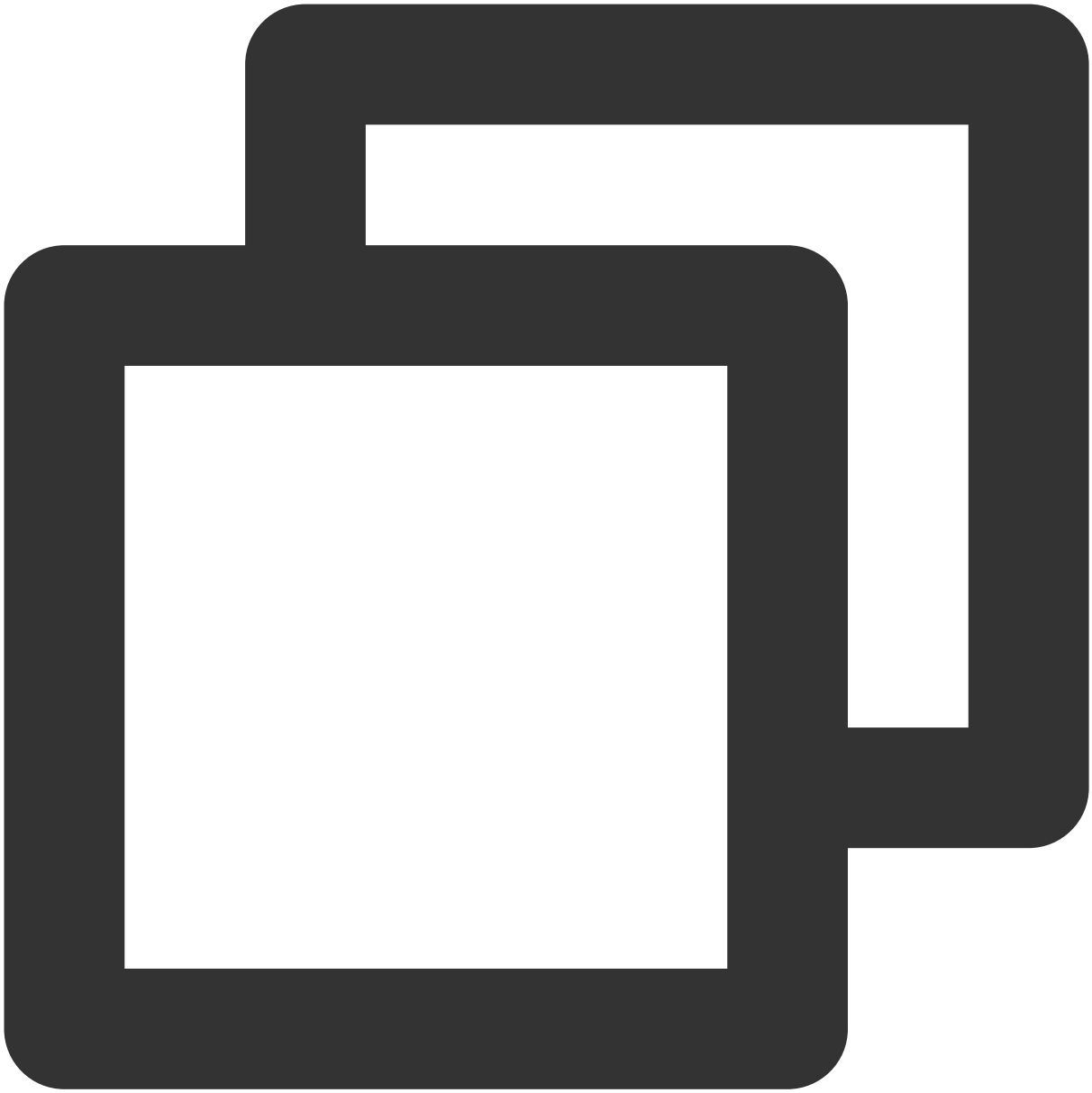
首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“ API 3.0 签名 v3 ”，可以生成签名过程进行验证，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的7种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

云 API 支持 GET 和 POST 请求。对于 GET 方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。对于 POST 方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式绝大多数接口均支持，multipart 格式只有特定接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。推荐使用 POST 请求，因为两者的结果并无差异，但 GET 请求只支持32KB以内的请求包。

下面以 查看实例列表 接口为例，分步骤介绍签名的计算过程。我们选择该接口是因为：

1. 云服务器默认已开通，该接口很常用。
2. 该接口是只读的，不会改变现有资源的状态。
3. 接口覆盖的参数种类较全，可以演示包含数据结构的数组如何使用。

1. 拼接规范请求串

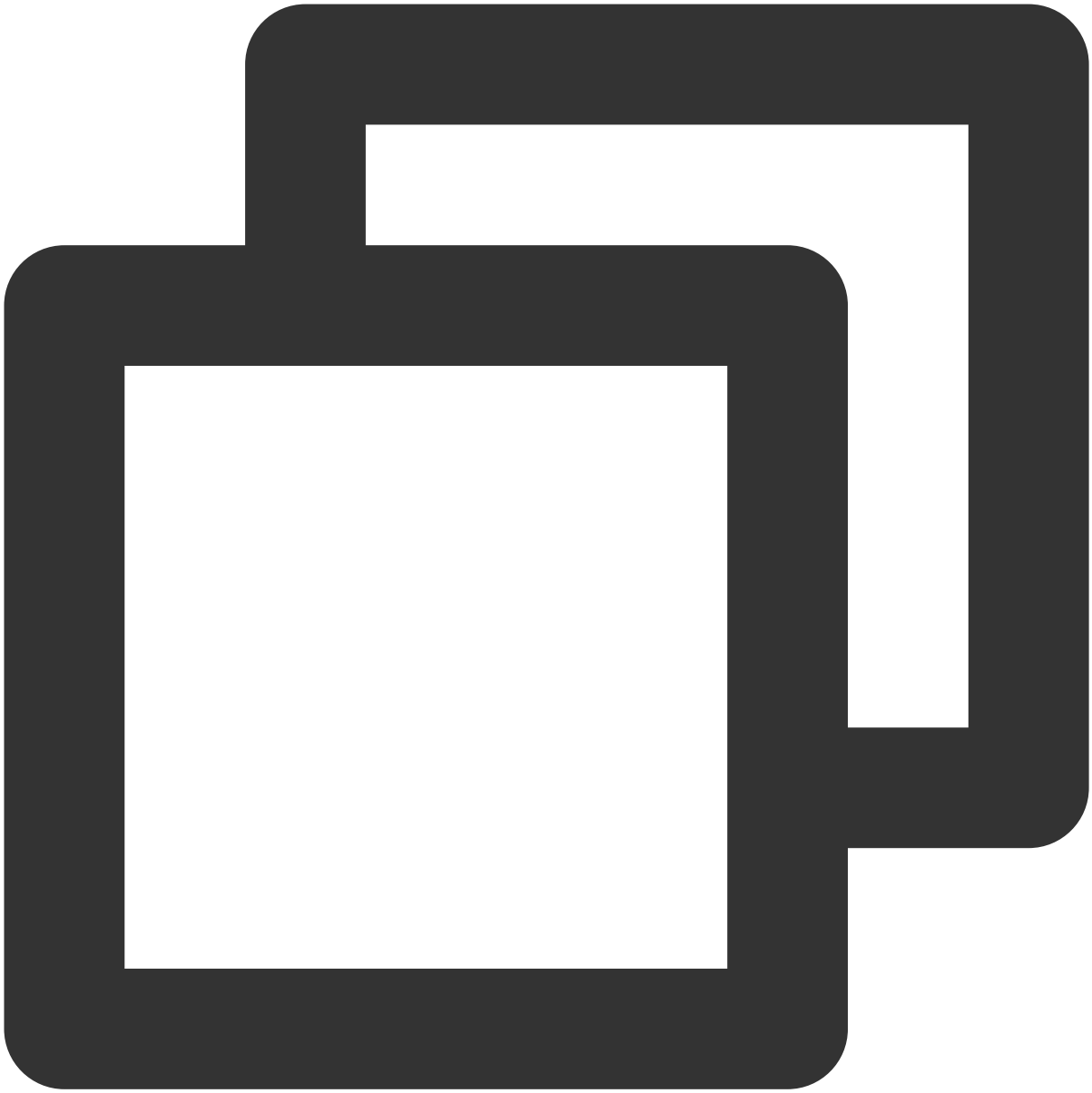


```
CanonicalRequest =
    HTTPRequestMethod + '\\n' +
    CanonicalURI + '\\n' +
    CanonicalQueryString + '\\n' +
    CanonicalHeaders + '\\n' +
    SignedHeaders + '\\n' +
    HashedRequestPayload
```

字段名称	解释

HTTPRequestMethod	HTTP 请求方法（GET、POST ）。此示例取值为 `POST` 。
CanonicalURI	URI 参数，API 3.0 固定为正斜杠（/）。
CanonicalQueryString	发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串""，对于 G 求，则为 URL 中问号（?）后面的字符串内容，例如：Limit=10&Offset=0。 注意：CanonicalQueryString 需要参考 RFC3986 进行 URLEncode，字符集 UTF8，推荐使用 语言标准库，所有特殊字符均需编码，大写形式。
CanonicalHeaders	参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部 签名以提高自身请求的唯一性和安全性。 拼接规则：头部 key 和 value 统一转成小写，掉首尾空格，按照 key:value\\n 格式拼接；多个头部，按照头部 key（小写）的 ASCII 行拼接。此示例计算结果是 `content-type:application/json; charset=ut 8\\nhost:cvm.tencentcloudapi.com\\n` 。 **注意：content-type 必须和实际发送的相符合，有些编程语言网络库即使未指定也会 加 charset 值，如果签名时和发送时不一致，服务器会返回签名校验失败。**
SignedHeaders	参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包 部内容是一一对应的。content-type 和 host 为必选头部。 拼接规则：头部 key 统一转 写；多个头部 key（小写）按照 ASCII 升序进行拼接，并且以分号（;）分隔。此示例 content-type;host
HashedRequestPayload	请求正文（Requestpayload，即 body，此示例为 `{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-name"` 的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(RequestPayload))), HTTP 请求正文做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。 GET 请求，RequestPayload 固定为空字符串。此示例计算结果是 `35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f`

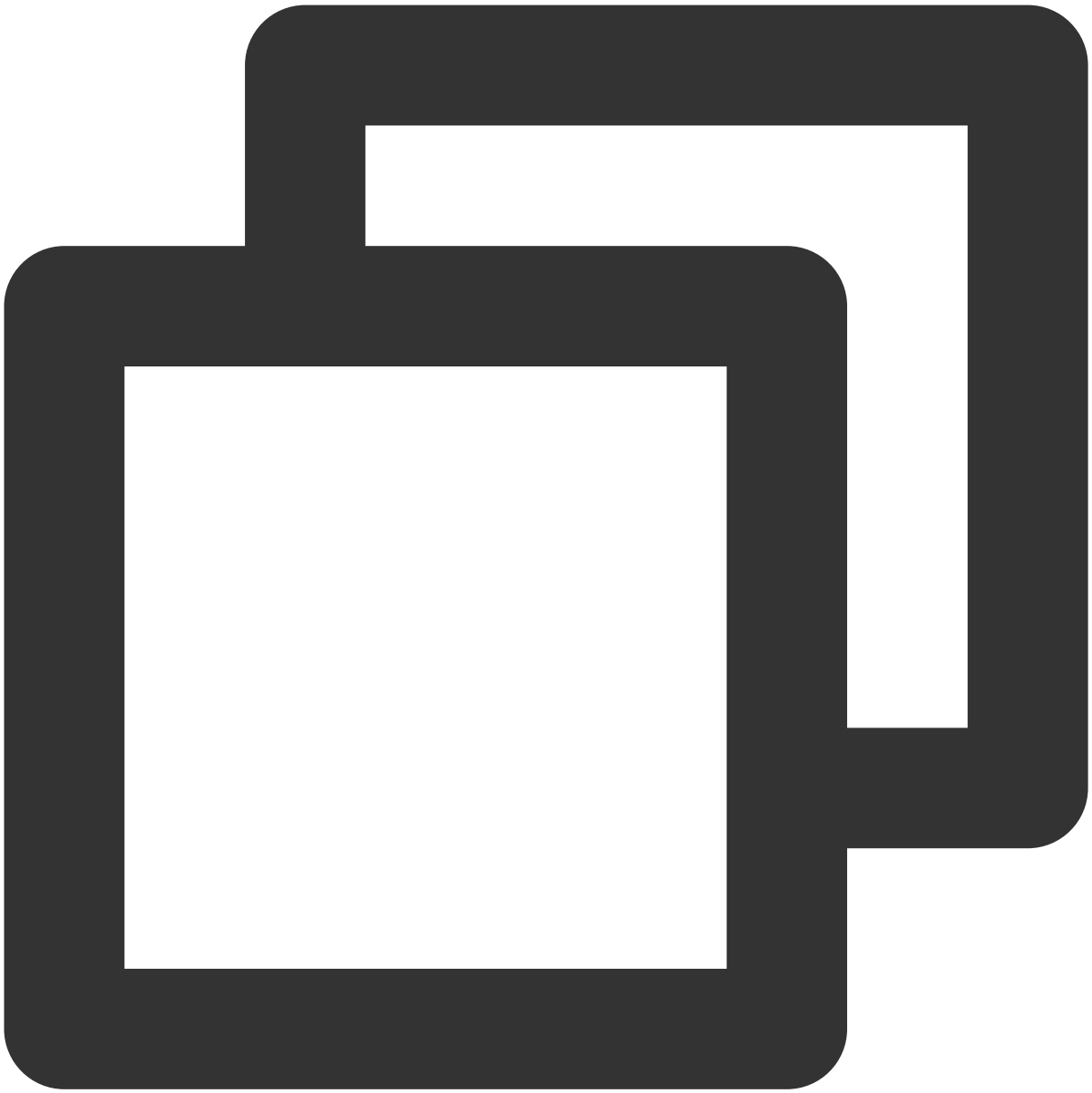
根据以上规则，示例中得到的规范请求串如下：



```
POST
/

content-type:application/json; charset=utf-8
host:cvm.tencentcloudapi.com

content-type;host
35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f064
```

2. 拼接待签名字符串

按如下格式拼接待签名字符串：



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

字段名称	解释
Algorithm	签名算法，目前固定为 `TC3-HMAC-SHA256` 。

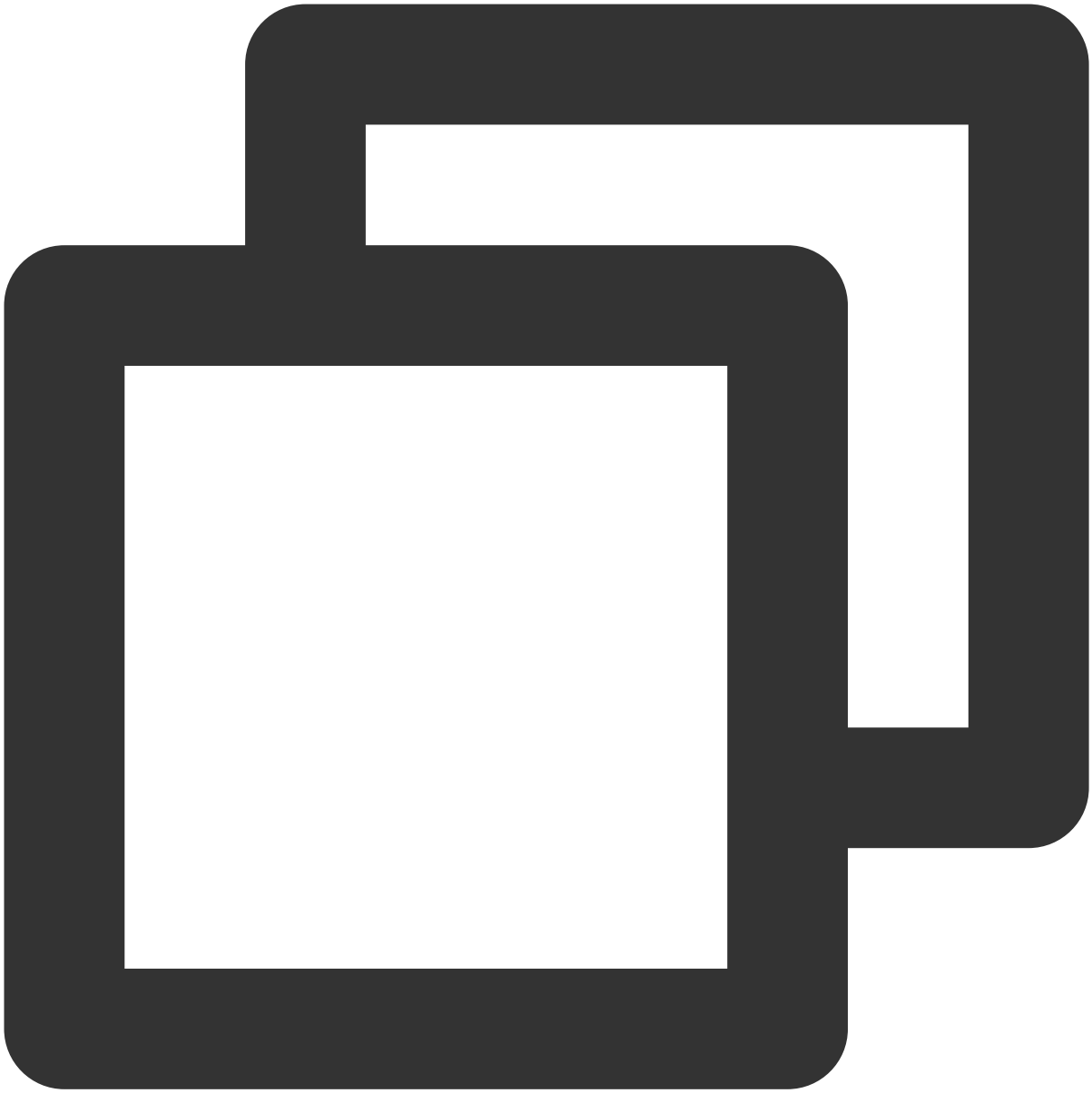
RequestTimestamp	请求时间戳，即请求头部的公共参数 X-TC-Timestamp 取值，取当前时间 UNIX 时间确到秒。此示例取值为 `1551113065` 。
CredentialScope	凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符串（tc3_request）。**Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Time 换算的 UTC 标准时间日期一致**；service 为产品名，必须与调用的产品域名一致。此算结果是 2019-02-25/cvm/tc3_request。
HashedCanonicalRequest	前述步骤拼接所得规范请求串的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))。此示例计算结果是 `5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d`

注意：

Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为 1551113065，在东八区的时间是 2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为 2019-02-25，而不是 2019-02-26。

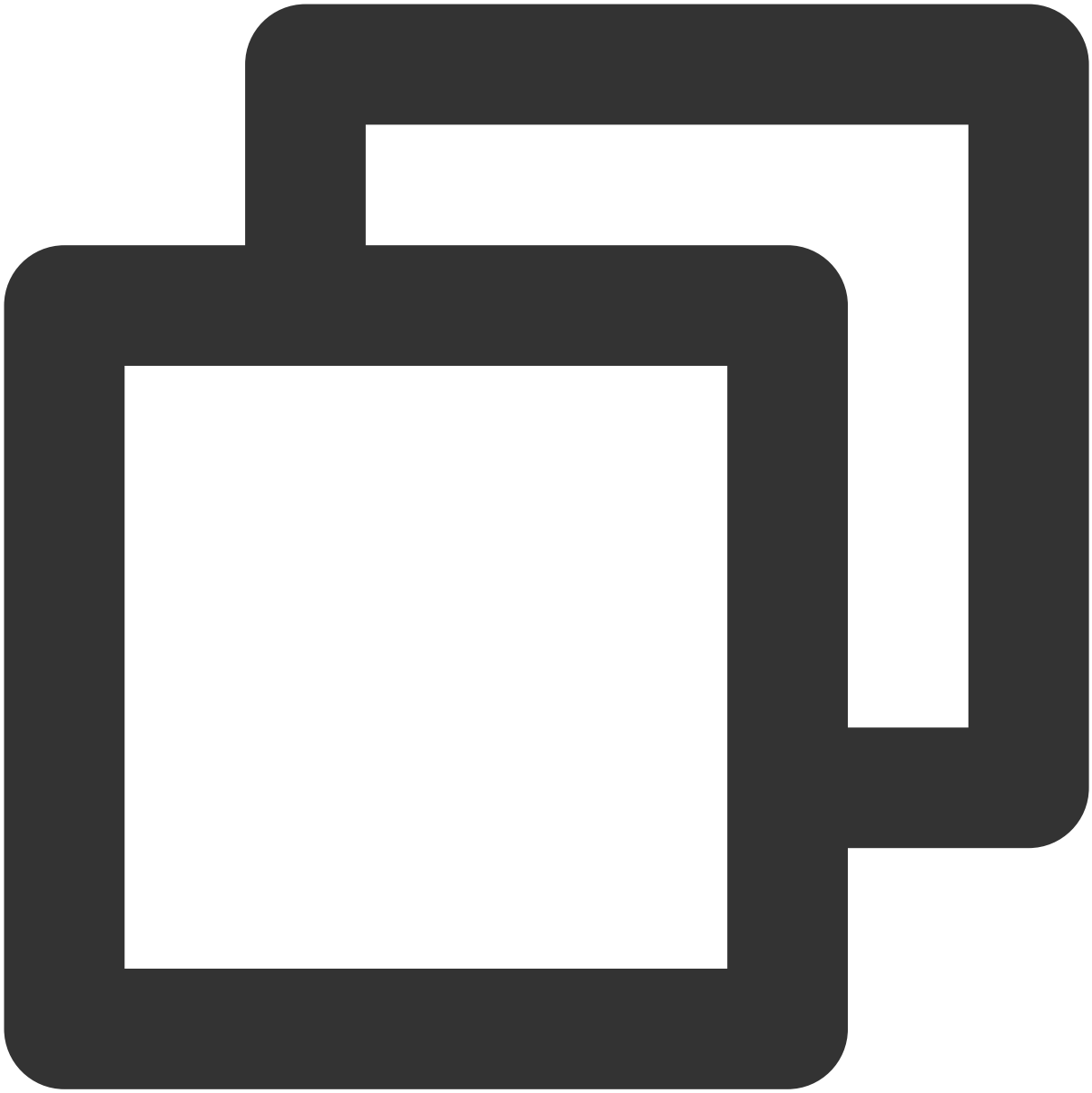
Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败，返回签名过期错误。

根据以上规则，示例中得到的待签名字符串如下：



```
TC3-HMAC-SHA256
1551113065
2019-02-25/cvm/tc3_request
5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d7031
```

3. 计算签名(伪代码)



```
SecretKey = "Gu5t9xGAR***********EXAMPLE"
byte[] secretDate = hmac256(("TC3" + SecretKey).getBytes(UTF8), date);
byte[] secretService = hmac256(secretDate, service);
byte[] secretSigning = hmac256(secretService, "tc3_request");
String signature = DatatypeConverter.printHexBinary(hmac256(secretSigning, stringTo
System.out.println(signature);
```

派生出的密钥 `SecretDate` 、 `SecretService` 和 `SecretSigning` 是二进制的数据，可能包含不可打印字符，此处不展示中间结果。

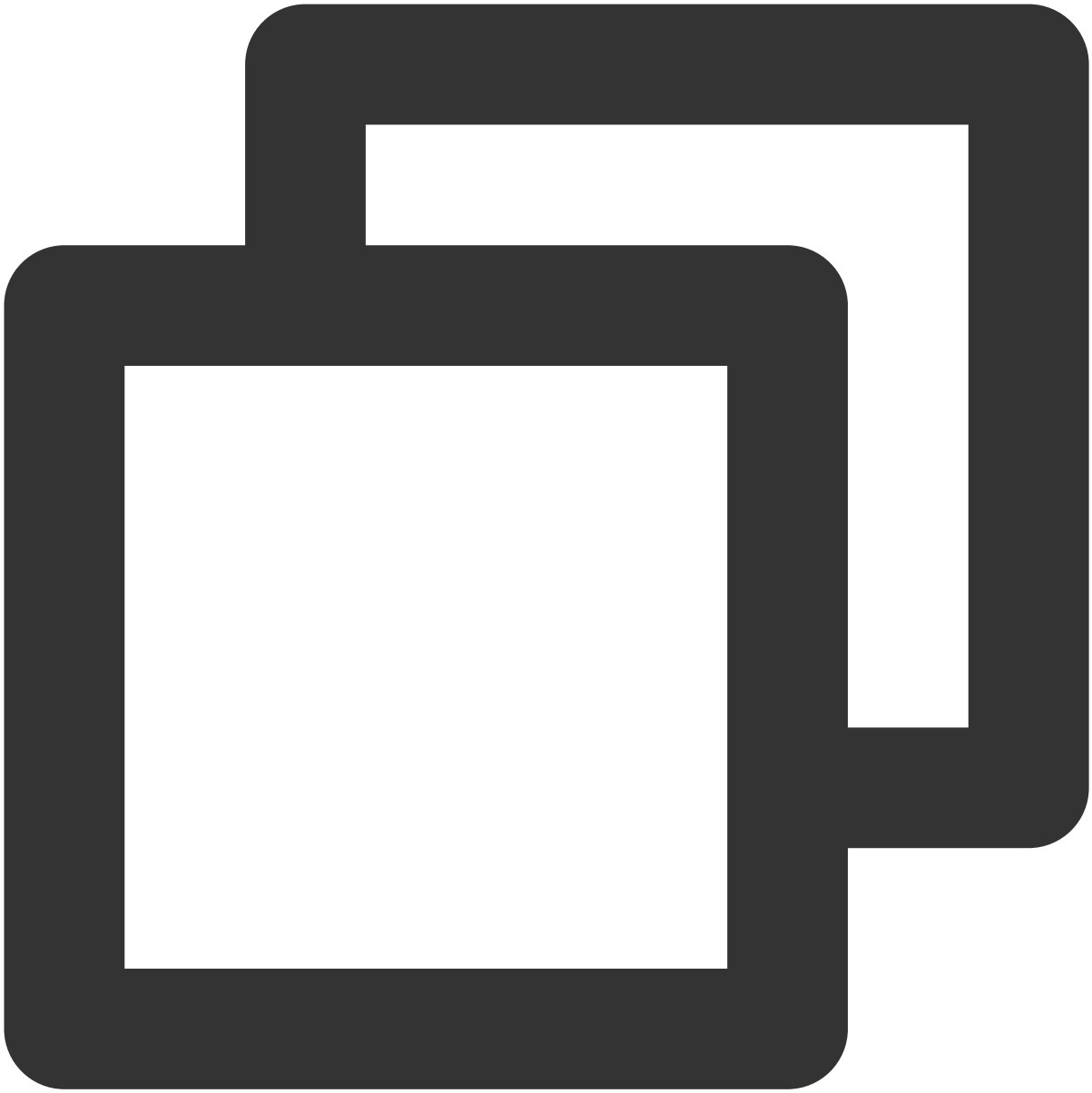
字段名称	解释

SecretKey	原始的 SecretKey，即 `Gu5t9xGAR**********EXAMPLE` 。
date	即 Credential 中的 Date 字段信息。此示例取值为 `2019-02-25` 。
service	即 Credential 中的 Service 字段信息。此示例取值为 `cvm`

此示例计算结果是 `72e494ea8****************************************a96525168` 。

4. 拼接 Authorization

按如下格式拼接 Authorization：



```
String Authorization =
    Algorithm + ' ' +
    'Credential=' + SecretId + '/' + CredentialScope + ', ' +
    'SignedHeaders=' + SignedHeaders + ', ' +
    'Signature=' + Signature
```

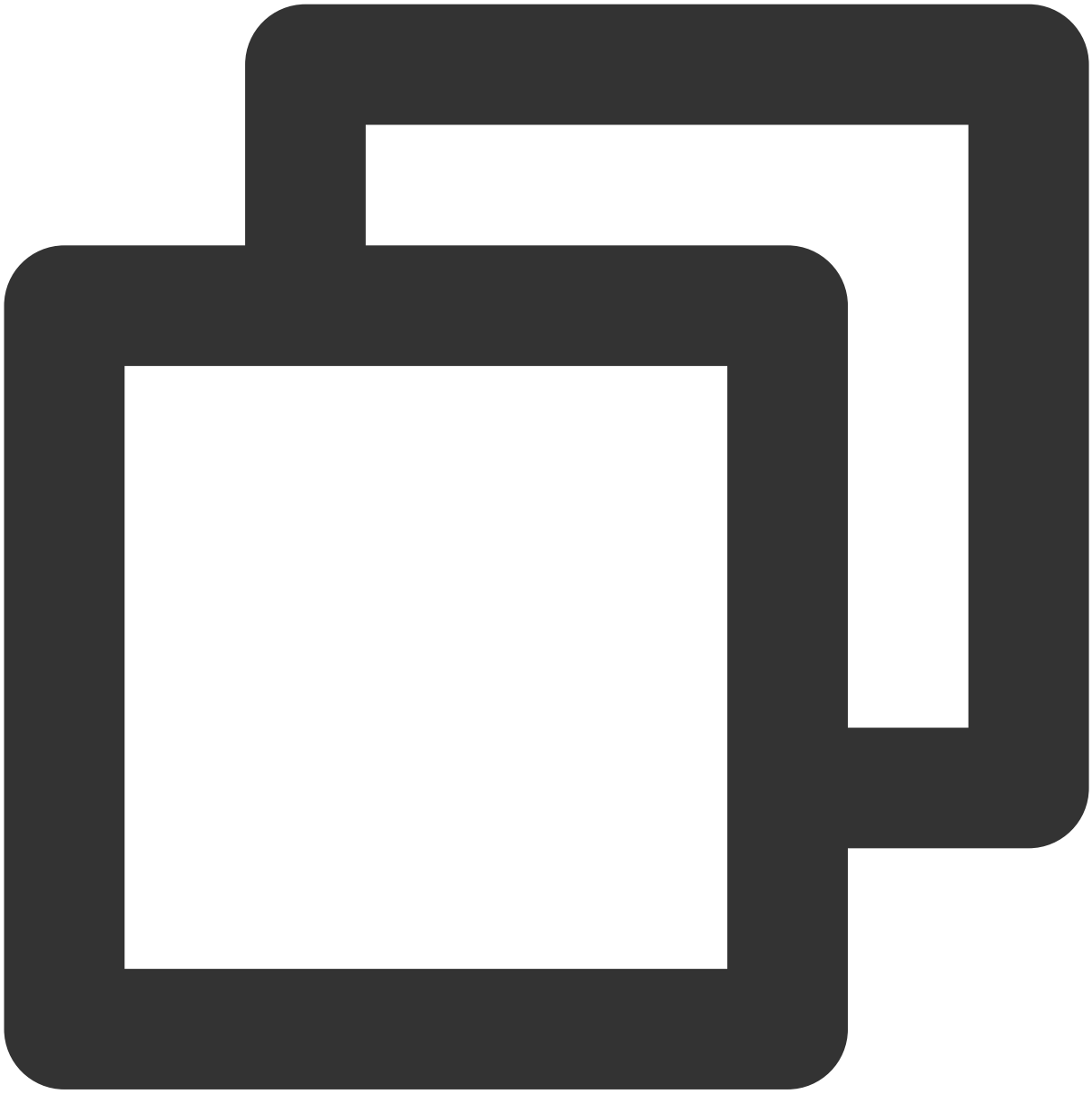
字段名称	解释
Algorithm	签名方法，固定为 `TC3-HMAC-SHA256` 。
SecretId	

	密钥对中的 SecretId，即 `AKIDz8krbsJ5**********mLPx3EXAMPL` 。
CredentialScope	见上文，凭证范围。此示例计算结果是 `2019-02-25/cvm/tc3_request` 。
SignedHeaders	见上文，参与签名的头部信息。此示例取值为 `content-type;host` 。
Signature	签名值。此示例计算结果是 `72e494ea8****************************************a96525168` 。

根据以上规则，示例中得到的值为：

```
TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPL/2019-02-25/cvm/tc3_req
```

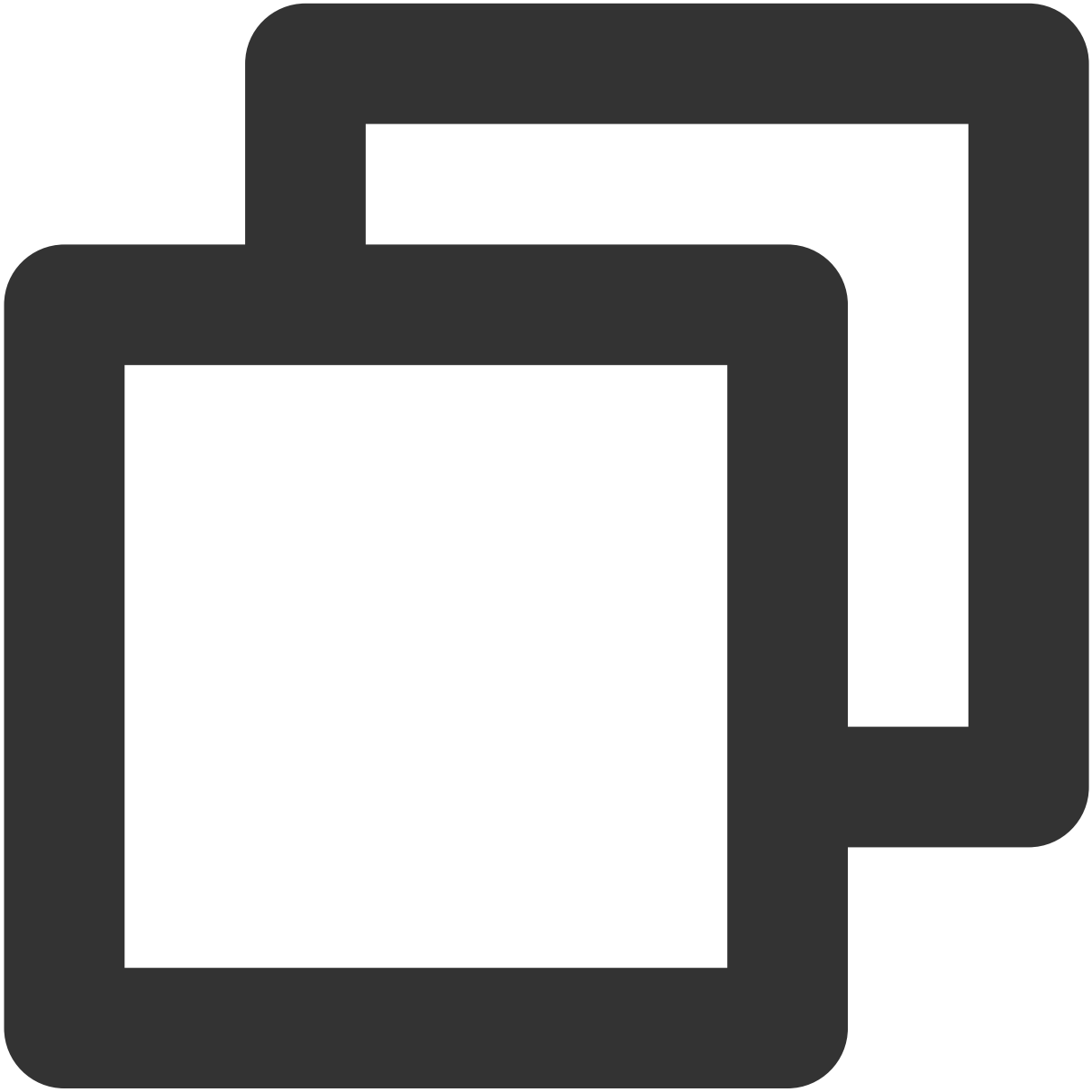
最终完整的调用信息如下：



```
POST https://cvm.tencentcloudapi.com/
Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPL/2019-02
Content-Type: application/json; charset=utf-8
Host: cvm.tencentcloudapi.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1551113065
```

```
X-TC-Region: ap-guangzhou

{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-na
```

5. API 3.0 签名 V3示例



```
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
```

```
import java.util.TimeZone;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

public class TencentCloudAPITC3Demo {
    private final static Charset UTF8 = StandardCharsets.UTF_8;
    private final static String SECRET_ID = "AKIDz8krbsJ5**********mLPx3EXAMPL";
    private final static String SECRET_KEY = "Gu5t9xGAR***********EXAMPLE";
    private final static String CT_JSON = "application/json; charset=utf-8";

    public static byte[] hmac256(byte[] key, String msg) throws Exception {
        Mac mac = Mac.getInstance("HmacSHA256");
        SecretKeySpec secretKeySpec = new SecretKeySpec(key, mac.getAlgorithm());
        mac.init(secretKeySpec);
        return mac.doFinal(msg.getBytes(UTF8));
    }

    public static String sha256Hex(String s) throws Exception {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] d = md.digest(s.getBytes(UTF8));
        return DatatypeConverter.printHexBinary(d).toLowerCase();
    }

    public static void main(String[] args) throws Exception {
        String service = "cvm";
        String host = "cvm.tencentcloudapi.com";
        String region = "ap-guangzhou";
        String action = "DescribeInstances";
        String version = "2017-03-12";
        String algorithm = "TC3-HMAC-SHA256";
        String timestamp = "1551113065";
        //String timestamp = String.valueOf(System.currentTimeMillis() / 1000);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        // 注意时区，否则容易出错
        sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
        String date = sdf.format(new Date(Long.valueOf(timestamp + "000")));

        // ************* 步骤 1：拼接规范请求串 *************
        String httpRequestMethod = "POST";
        String canonicalUri = "/";
        String canonicalQueryString = "";
        String canonicalHeaders = "content-type:application/json; charset=utf-8\\n"
        String signedHeaders = "content-type;host";

        String payload = "{\\"Limit\\": 1, \\"Filters\\": [{\\"Values\\": [\\"\\\\u
```

```
        String hashedRequestPayload = sha256Hex(payload);
        String canonicalRequest = httpRequestMethod + "\\n" + canonicalUri + "\\n"
                + canonicalHeaders + "\\n" + signedHeaders + "\\n" + hashedRequestP
        System.out.println(canonicalRequest);

        // ************* 步骤 2：拼接待签名字符串 *************
        String credentialScope = date + "/" + service + "/" + "tc3_request";
        String hashedCanonicalRequest = sha256Hex(canonicalRequest);
        String stringToSign = algorithm + "\\n" + timestamp + "\\n" + credentialSco
        System.out.println(stringToSign);

        // ************* 步骤 3：计算签名 *************
        byte[] secretDate = hmac256(("TC3" + SECRET_KEY).getBytes(UTF8), date);
        byte[] secretService = hmac256(secretDate, service);
        byte[] secretSigning = hmac256(secretService, "tc3_request");
        String signature = DatatypeConverter.printHexBinary(hmac256(secretSigning,
        System.out.println(signature);

        // ************* 步骤 4：拼接 Authorization *************
        String authorization = algorithm + " " + "Credential=" + SECRET_ID + "/" +
                + "SignedHeaders=" + signedHeaders + ", " + "Signature=" + signatur
        System.out.println(authorization);

        TreeMap<String, String> headers = new TreeMap<String, String>();
        headers.put("Authorization", authorization);
        headers.put("Content-Type", CT_JSON);
        headers.put("Host", host);
        headers.put("X-TC-Action", action);
        headers.put("X-TC-Timestamp", timestamp);
        headers.put("X-TC-Version", version);
        headers.put("X-TC-Region", region);

        StringBuilder sb = new StringBuilder();
        sb.append("curl -X POST https://").append(host)
        .append(" -H \\"Authorization: ").append(authorization).append("\\"")
        .append(" -H \\"Content-Type: application/json; charset=utf-8\\"")
        .append(" -H \\"Host: ").append(host).append("\\"")
        .append(" -H \\"X-TC-Action: ").append(action).append("\\"")
        .append(" -H \\"X-TC-Timestamp: ").append(timestamp).append("\\"")
        .append(" -H \\"X-TC-Version: ").append(version).append("\\"")
        .append(" -H \\"X-TC-Region: ").append(region).append("\\"")
        .append(" -d '").append(payload).append("'");
        System.out.println(sb.toString());
    }
}
```

2. 获取 API 3.0 V1 版本签名

签名方法 v1 简单易用，但是功能和安全性都不如签名方法 v3，推荐使用签名方法 v3。

注意：

首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“ API 3.0 签名 v1 ”，可以生成签名过程进行验证，并提供了部分编程语言的签名示例，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的 7 种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

以云服务器查看实例列表（DescribeInstances）请求为例，当用户调用这一接口时，其请求参数可能如下:

参数名称	中文	参数值
Action	方法名	DescribeInstances
SecretId	密钥 ID	`AKIDz8krbsJ5**********mLPx3EXAMPL`
Timestamp	当前时间戳	1465185768
Nonce	随机正整数	11886
Region	实例所在区域	ap-guangzhou
InstanceIds.0	待查询的实例 ID	ins-09dx96dg
Offset	偏移量	0
Limit	最大允许输出	20
Version	接口版本号	2017-03-12

1. 对参数排序

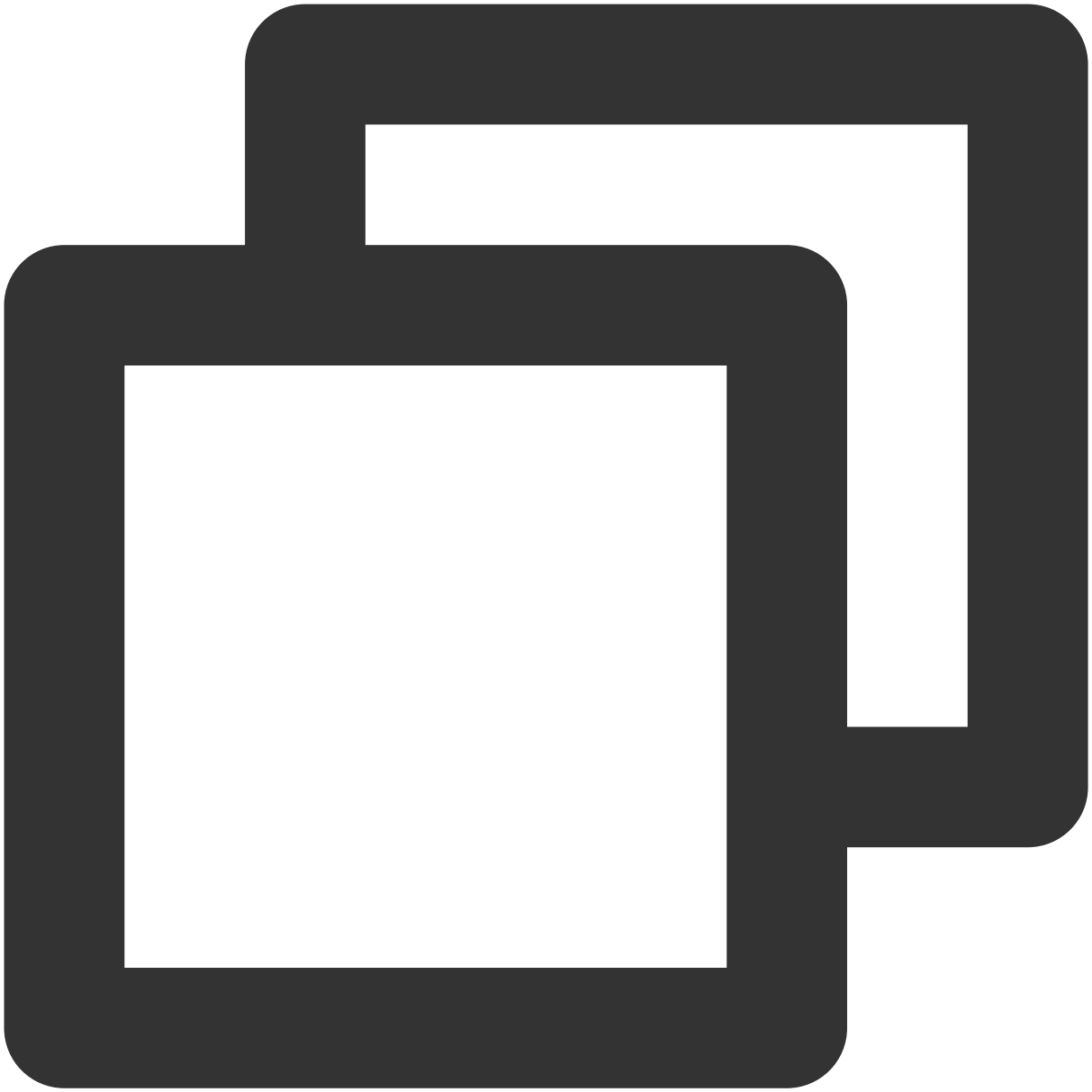
首先对所有请求参数按参数名的字典序（ ASCII 码）升序排序。

注意：

只按参数名进行排序，参数值保持对应即可，不参与比大小。

按 ASCII 码比大小，如 InstanceIds.2 要排在 InstanceIds.12 后面，不是按字母表，也不是按数值。用户可以借助编程语言中的相关排序函数来实现这一功能，如 PHP 中的 ksort 函数。

上述示例参数的排序结果如下：



```
{
    'Action' : 'DescribeInstances',
    'InstanceIds.0' : 'ins-09dx96dg',
    'Limit' : 20,
    'Nonce' : 11886,
    'Offset' : 0,
    'Region' : 'ap-guangzhou',
    'SecretId' : 'AKIDz8krbsJ5**********mLPx3EXAMPL',
    'Timestamp' : 1465185768,
    'Version': '2017-03-12',
}
```

使用程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

2. 拼接规范请求串

此步骤生成请求字符串。 将把上一步排序好的请求参数格式化成“参数名称=参数值”的形式，如对 Action 参数，其参数名称为 " Action " ， 参数值为 " DescribeInstances " ， 因此格式化后就为 Action=DescribeInstances 。

注意：

“参数值”为原始值而非 url 编码后的值。

然后将格式化后的各个参数用"&"拼接在一起，最终生成的请求字符串为:

```
Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&R
```

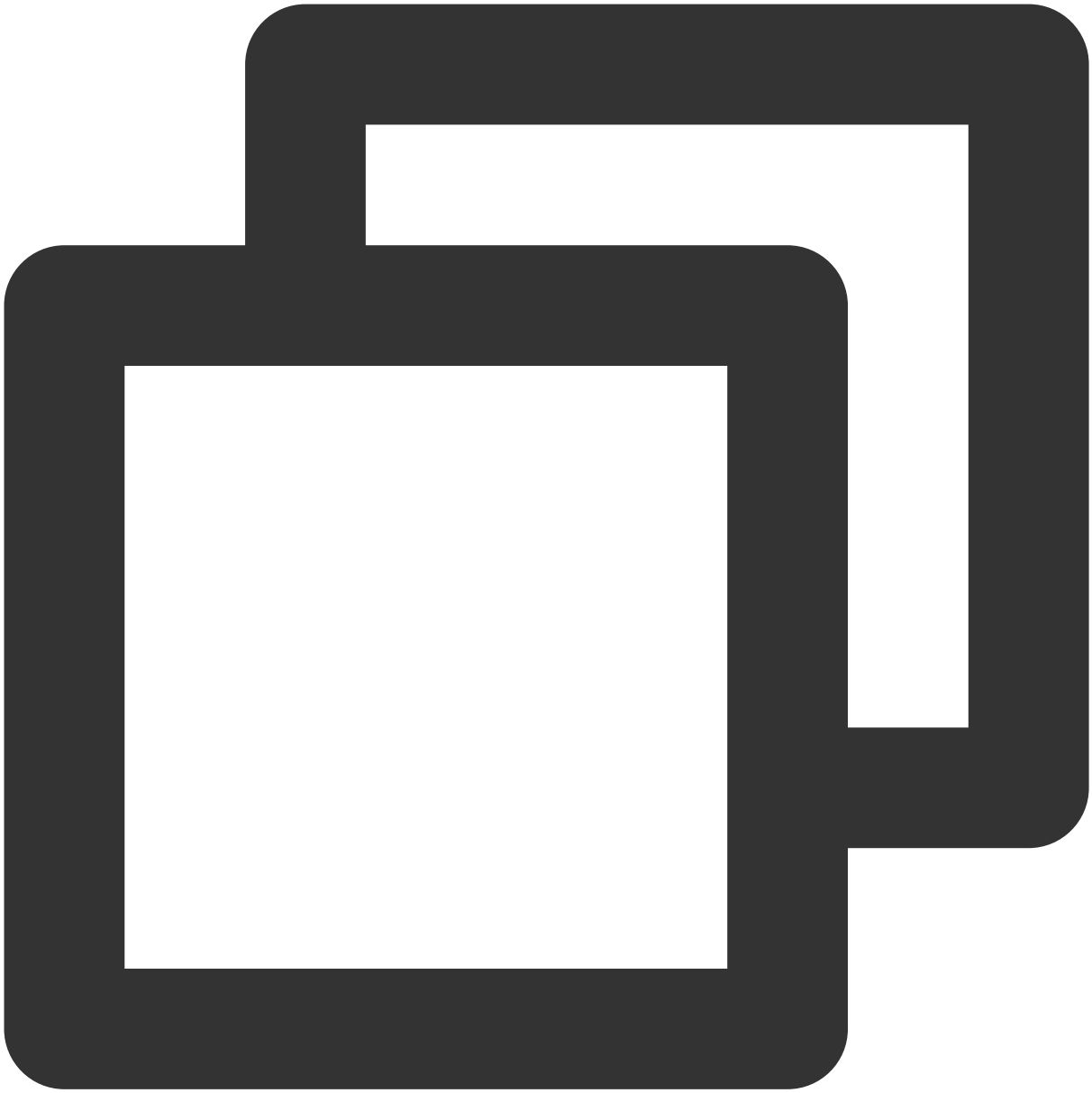
3. 拼接待签名字符串

此步骤生成签名原文字符串。 签名原文字符串由以下几个参数构成:

1. 请求方法：支持 POST 和 GET 方式，这里使用 GET 请求，注意方法为全大写。
2. 请求主机：查看实例列表（DescribeInstances）的请求域名为 `cvm.tencentcloudapi.com` 。实际的请求域名根据接口所属模块的不同而不同，详见各接口说明。
3. 请求路径：当前版本云 API 的请求路径固定为 `/` 。
4. 请求字符串：即上一步生成的请求字符串。

签名原文串的拼接规则为： `请求方法 + 请求主机 +请求路径 + ? + 请求字符串` 。

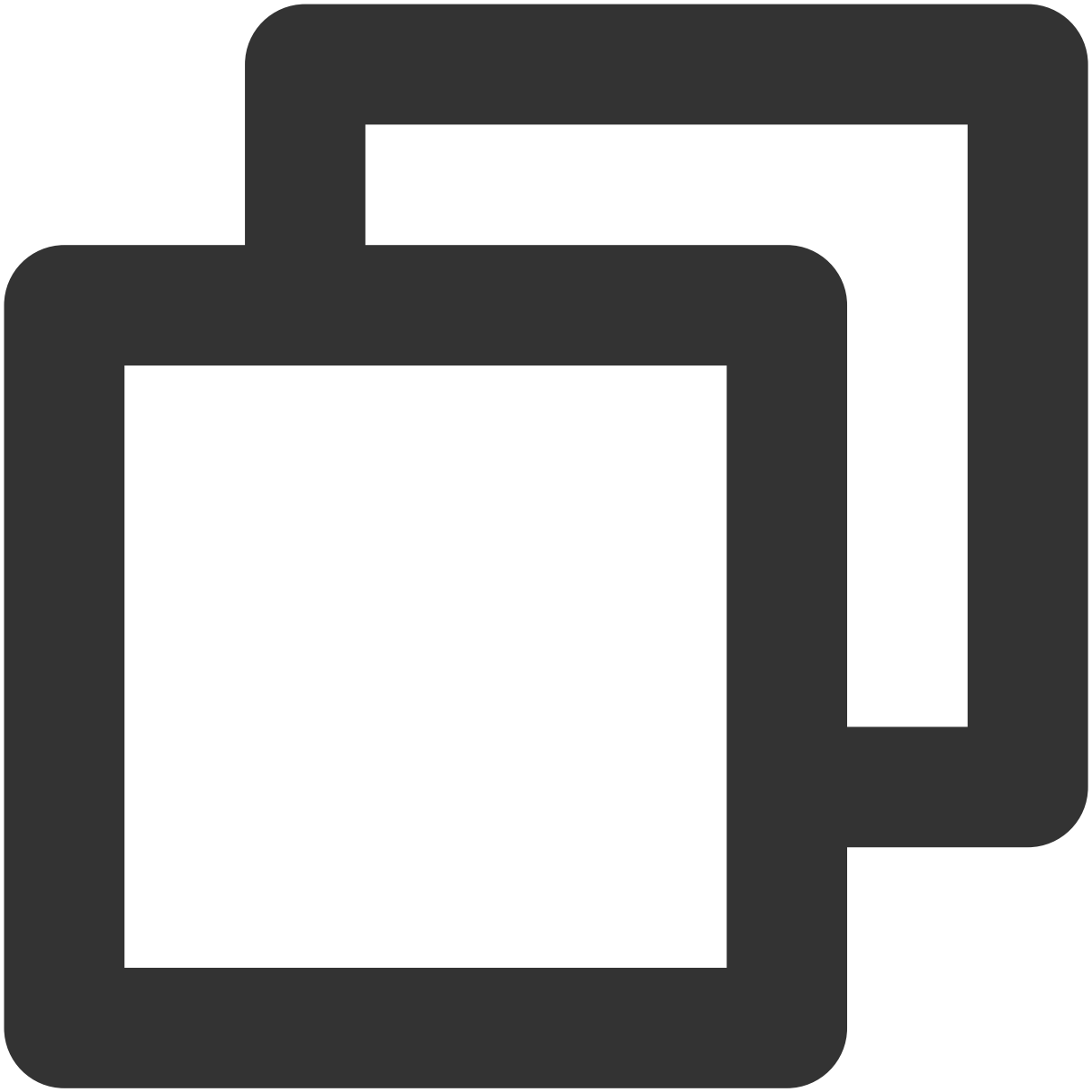
示例的拼接结果为：



```
GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Lim
```

4. 计算签名（伪代码）

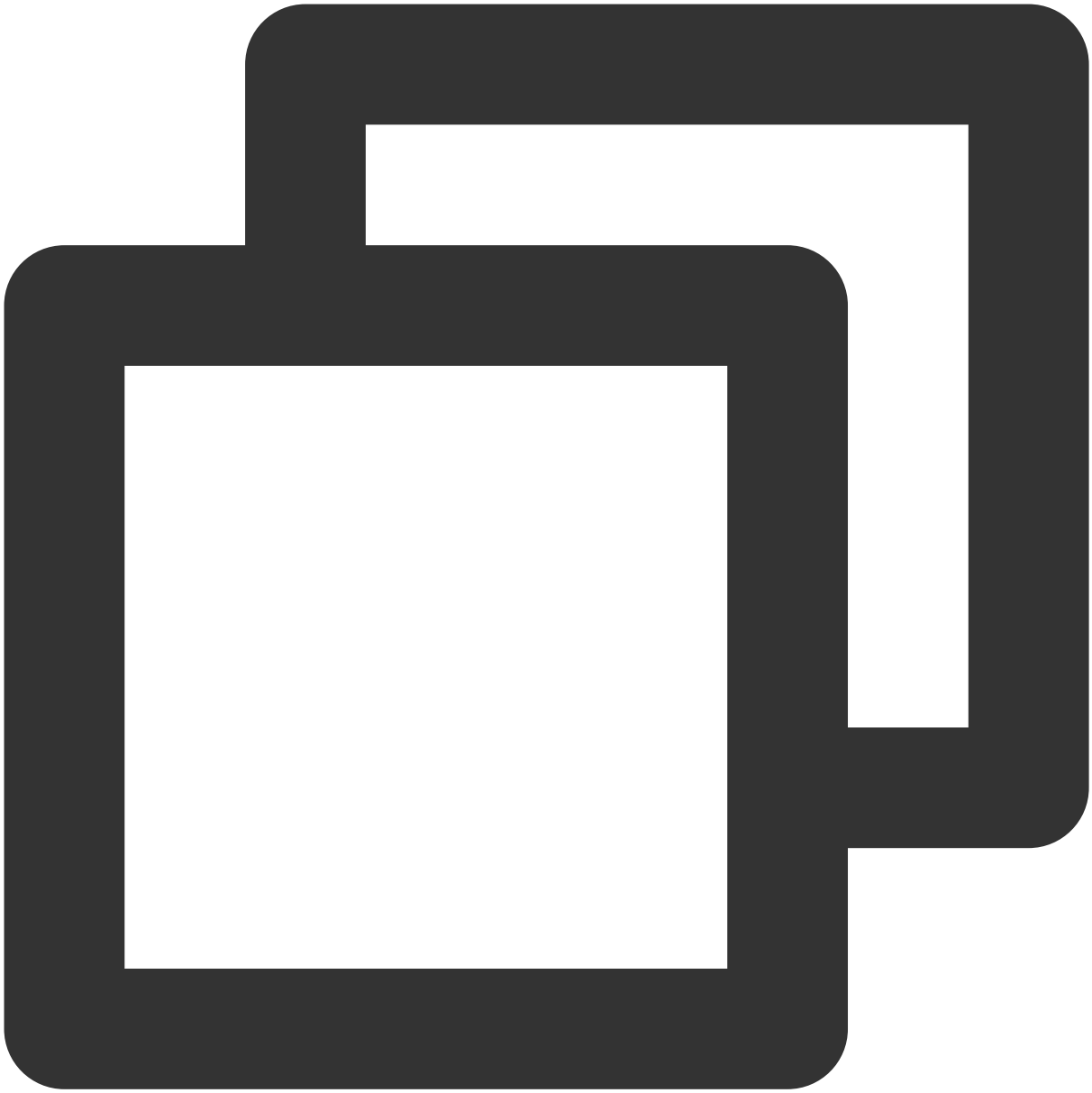
此步骤生成签名串。 首先使用 HMAC-SHA1 算法对上一步中获得的**签名原文字符串**进行签名，然后将生成的签名串使用 Base64 进行编码，即可获得最终的签名串。



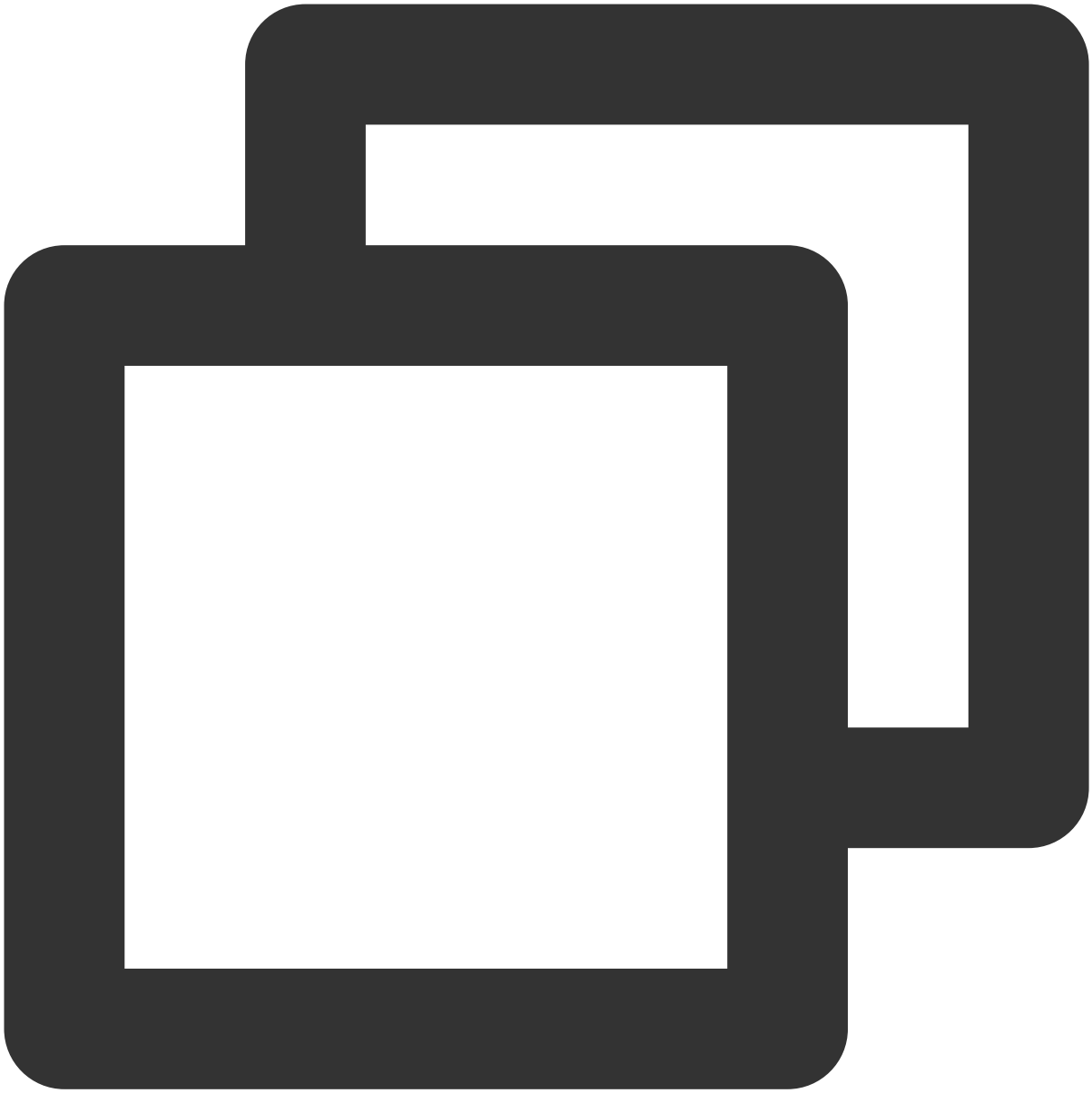
```
import javax.crypto.Mac;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.codec.binary.Base64;

/**
 *
 * [Hmac-SHA1 签名算法]
 * @String  encryptText [加密字符源串]
 * @String  encryptKey  [加密密钥]
 * @return [签名值]
```

```
 */
public class HmacSHA1 {
    private static final String MAC_NAME = "HmacSHA1";
    public static final String ENCODING = "UTF-8";
    //签名方法
    public static String HmacSHA1Encrypt(String s,String secret_key ) throws Except
        byte[] data = encryptKey.getBytes( ENCODING );
        SecretKey secretKey = new SecretKeySpec( data, MAC_NAME );
        Mac mac = Mac.getInstance( MAC_NAME );
        mac.init( secretKey );
        byte[] text = encryptText.getBytes( ENCODING );
        byte[] digest = mac.doFinal( text );
        return new String(Base64.encodeBase64(digest));
    }
}
String secret_key = "Gu5t9xGAR**********EXAMPLE";
String s = "GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-
//最终签名串
String Signature = new HmacSHA1();
System.out.println(Signature)
```

5. 获取调用信息并发送请求



```
# 实际调用，成功后可能如果是消费接口会产生计费(此处以Python语言为例发送get请求)
resp = requests.get("https://" + endpoint, params=data)
print(resp.url)
```



最终得到的请求串为
`https://cvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96d`

字段名称	解释
endpoint	服务地址， 例如： `cvm.tencentcloudapi.com`
data	API 3.0 签名 V1 所举示例接口参数，**注意** 这里需要将计算的签名已键值对的形式加入data中

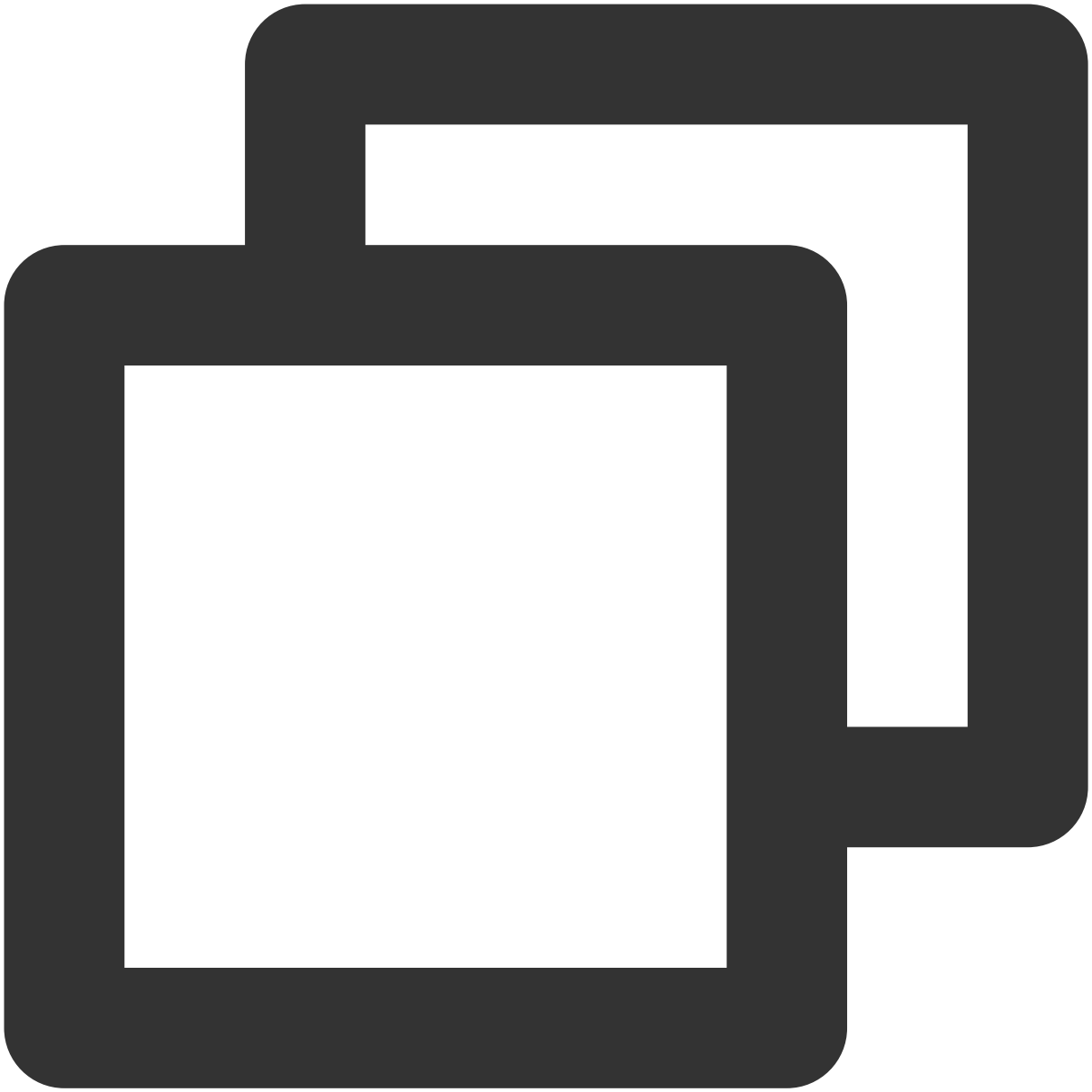
注意：

由于示例中的密钥是虚构的，时间戳也不是系统当前时间，因此如果将此 url 在浏览器中打开或者用 curl 等命令调用时会返回鉴权错误：签名过期。为了得到一个可以正常返回的 url ，需要修改示例中的 SecretId 和 SecretKey 为真实的密钥，并使用系统当前时间戳作为 Timestamp 。
为了更清楚的解释签名过程，下面以 Java 语言为例，将上述的签名过程具体实现。请求的域名、调用的接口和参数的取值都以上述签名过程为准，代码只为解释签名过程，并不具备通用性，实际开发请尽量使用 SDK 。

6. 签名串编码

生成的签名串并不能直接作为请求参数，需要对其进行 URL 编码。
如上一步生成的签名串为 `Eli****************cGeI=` ，最终得到的签名串请求参数（Signature）为： `EliP**********************eI%3d` ，它将用于生成最终的请求 URL。
注意：
如果用户的请求方法是 GET，或者请求方法为 POST 同时 Content-Type 为 application/x-www-form-urlencoded，则发送请求时所有请求参数的值均需要做 URL 编码，参数键和=符号不需要编码。非 ASCII 字符在 URL 编码前需要先以 **UTF-8** 进行编码。
有些编程语言的网络库会自动为所有参数进行 urlencode，在这种情况下，就不需要对签名串进行 URL 编码了，否则两次 URL 编码会导致签名失败。
其他参数值也需要进行编码，编码采用 RFC 3986。使用 `%XY` 对特殊字符例如汉字进行百分比编码，其中“ X ”和“ Y ”为十六进制字符（0-9 和大写字母 A-F），使用小写将引发错误。

7. API 3.0 签名 V1示例



```
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.Random;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

public class TencentCloudAPIDemo {
    private final static String CHARSET = "UTF-8";
```

```
    public static String sign(String s, String key, String method) throws Exception
        Mac mac = Mac.getInstance(method);
        SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(CHARSET), mac.
        mac.init(secretKeySpec);
        byte[] hash = mac.doFinal(s.getBytes(CHARSET));
        return DatatypeConverter.printBase64Binary(hash);
    }

    public static String getStringToSign(TreeMap<String, Object> params) {
        StringBuilder s2s = new StringBuilder("GETcvm.tencentcloudapi.com/?");
        // 签名时要求对参数进行字典排序，此处用TreeMap保证顺序
        for (String k : params.keySet()) {
            s2s.append(k).append("=").append(params.get(k).toString()).append("&");
        }
        return s2s.toString().substring(0, s2s.length() - 1);
    }

    public static String getUrl(TreeMap<String, Object> params) throws UnsupportedE
        StringBuilder url = new StringBuilder("https://cvm.tencentcloudapi.com/?");
        // 实际请求的url中对参数顺序没有要求
        for (String k : params.keySet()) {
            // 需要对请求串进行urlencode，由于key都是英文字母，故此处仅对其value进行urlencoc
            url.append(k).append("=").append(URLEncoder.encode(params.get(k).toStri
        }
        return url.toString().substring(0, url.length() - 1);
    }

    public static void main(String[] args) throws Exception {
        TreeMap<String, Object> params = new TreeMap<String, Object>(); // TreeMap可
        // 实际调用时应当使用随机数，例如：params.put("Nonce", new Random().nextInt(java
        params.put("Nonce", 11886); // 公共参数
        // 实际调用时应当使用系统当前时间，例如：   params.put("Timestamp", System.current
        params.put("Timestamp", 1465185768); // 公共参数
        params.put("SecretId", "AKIDz8krbsJ5**********mLPx3EXAMPL"); // 公共参数
        params.put("Action", "DescribeInstances"); // 公共参数
        params.put("Version", "2017-03-12"); // 公共参数
        params.put("Region", "ap-guangzhou"); // 公共参数
        params.put("Limit", 20); // 业务参数
        params.put("Offset", 0); // 业务参数
        params.put("InstanceIds.0", "ins-09dx96dg"); // 业务参数
        params.put("Signature", sign(getStringToSign(params), "Gu5t9xGAR***********
        System.out.println(getUrl(params));
    }
}
```

API 2.0 签名

此签名现已不在维护，建议使用性能更优的 **API 3.0 签名**，如需使用，请直接在 API Explorer 的 **签名串生成 > 选择 API 2.0** 签名版本进行操作。

签名失败

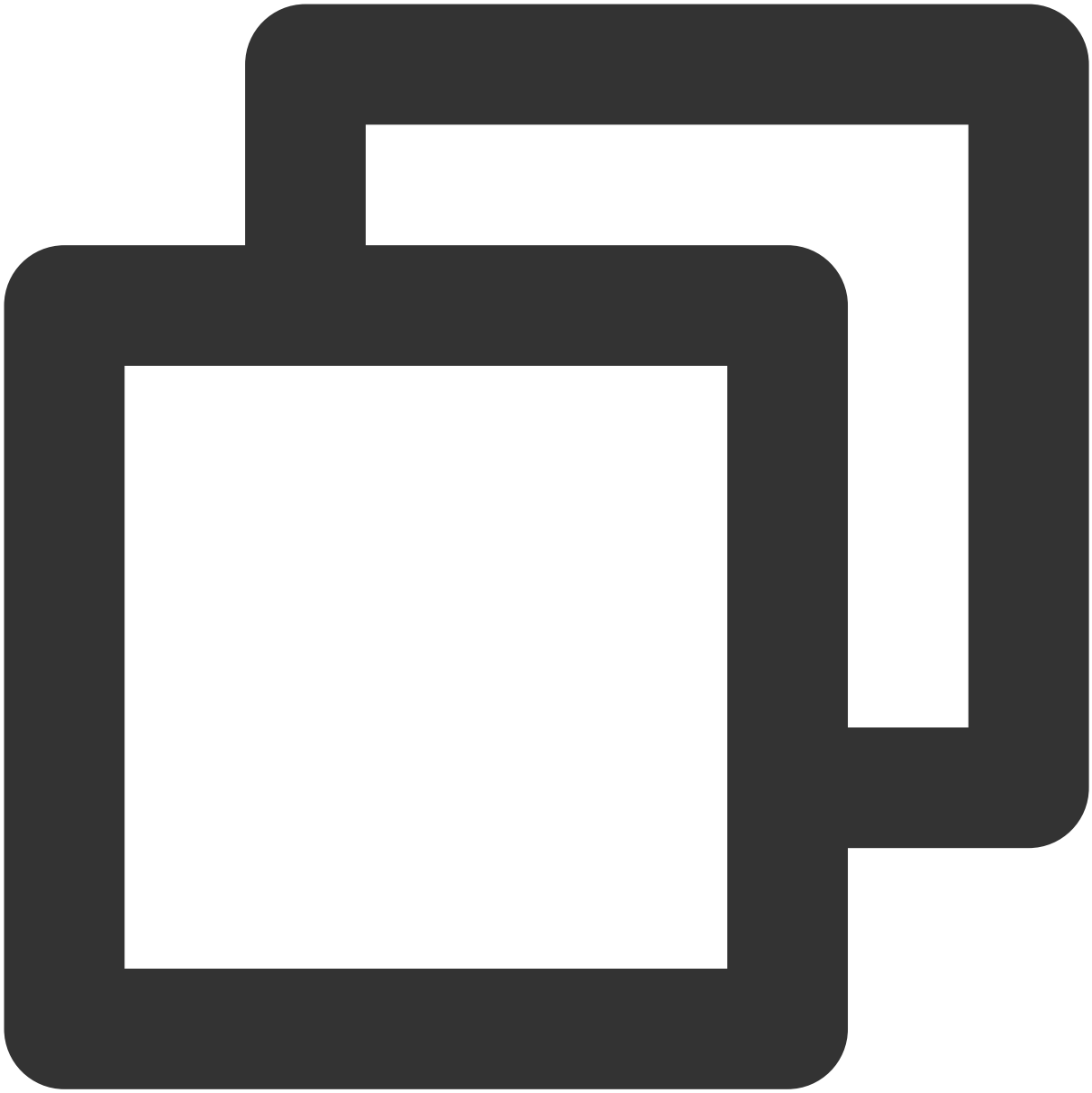
存在以下签名失败的错误码，请根据实际情况处理。

错误码	错误描述
AuthFailure.SignatureExpire	签名过期。Timestamp 与服务器接收到请求的时间相差不得超过五分钟。
AuthFailure.SecretIdNotFound	密钥不存在。请到控制台查看密钥是否被禁用，是否少复制了字符或者多了字符。
AuthFailure.SignatureFailure	签名错误。可能是签名计算错误，或者签名与实际发送的内容不相符合，也有可能是密钥 SecretKey 错误导致的。
AuthFailure.TokenFailure	临时证书 Token 错误。
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。

返回结果

正确返回结果

以云服务器的接口查看实例状态列表（ DescribeInstancesStatus ）2017-03-12版本为例，若调用成功，其可能的返回如下为：



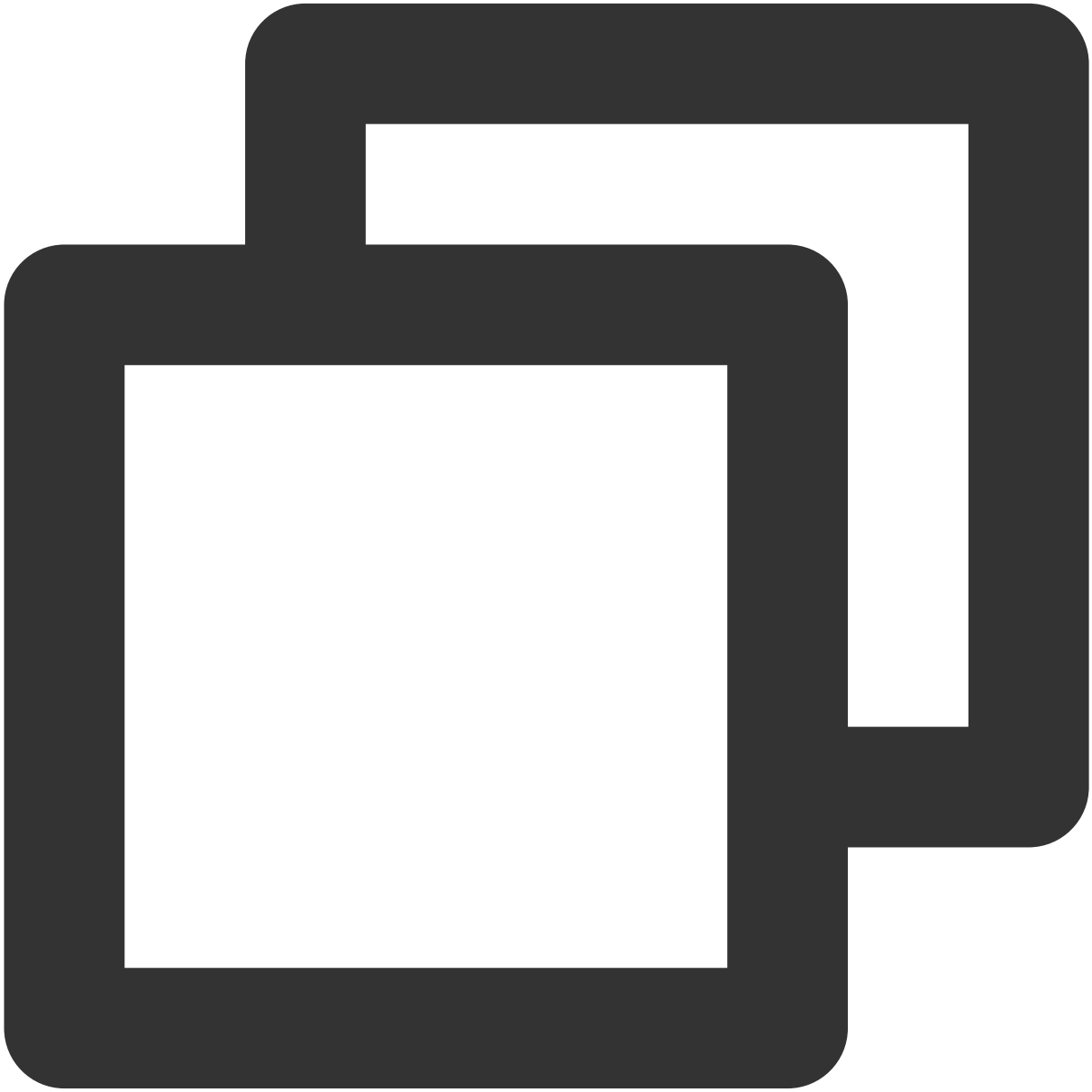
```
{
    "Response": {
        "TotalCount": 0,
        "InstanceStatusSet": [],
        "RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
    }
}
```

`Response` 及其内部的 `RequestId` 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 `TotalCount` 和 `InstanceStatusSet` 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 `TotalCount` 在此情况下的返回值为 0， `InstanceStatusSet` 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：



```
{
    "Response": {
        "Error": {
```

```
            "Code": "AuthFailure.SignatureFailure",
            "Message": "The provided credentials could not be validated. Please che
        },
        "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
    }
}
```

`Error` 的出现代表着该请求调用失败。 `Error` 字段连同其内部的 `Code` 和 `Message` 字段在调用失败时是必定返回的。

`Code` 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。

`Message` 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。

`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。 Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。

LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	HTTPS 请求方法错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

版权所有：腾讯云计算（北京）有限责任公司

Node.js API

最近更新时间：2023-03-07 18:16:40

腾讯云 API 全新升级3.0，该版本进行了性能优化且全地域部署、支持就近和按地域接入、访问时延下降显著，接口描述更加详细、错误码描述更加全面、SDK 增加接口级注释，让您更加方便快捷的使用腾讯云产品。这里针对 Node.js API 调用方式进行简单说明。
现已支持云服务器（CVM）、云硬盘（CBS）、私有网络（VPC）、云数据库（TencentDB）等 腾讯云产品，后续会支持其他的云产品接入，敬请期待。

了解请求结构

1. 服务地址（endpoint）

API 支持就近地域接入（例如：cvm 产品域名为 `cvm.tencentcloudapi.com` ），也支持指定地域域名访问（例如：广州地域的域名为 `cvm.ap-guangzhou.tencentcloudapi.com` ），各地域参数见下文公共参数中的地域列表，详情请参见各产品的“请求结构”文档说明判断是否支持该地域。
注意：
对时延敏感的业务，建议指定带地域的域名。

2. 通信协议

腾讯云 API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法：
POST（推荐）
GET
POST 请求支持的 Content-Type 类型：
application/json（推荐），必须使用签名方法 v3（TC3-HMAC-SHA256）。
application/x-www-form-urlencoded，必须使用签名方法 v1（HmacSHA1 或 HmacSHA256）。
multipart/form-data（仅部分接口支持），必须使用签名方法 v3（TC3-HMAC-SHA256）。
GET 请求的请求包大小不得超过32KB。POST 请求使用签名方法 v1（HmacSHA1、HmacSHA256）时不得超过1MB。POST 请求使用签名方法 v3（TC3-HMAC-SHA256）时支持10MB。

4. 字符编码

均使用 UTF-8 编码。

版权所有：腾讯云计算（北京）有限责任公司

公共参数

-公共参数是用于标识用户和接口签名的参数，每次请求均需要携带这些参数，才能正常发起请求。

签名方法 V3 公共参数

签名方法 v3 （有时也称作 TC3-HMAC-SHA256）相比签名方法 v1 （有些文档可能会简称签名方法），更安全，支持更大的请求包，支持 POST JSON 格式，性能有一定提升，推荐使用该签名方法计算签名。使用方法见下文 “签名方法介绍” 。

参数名称	类型	必选	描述
X-TC-Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
X-TC-Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702。**注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。**
X-TC-Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本2017-03-12。
Authorization	String	是	HTTP 标准身份认证头部字段，例如： TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=72e494ea8**********a96525168 其中： TC3-HMAC-SHA256：签名方法，目前固定取该值。 Credential：签名凭证，AKIDEXAMPLE 是 SecretId。 Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致。 service 为产品名，通常为域名前缀，例如域名 cvm.tencentcloudapi.com 意味着产品名是 cvm。本产品取值为 cvm。 SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部。 Signature：签名摘要，计算过程详见下文。
X-TC-Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

签名方法 V1 公共参数

使用签名方法 v1 （有时会称作 HmacSHA256 和 HmacSHA1），公共参数需要统一放到请求串中。

参数名称	类型	必选	描述
Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如 1529223702，如果与当前时间相差过大，会引起签名过期错误。
Nonce	Integer	是	随机正整数，与 Timestamp 联合起来，用于防止重放攻击。
SecretId	String	是	在 云API密钥 上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey，而 SecretKey 会用来生成请求签名 Signature。
Signature	String	是	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。具体计算方法参见下文“签名方法介绍” 。
Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
SignatureMethod	String	否	签名方式，目前支持 HmacSHA256 和 HmacSHA1。只有指定此参数为 HmacSHA256 时，才使用 HmacSHA256 算法验证签名，其他情况均使用 HmacSHA1 验证签名。
Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token 。

地域列表

由于各个产品支持地域不同，具体详情请参考各产品文档中的地域列表。
例如，您可以参考云服务器的 地域列表。

Node.js API 调用方式

腾讯云 API 会对每个请求进行身份验证，用户需要使用安全凭证，经过特定的步骤对请求进行签名（Signature），每个请求都需要在公共请求参数中指定该签名结果并以指定的方式和格式发送请求。

假设用户的 SecretId 和 SecretKey 分别是： `AKIDz8krbsJ5**********mLPx3EXAMPLE` 和 `Gu5t9xGAR***********EXAMPLE` 。用户想查看广州区云服务器名为“未命名”的主机状态，只返回一条数据。则请求可能为：

```
curl -X POST https://cvm.tencentcloudapi.com \\
-H "Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/20
-H "Content-Type: application/json; charset=utf-8" \\
-H "Host: cvm.tencentcloudapi.com" \\
```

```
  -H "X-TC-Action: DescribeInstances" \\
  -H "X-TC-Timestamp: 1551113065" \\
  -H "X-TC-Version: 2017-03-12" \\
  -H "X-TC-Region: ap-guangzhou" \\
  -d '{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instanc
```

步骤1：申请安全凭证

本文使用的安全凭证为密钥，密钥包括 SecretId 和 SecretKey。每个用户最多可以拥有两对密钥。

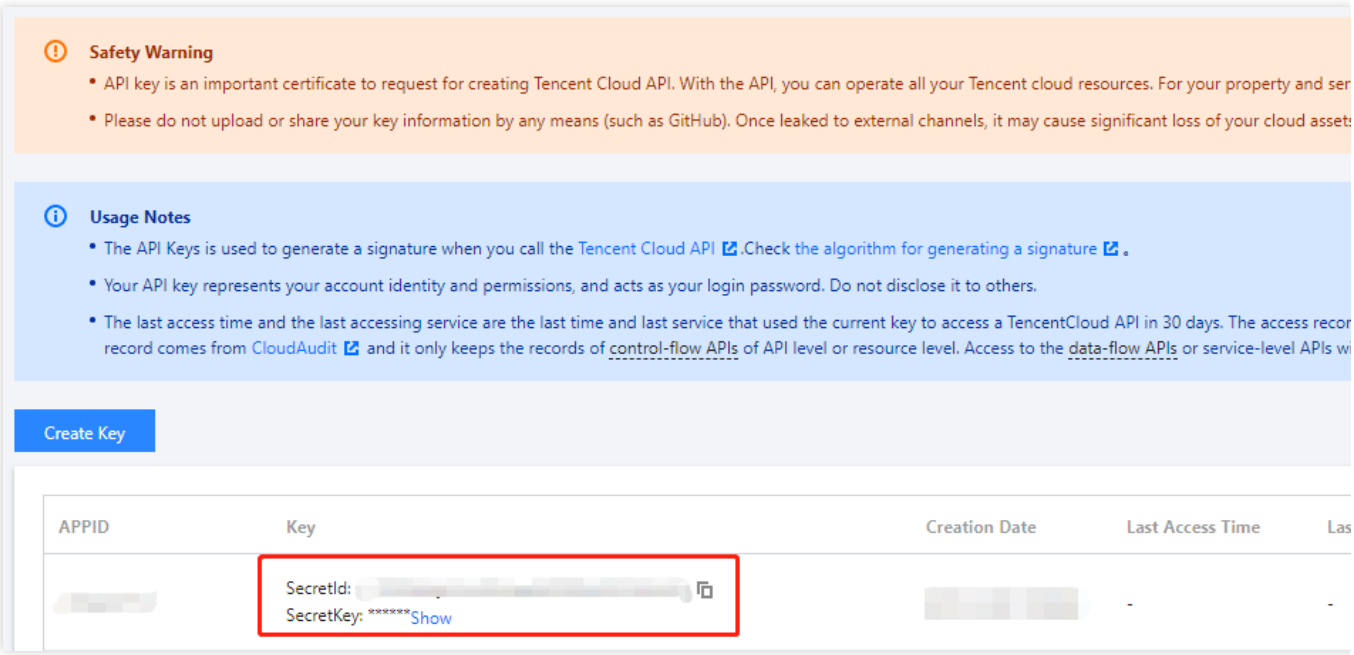
SecretId：用于标识 API 调用者身份，可以简单类比为用户名。

SecretKey：用于验证 API 调用者的身份，可以简单类比为密码。

注意：

用户必须严格保管安全凭证，避免泄露，否则将危及财产安全。如已泄漏，请立刻禁用该安全凭证。

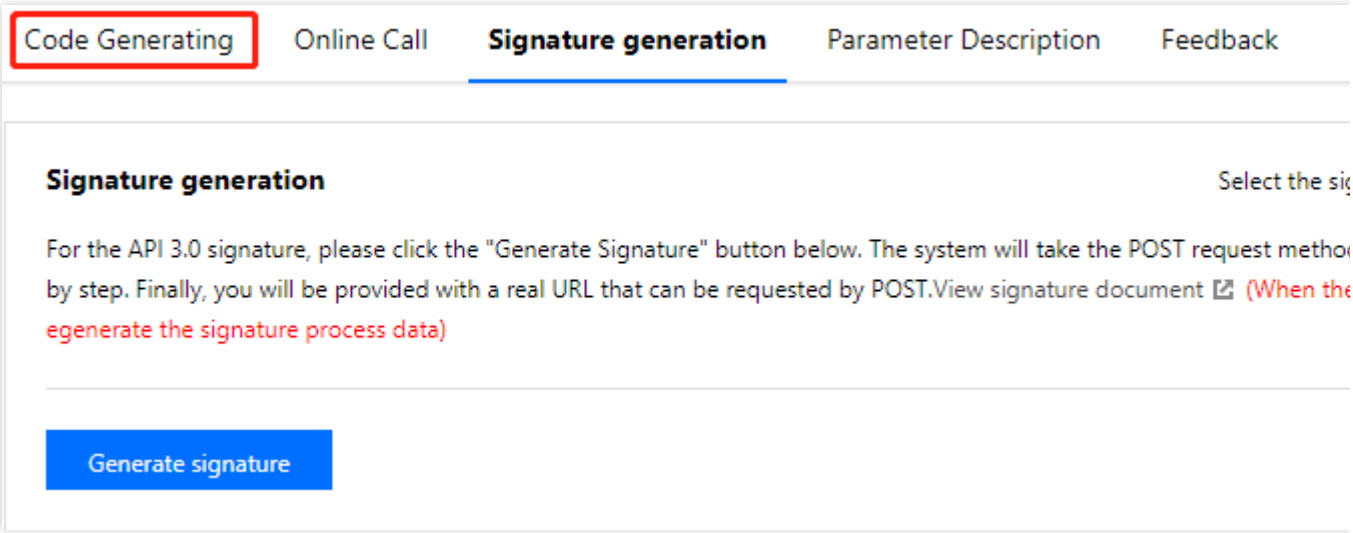
前往 API密钥管理 页面，即可进行获取。如下图所示：



步骤2：

1. 获取 API 3.0 V3 版本签名

签名方法 v3（TC3-HMAC-SHA256）功能上覆盖了以前的签名方法 v1，而且更安全，支持更大的请求，支持 json 格式，性能有一定提升，推荐使用该签名方法计算签名。



注意：

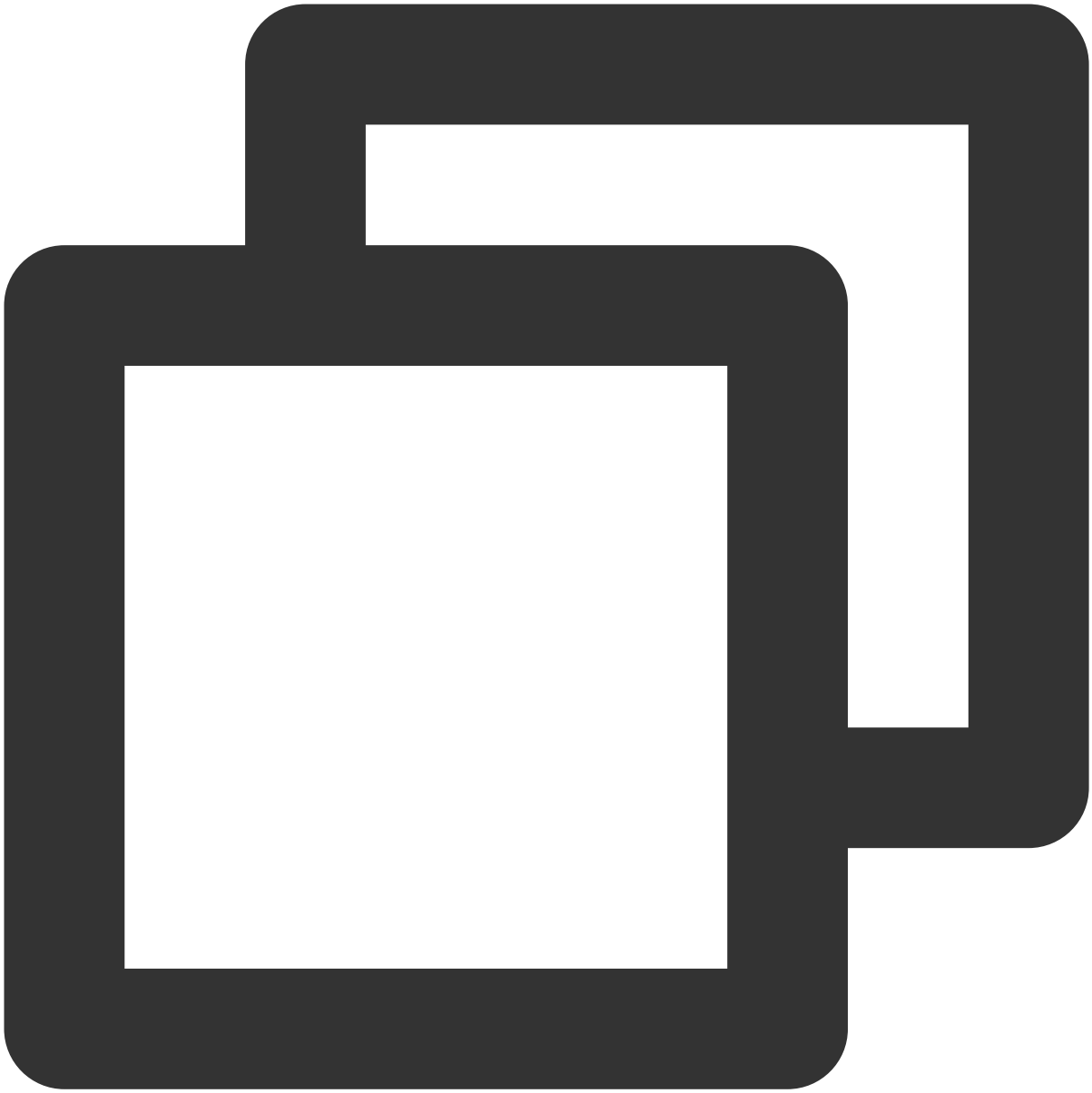
首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“ API 3.0 签名 v3 ”，可以生成签名过程进行验证，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的7种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

云 API 支持 GET 和 POST 请求。对于 GET 方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。对于 POST 方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式绝大多数接口均支持，multipart 格式只有特定接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。推荐使用 POST 请求，因为两者的结果并无差异，但 GET 请求只支持32KB以内的请求包。

下面以 查看实例列表 接口为例，分步骤介绍签名的计算过程。我们选择该接口是因为：

1. 云服务器默认已开通，该接口很常用；
2. 该接口是只读的，不会改变现有资源的状态；
3. 接口覆盖的参数种类较全，可以演示包含数据结构的数组如何使用。

1. 拼接规范请求串

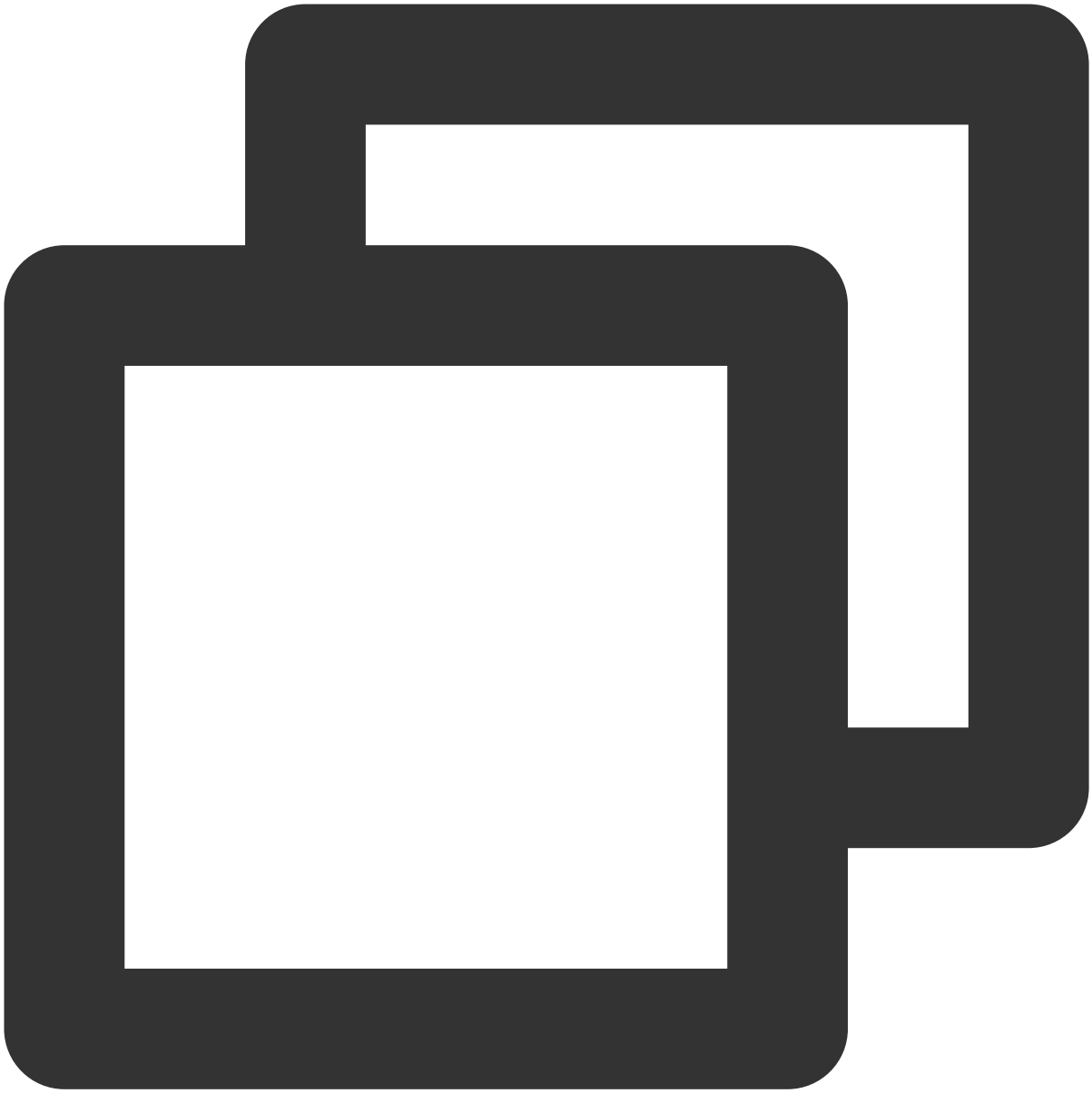


```
CanonicalRequest =
    HTTPRequestMethod + '\\n' +
    CanonicalURI + '\\n' +
    CanonicalQueryString + '\\n' +
    CanonicalHeaders + '\\n' +
    SignedHeaders + '\\n' +
    HashedRequestPayload
```

字段名称	解释

HTTPRequestMethod	HTTP 请求方法（GET、POST ）。此示例取值为 `POST` 。
CanonicalURI	URI 参数，API 3.0 固定为正斜杠（/）。
CanonicalQueryString	发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串""，对于 G 求，则为 URL 中问号（?）后面的字符串内容，例如：Limit=10&Offset=0。 注意：CanonicalQueryString 需要参考 RFC3986 进行 URLEncode，字符集 UTF8，推荐使用 语言标准库，所有特殊字符均需编码，大写形式。
CanonicalHeaders	参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部 签名以提高自身请求的唯一性和安全性。 拼接规则：头部 key 和 value 统一转成小写，掉首尾空格，按照 key:value\\n 格式拼接；多个头部，按照头部 key（小写）的 ASCII 行拼接。此示例计算结果是 `content-type:application/json; charset=ut 8\\nhost:cvm.tencentcloudapi.com\\n` 。 注意：content-type 必须和实际发 符合，有些编程语言网络库即使未指定也会自动添加 charset 值，如果签名时和发送时 致，服务器会返回签名校验失败。
SignedHeaders	参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包 部内容是一一对应的。content-type 和 host 为必选头部。 拼接规则：头部 key 统一转 写；多个头部 key（小写）按照 ASCII 升序进行拼接，并且以分号（;）分隔。此示例 content-type;host
HashedRequestPayload	请求正文（Requestpayload，即 body，此示例为 `{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-name"` 的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(RequestPayload))), HTTP 请求正文做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。 GET 请求，RequestPayload 固定为空字符串。此示例计算结果是 `35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f`

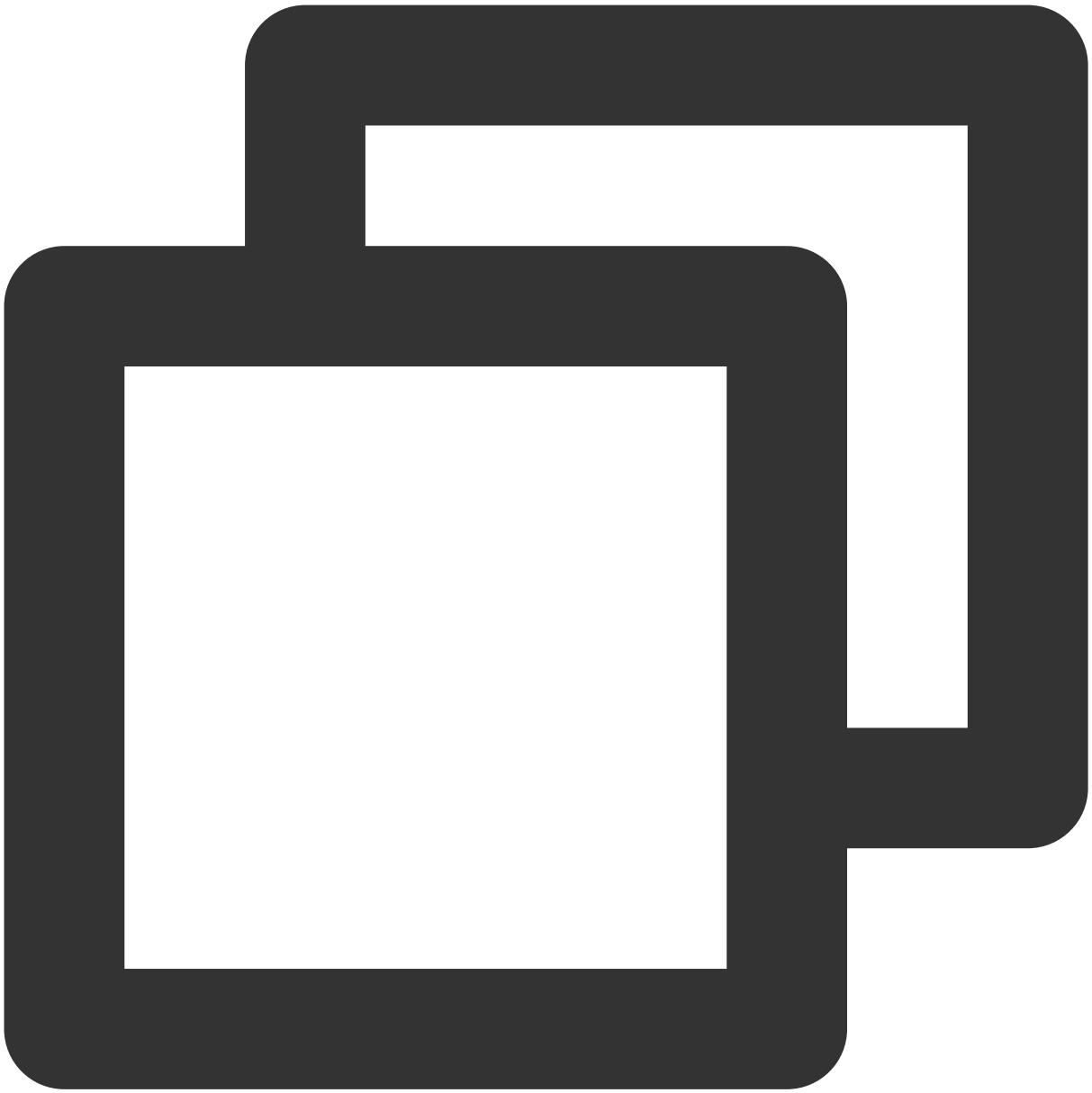
根据以上规则，示例中得到的规范请求串如下：



```
POST
/

content-type:application/json; charset=utf-8
host:cvm.tencentcloudapi.com

content-type;host
35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f064
```

2. 拼接待签名字符串

按如下格式拼接待签名字符串：



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

字段名称	解释
Algorithm	签名算法，目前固定为 `TC3-HMAC-SHA256` 。

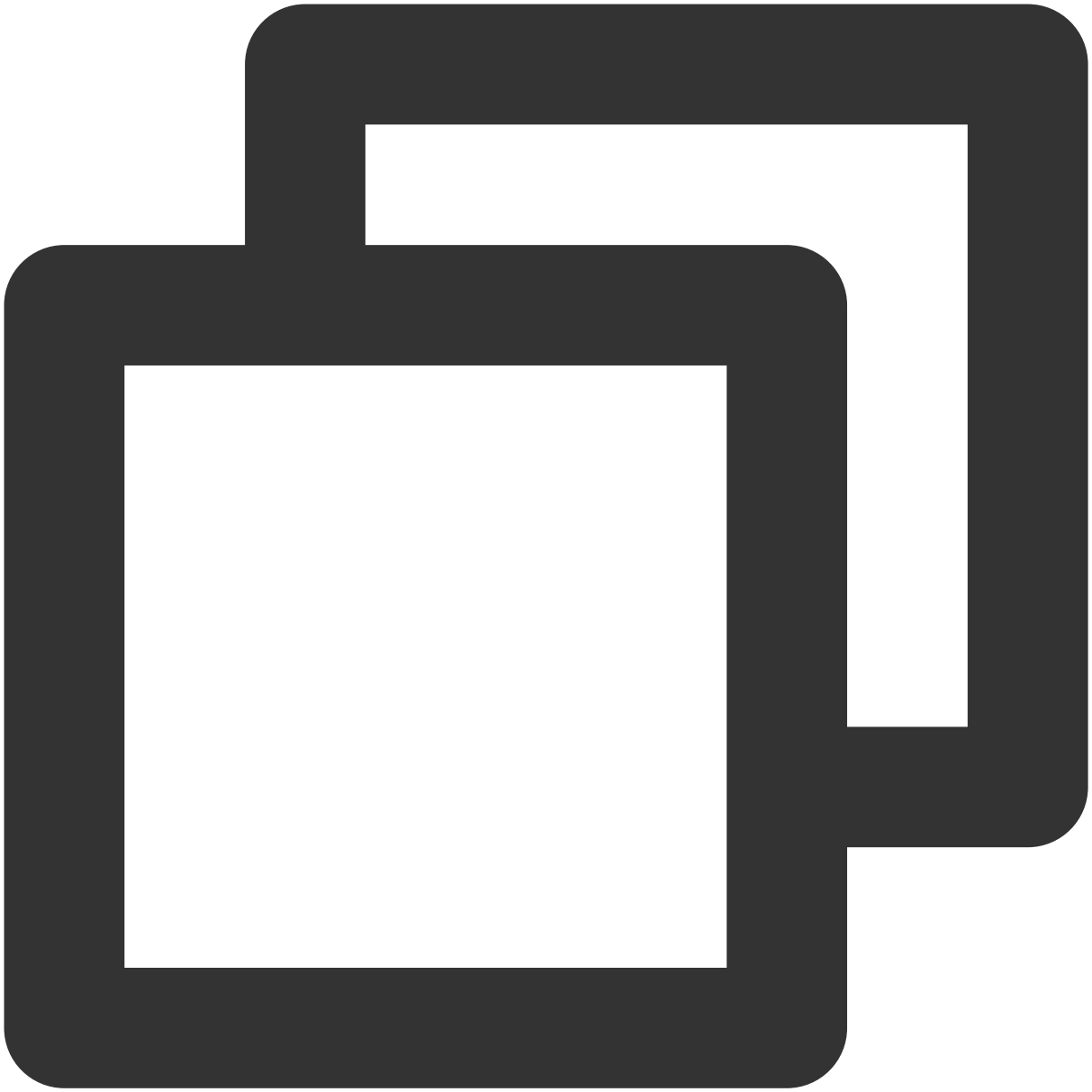
RequestTimestamp	请求时间戳，即请求头部的公共参数 X-TC-Timestamp 取值，取当前时间 UNIX 时间 确到秒。此示例取值为 `1551113065` 。
CredentialScope	凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符日 （tc3_request）。**Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Time 换算的 UTC 标准时间日期一致**；service 为产品名，必须与调用的产品域名一致。此 算结果是 2019-02-25/cvm/tc3_request。
HashedCanonicalRequest	前述步骤拼接所得规范请求串的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))。此示例计算结果是 `5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d`

注意：

Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为1551113065，在东八区的时间是2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为2019-02-25，而不是2019-02-26。

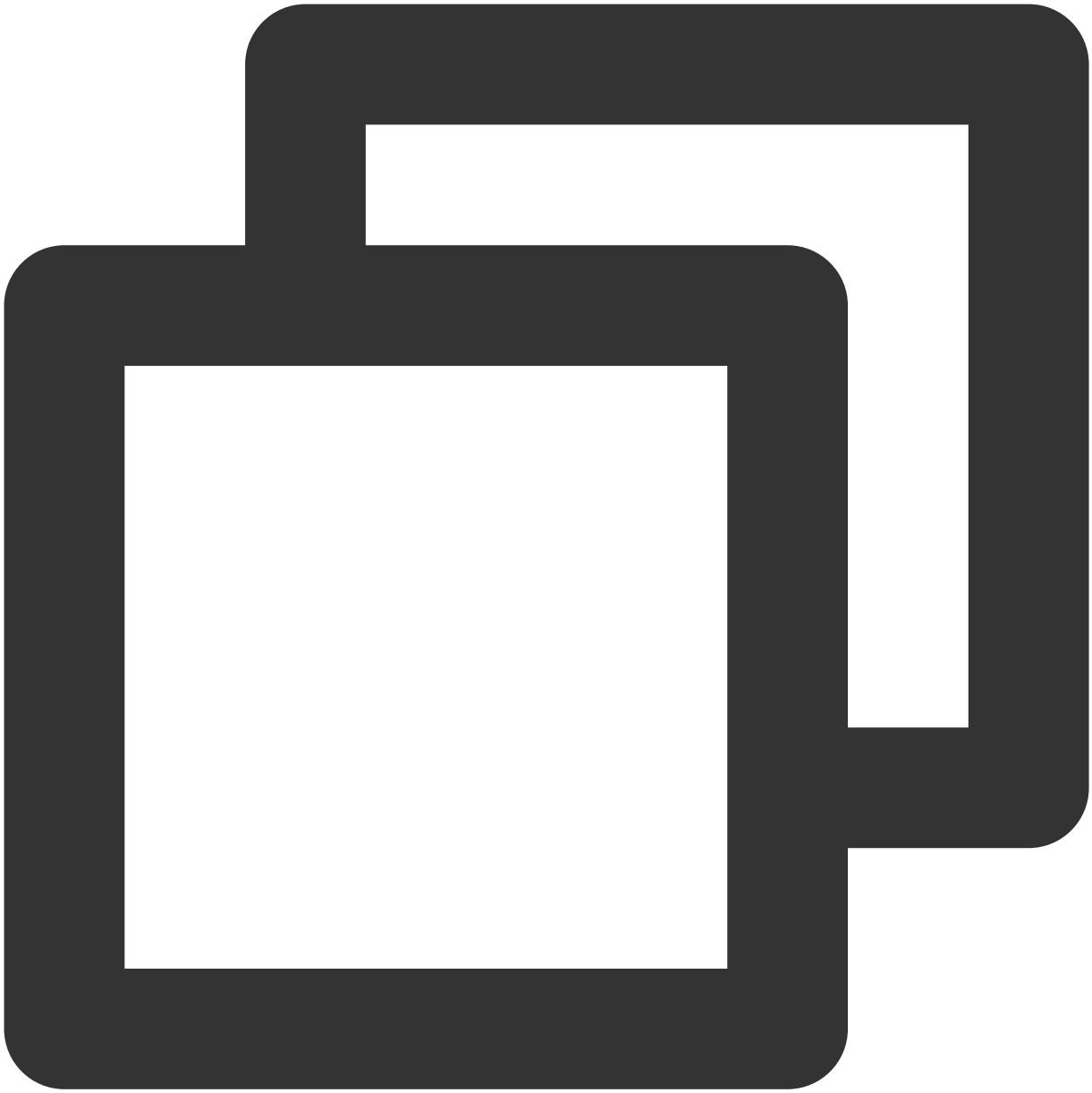
Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败，返回签名过期错误。

根据以上规则，示例中得到的待签名字符串如下：



```
TC3-HMAC-SHA256
1551113065
2019-02-25/cvm/tc3_request
5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d7031
```

3.计算签名(伪代码)



```
const kDate = sha256(date, 'TC3' + SECRET_KEY)
const kService = sha256(service, kDate)
const kSigning = sha256('tc3_request', kService)
const signature = sha256(stringToSign, kSigning, 'hex')
```

派生出的密钥 `SecretDate` 、 `SecretService` 和 `SecretSigning` 是二进制的数据，可能包含不可打印字符，此处不展示中间结果。

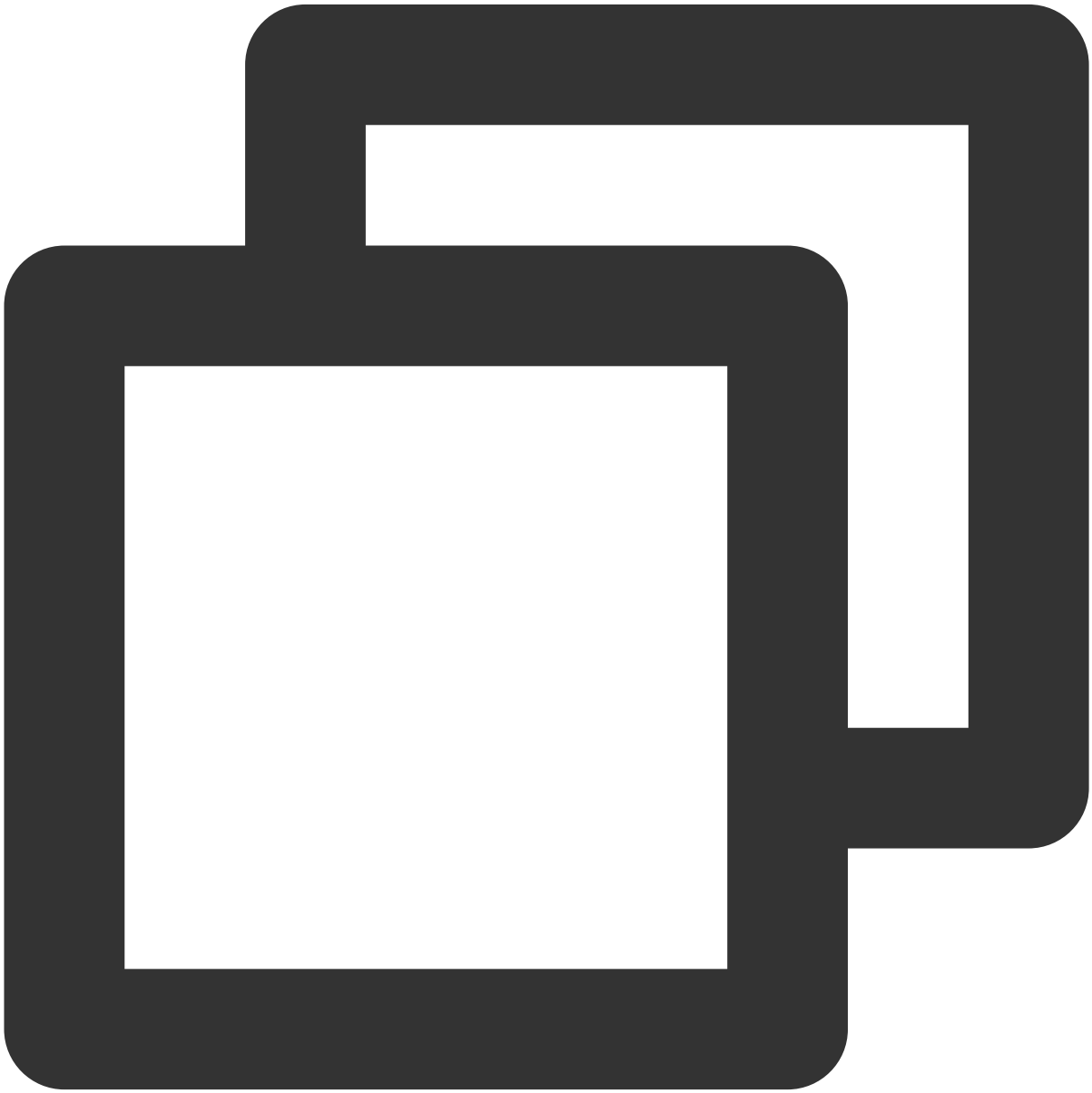
字段名称	解释
secretKey	原始的 SecretKey，即 `Gu5t9xGAR**********EXAMPLE` 。

date	即 Credential 中的 Date 字段信息。此示例取值为 `2019-02-25` 。
service	即 CredentialScope 中的 Service 字段信息。此示例取值为 `cvm` 。

此示例计算结果是 `72e494ea8*****************************************a96525168` 。

4. 拼接 Authorization

按如下格式拼接 Authorization：

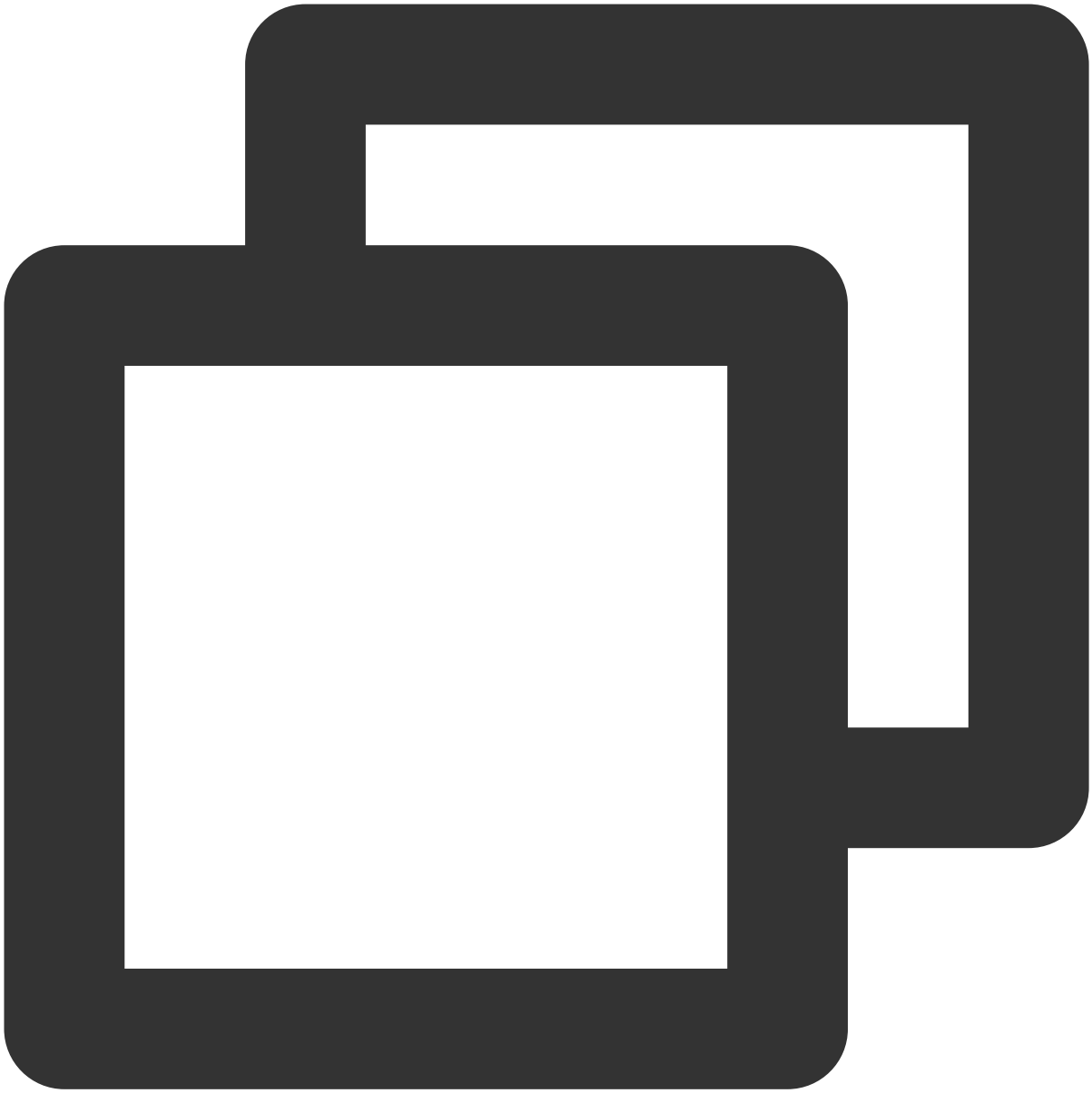


```
Authorization =
```

```
    Algorithm + ' ' +
    'Credential=' + SecretId + '/' + CredentialScope + ', ' +
    'SignedHeaders=' + SignedHeaders + ', ' +
    'Signature=' + Signature
```

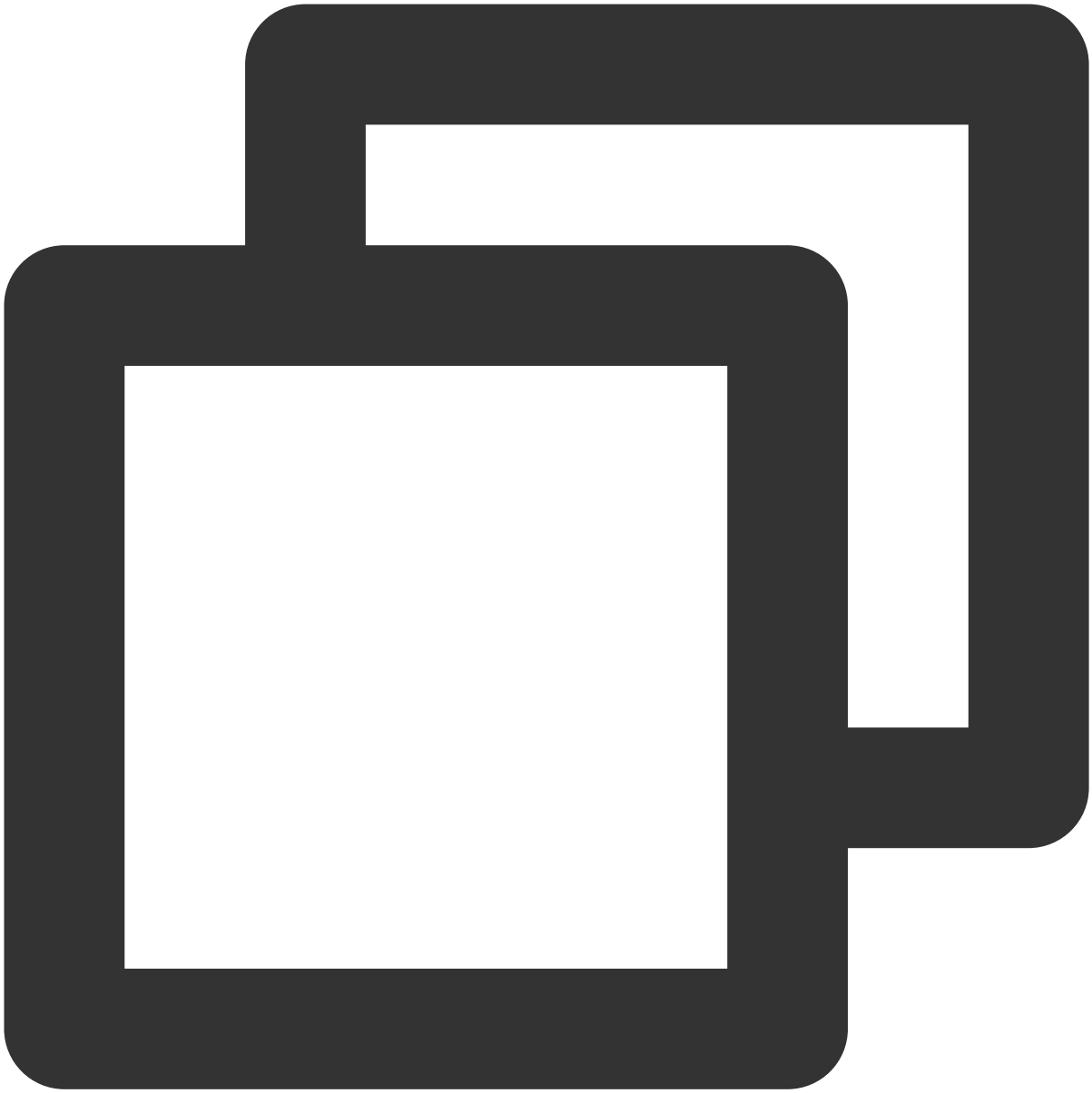
字段名称	解释
Algorithm	签名方法，固定为 `TC3-HMAC-SHA256` 。
SecretId	密钥对中的 SecretId，即 `AKIDz8krbsJ5**********mLPx3EXAMPLE` 。
CredentialScope	见上文，凭证范围。此示例计算结果是 `2019-02-25/cvm/tc3_request` 。
SignedHeaders	见上文，参与签名的头部信息。此示例取值为 `content-type;host` 。
Signature	签名值。此示例计算结果是 `72e494ea8****************************************a96525168` 。

根据以上规则，示例中得到的值为：



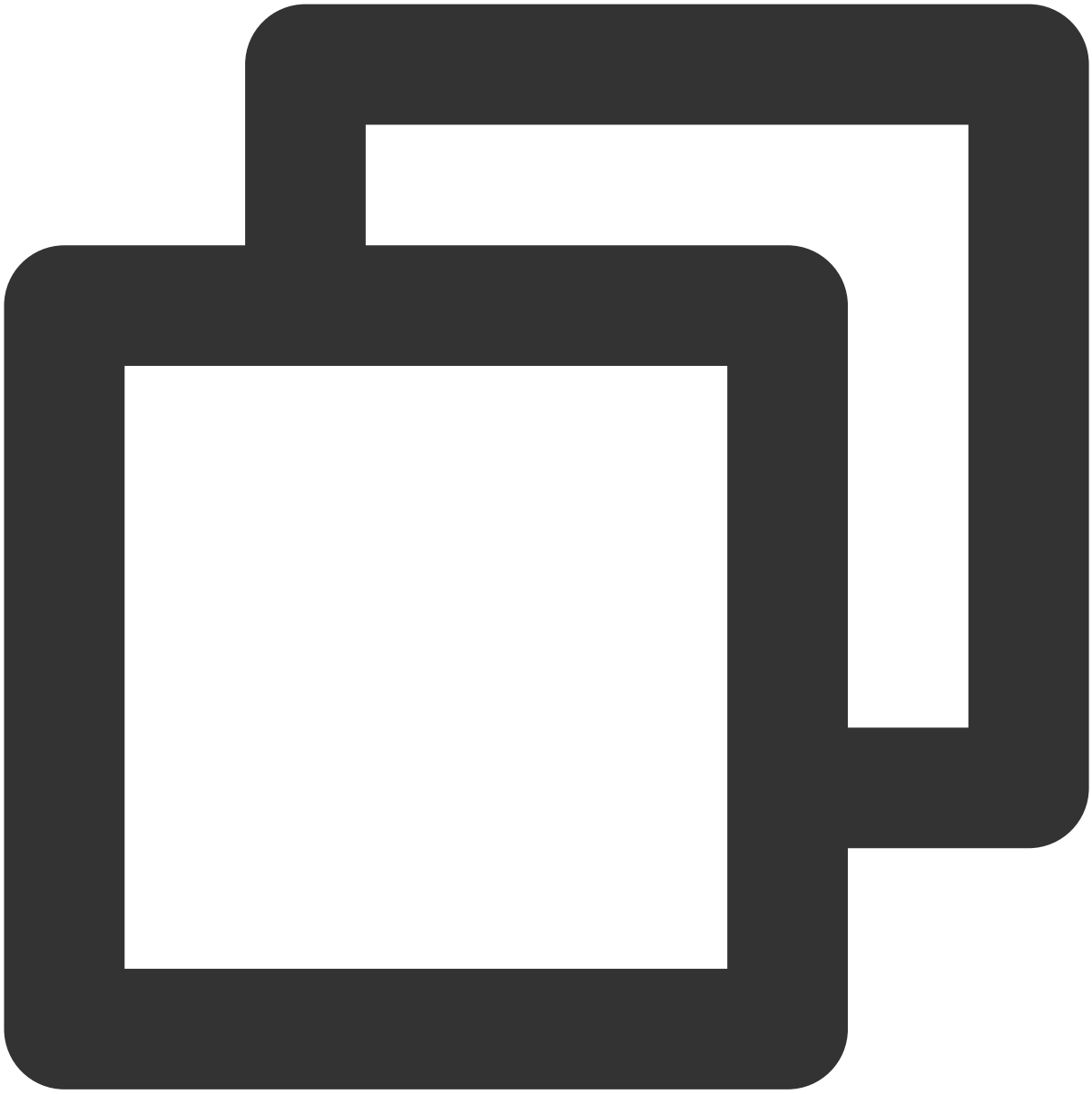
```
TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/2019-02-25/cvm/tc3_re
```

最终完整的调用信息如下：



```
POST https://cvm.tencentcloudapi.com/
Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/2019-0
Content-Type: application/json; charset=utf-8
Host: cvm.tencentcloudapi.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1551113065
X-TC-Region: ap-guangzhou

{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-na
```

5. API 3.0 签名 V3示例



```
const crypto = require('crypto');

function sha256(message, secret = '', encoding) {
    const hmac = crypto.createHmac('sha256', secret)
    return hmac.update(message).digest(encoding)
}
function getHash(message, encoding = 'hex') {
    const hash = crypto.createHash('sha256')
    return hash.update(message).digest(encoding)
```

```
}
function getDate(timestamp) {
    const date = new Date(timestamp * 1000)
    const year = date.getUTCFullYear()
    const month = ('0' + (date.getUTCMonth() + 1)).slice(-2)
    const day = ('0' + date.getUTCDate()).slice(-2)
    return `${year}-${month}-${day}`
}
function main(){
    // 密钥参数
    const SECRET_ID = "AKIDz8krbsJ5**********mLPx3EXAMPLE"
    const SECRET_KEY = "Gu5t9xGAR***********EXAMPLE"

    const endpoint = "cvm.tencentcloudapi.com"
    const service = "cvm"
    const region = "ap-guangzhou"
    const action = "DescribeInstances"
    const version = "2017-03-12"
    //const timestamp = getTime()
    const timestamp = 1551113065
    //时间处理, 获取世界时间日期
    const date = getDate(timestamp)

    // ************* 步骤 1：拼接规范请求串 *************
    const signedHeaders = "content-type;host"

    const payload = "{\\"Limit\\": 1, \\"Filters\\": [{\\"Values\\": [\\"\\\\u672a\

    const hashedRequestPayload = getHash(payload);
    const httpRequestMethod = "POST"
    const canonicalUri = "/"
    const canonicalQueryString = ""
    const canonicalHeaders = "content-type:application/json; charset=utf-8\\n" + "h

    const canonicalRequest = httpRequestMethod + "\\n"
                        + canonicalUri + "\\n"
                        + canonicalQueryString + "\\n"
                        + canonicalHeaders + "\\n"
                        + signedHeaders + "\\n"
                        + hashedRequestPayload
    console.log(canonicalRequest)
    console.log("----------------------------")

    // ************* 步骤 2：拼接待签名字符串 *************
    const algorithm = "TC3-HMAC-SHA256"
    const hashedCanonicalRequest = getHash(canonicalRequest);
    const credentialScope = date + "/" + service + "/" + "tc3_request"
```

```
    const stringToSign = algorithm + "\\n" +
                    timestamp + "\\n" +
                    credentialScope + "\\n" +
                    hashedCanonicalRequest
    console.log(stringToSign)
    console.log("----------------------------")

    // ************* 步骤 3：计算签名 *************
    const kDate = sha256(date, 'TC3' + SECRET_KEY)
    const kService = sha256(service, kDate)
    const kSigning = sha256('tc3_request', kService)
    const signature = sha256(stringToSign, kSigning, 'hex')
    console.log(signature)
    console.log("----------------------------")

    // ************* 步骤 4：拼接 Authorization *************
    const authorization = algorithm + " " +
                    "Credential=" + SECRET_ID + "/" + credentialScope + ", " +
                    "SignedHeaders=" + signedHeaders + ", " +
                    "Signature=" + signature
    console.log(authorization)
    console.log("----------------------------")

    const Call_Information = 'curl -X POST ' + "https://" + endpoint
                          + ' -H "Authorization: ' + authorization + '"'
                          + ' -H "Content-Type: application/json; charset=utf-8"'
                          + ' -H "Host: ' + endpoint + '"'
                          + ' -H "X-TC-Action: ' + action + '"'
                          + ' -H "X-TC-Timestamp: ' + timestamp.toString() + '"'
                          + ' -H "X-TC-Version: ' + version + '"'
                          + ' -H "X-TC-Region: ' + region + '"'
                          + " -d '" + payload + "'"
    console.log(Call_Information)
}
main()
```

2. 获取 API 3.0 V1 版本签名

签名方法 v1 简单易用，但是功能和安全性都不如签名方法 v3，推荐使用签名方法 v3。

注意：

首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“API 3.0 签名 v1”，可以生成签名过程进行验证，并提供了部分编程语言的签名示例，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的 7 种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

以云服务器查看实例列表（DescribeInstances）请求为例，当用户调用这一接口时，其请求参数可能如下:

参数名称	中文	参数值

Action	方法名	DescribeInstances
SecretId	密钥 ID	`AKIDz8krbsJ5*********mLPx3EXAMPLE`
Timestamp	当前时间戳	1465185768
Nonce	随机正整数	11886
Region	实例所在区域	ap-guangzhou
InstanceIds.0	待查询的实例 ID	ins-09dx96dg
Offset	偏移量	0
Limit	最大允许输出	20
Version	接口版本号	2017-03-12

1. 对参数排序

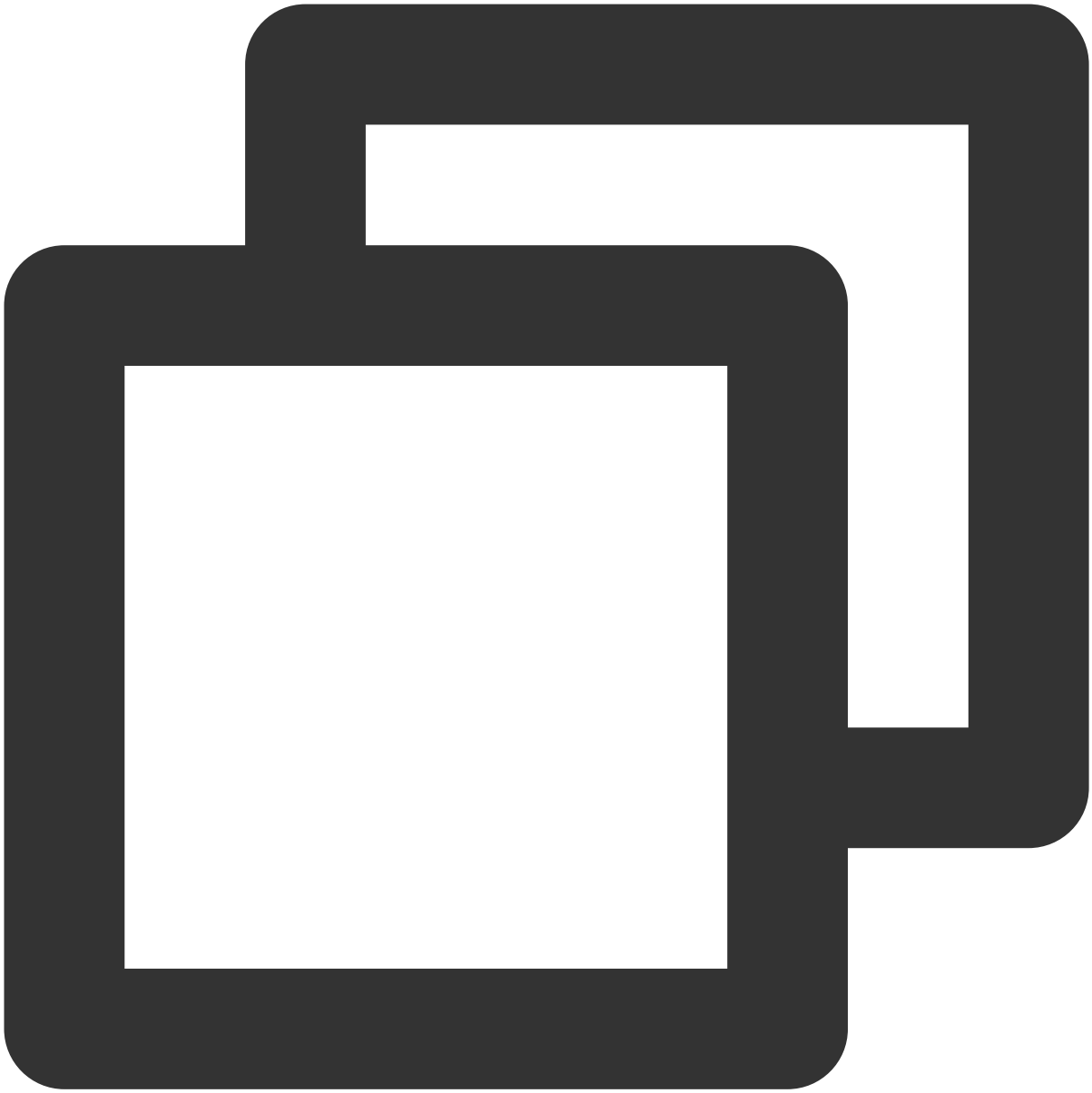
首先对所有请求参数按参数名的字典序（ASCII 码）升序排序。

注意：

只按参数名进行排序，参数值保持对应即可，不参与比大小；

按 ASCII 码比大小，如 InstanceIds.2 要排在 InstanceIds.12 后面，不是按字母表，也不是按数值。用户可以借助编程语言中的相关排序函数来实现这一功能，如 PHP 中的 ksort 函数。

上述示例参数的排序结果如下：



```
{
    'Action' : 'DescribeInstances',
    'InstanceIds.0' : 'ins-09dx96dg',
    'Limit' : 20,
    'Nonce' : 11886,
    'Offset' : 0,
    'Region' : 'ap-guangzhou',
    'SecretId' : 'AKIDz8krbsJ5**********mLPx3EXAMPLE',
    'Timestamp' : 1465185768,
    'Version': '2017-03-12',
}
```

使用程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

2. 拼接规范请求串

此步骤生成请求字符串。 将把上一步排序好的请求参数格式化成“参数名称=参数值”的形式，如对 Action 参数，其参数名称为 " Action " ， 参数值为 " DescribeInstances " ， 因此格式化后就为 Action=DescribeInstances 。

注意：

“参数值”为原始值而非 url 编码后的值。

然后将格式化后的各个参数用"&"拼接在一起，最终生成的请求字符串为:

```
Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&R
```

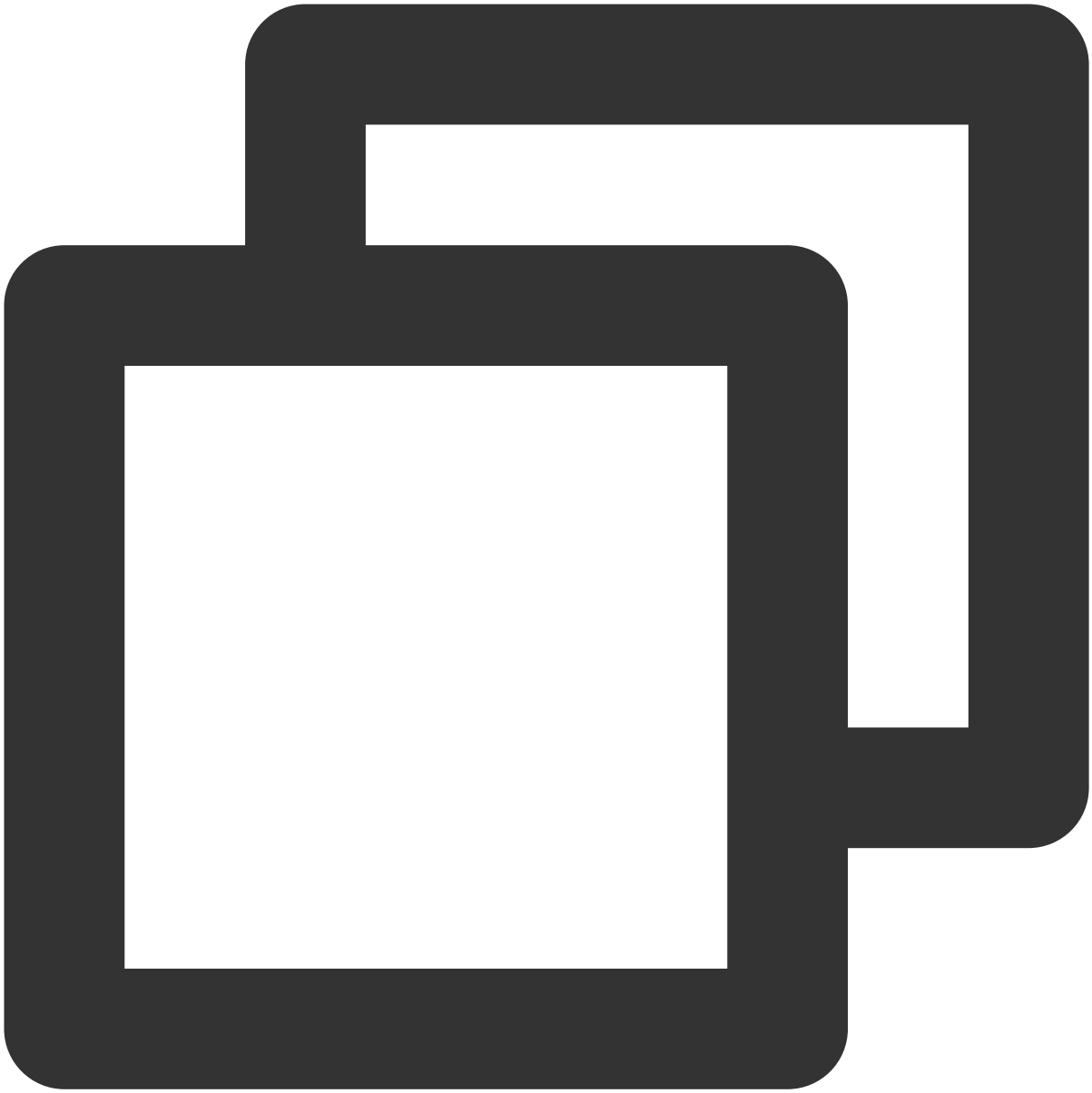
3. 拼接待签名字符串

此步骤生成签名原文字符串。 签名原文字符串由以下几个参数构成:

1. 请求方法：支持 POST 和 GET 方式，这里使用 GET 请求，注意方法为全大写。
2. 请求主机：查看实例列表（DescribeInstances）的请求域名为： `cvm.tencentcloudapi.com` 。实际的请求域名根据接口所属模块的不同而不同，详见各接口说明。
3. 请求路径：当前版本云API的请求路径固定为 / 。
4. 请求字符串：即上一步生成的请求字符串。

签名原文串的拼接规则为： `请求方法 + 请求主机 +请求路径 + ? + 请求字符串` 。

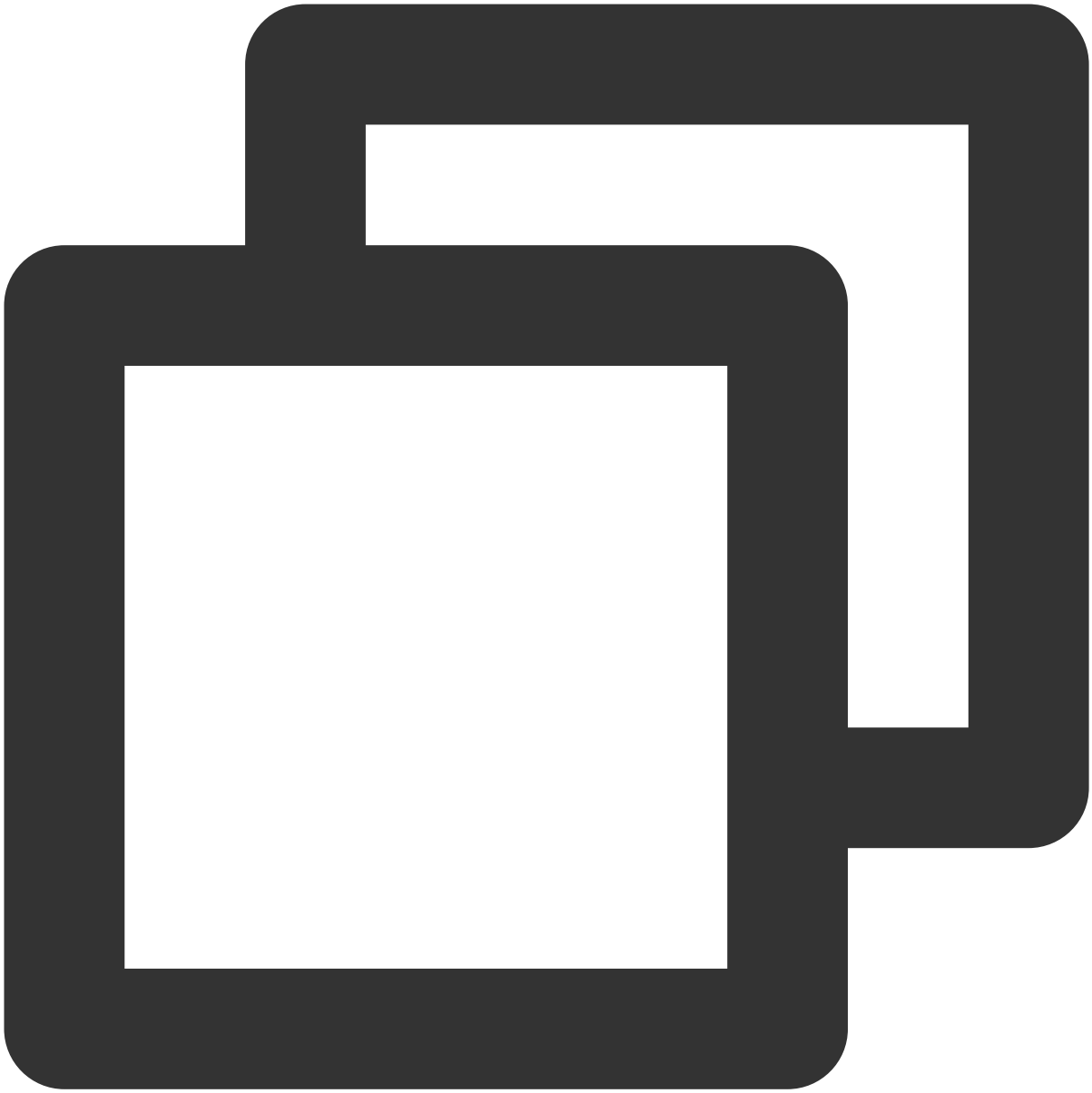
示例的拼接结果为：



```
GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Lim
```

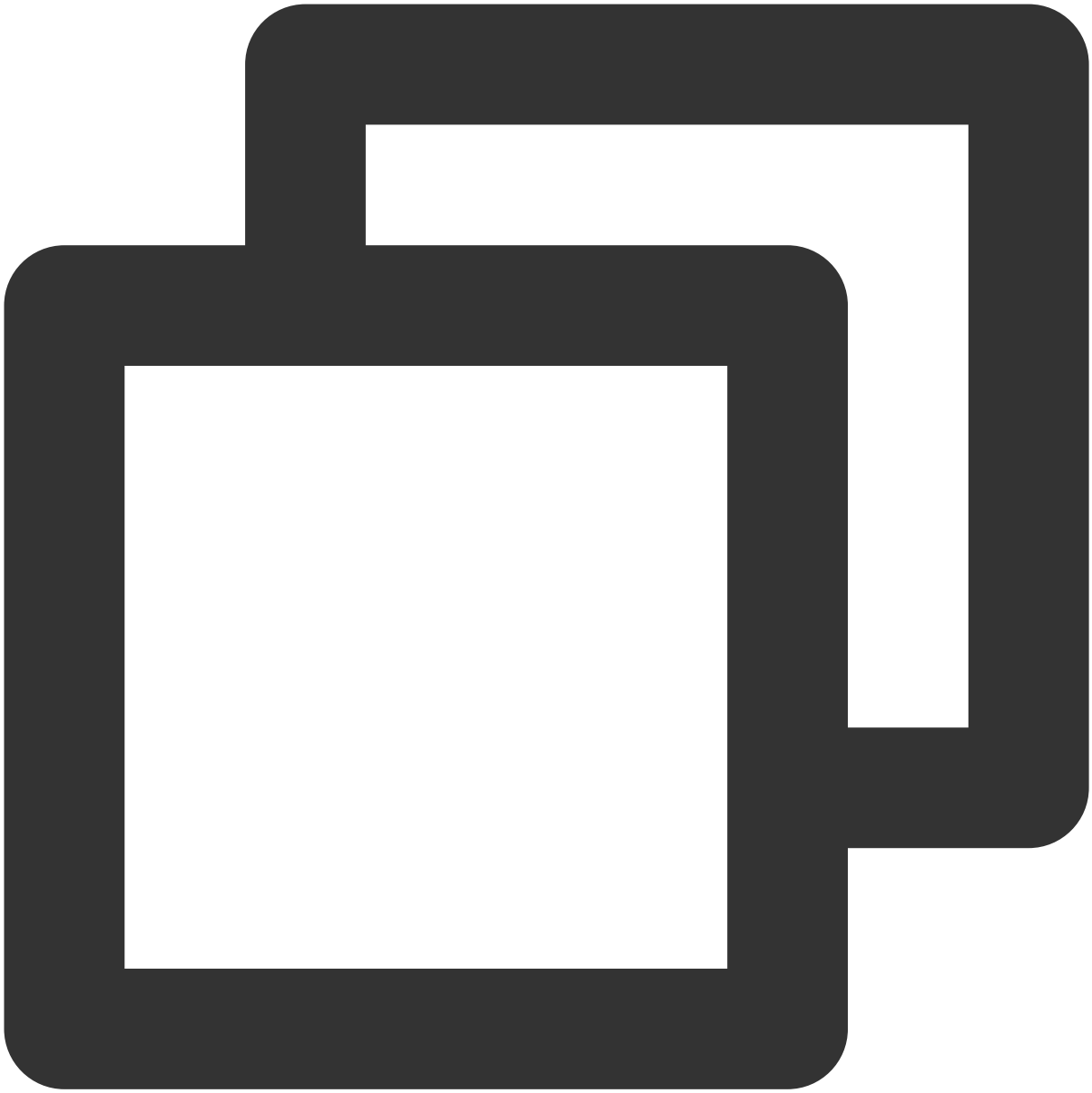
4. 计算签名

此步骤生成签名串。 首先使用 HMAC-SHA1 算法对上一步中获得的**签名原文字符串**进行签名，然后将生成的签名串使用 Base64 进行编码，即可获得签名串，详情请参考下文示例签名。
得到的签名串为：

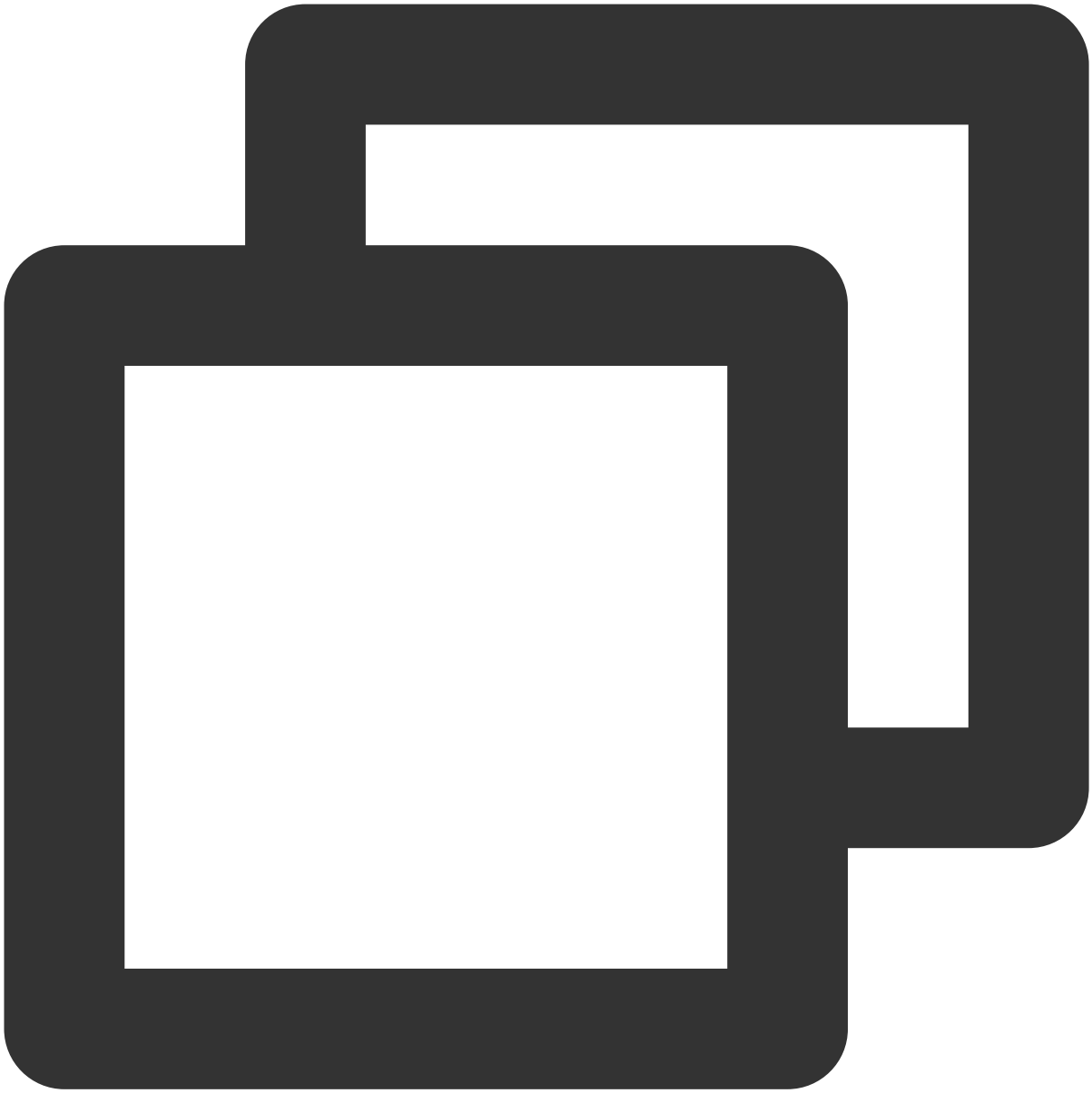


```
Eli*****************cGeI=
```

5. 获取调用信息并发送请求



```
# 实际调用，成功后可能如果是消费接口会产生计费(此处以Python语言为例发送get请求)
resp = requests.get("https://" + endpoint, params=data)
print(resp.url)
```



最终得到的请求串为
`https://cvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96d`

字段名称	解释
endpoint	服务地址， 例如： `cvm.tencentcloudapi.com`
data	API 3.0 签名 V1 所举示例接口参数，**注意：这里需要将计算的签名已键值对的形式加入 data 中**

注意：

由于示例中的密钥是虚构的，时间戳也不是系统当前时间，因此如果将此 url 在浏览器中打开或者用 curl 等命令调用时会返回鉴权错误：签名过期。为了得到一个可以正常返回的 url ，需要修改示例中的 SecretId 和 SecretKey 为真实的密钥，并使用系统当前时间戳作为 Timestamp。
为了更清楚的解释签名过程，下面以nodejs语言为例，将上述的签名过程具体实现。请求的域名、调用的接口和参数的取值都以上述签名过程为准，代码只为解释签名过程，并不具备通用性，实际开发请尽量使用 SDK 。

6. 签名串编码

生成的签名串并不能直接作为请求参数，需要对其进行 URL 编码。
如上一步生成的签名串为 `Eli****************cGeI=` ，最终得到的签名串请求参数（Signature）为： `EliP*********************eI%3D` ，它将用于生成最终的请求 URL。
注意：
如果用户的请求方法是 GET，或者请求方法为 POST 同时 Content-Type 为 application/x-www-form-urlencoded，则发送请求时所有请求参数的值均需要做 URL 编码，参数键和=符号不需要编码。非 ASCII 字符在 URL 编码前需要先以 UTF-8 进行编码。
有些编程语言的网络库会自动为所有参数进行 urlencode，在这种情况下，就不需要对签名串进行 URL 编码了，否则两次 URL 编码会导致签名失败。
其他参数值也需要进行编码，编码采用 RFC 3986。使用 `%XY` 对特殊字符例如汉字进行百分比编码，其中“ X ”和“ Y ”为十六进制字符（0-9 和大写字母 A-F），使用小写将引发错误。

7. API 3.0 签名 V1示例

```
const crypto = require('crypto');

function get_req_url(params, endpoint){
    params['Signature'] = escape(params['Signature']);
    const url_strParam = sort_params(params)
    return "https://" + endpoint + "/?" + url_strParam.slice(1);
}

function formatSignString(reqMethod, endpoint, path, strParam){
    let strSign = reqMethod + endpoint + path + "?" + strParam.slice(1);
    return strSign;
```

```
}
function sha1(secretKey, strsign){
    let signMethodMap = {'HmacSHA1': "sha1"};
    let hmac = crypto.createHmac(signMethodMap['HmacSHA1'], secretKey || "");
    return hmac.update(Buffer.from(strsign, 'utf8')).digest('base64')
}

function sort_params(params){
    let strParam = "";
    let keys = Object.keys(params);
    keys.sort();
    for (let k in keys) {
        //k = k.replace(/_/g, '.');
        strParam += ("&" + keys[k] + "=" + params[keys[k]]);
    }
    return strParam
}

function main(){
    // 密钥参数
    const SECRET_ID = "AKIDz8krbsJ5**********mLPx3EXAMPLE"
    const SECRET_KEY = "Gu5t9xGAR***********EXAMPLE"

    const endpoint = "cvm.tencentcloudapi.com"
    const Region = "ap-guangzhou"
    const Version = "2017-03-12"
    const Action = "DescribeInstances"
    const Timestamp = 1465185768  // 时间戳 2016-06-06 12:02:48, 此参数作为示例, 以实际
    // const Timestamp = Math.round(Date.now() / 1000)
    const Nonce = 11886  // 随机正整数
    //const nonce = Math.round(Math.random() * 65535)

    let params = {};
    params['Action'] = Action;
    params['InstanceIds.0'] = 'ins-09dx96dg';
    params['Limit'] = 20;
    params['Offset'] = 0;
    params['Nonce'] = Nonce;
    params['Region'] = Region;
    params['SecretId'] = SECRET_ID;
    params['Timestamp'] = Timestamp;
    params['Version'] = Version;

    // 1. 对参数排序,并拼接请求字符串
    strParam = sort_params(params)

    // 2. 拼接签名原文字符串
```

```
    const reqMethod = "GET";
    const path = "/";
    strSign = formatSignString(reqMethod, endpoint, path, strParam)
    console.log(strSign)
    console.log("-----------------------")

    // 3. 生成签名串
    params['Signature'] = sha1(SECRET_KEY, strSign)
    console.log(params['Signature'])
    console.log("-----------------------")

    // 4. 进行url编码并拼接请求url
    const req_url = get_req_url(params, endpoint)
    console.log(params['Signature'])
    console.log("-----------------------")
    console.log(req_url)
}
main()
```

API 2.0签名

此签名现已不在维护，建议使用性能更优的 **API 3.0签名**，如需使用，请直接在 API Explorer 的**签名串生成 > 选择 API 2.0签名**版本进行操作。

签名失败

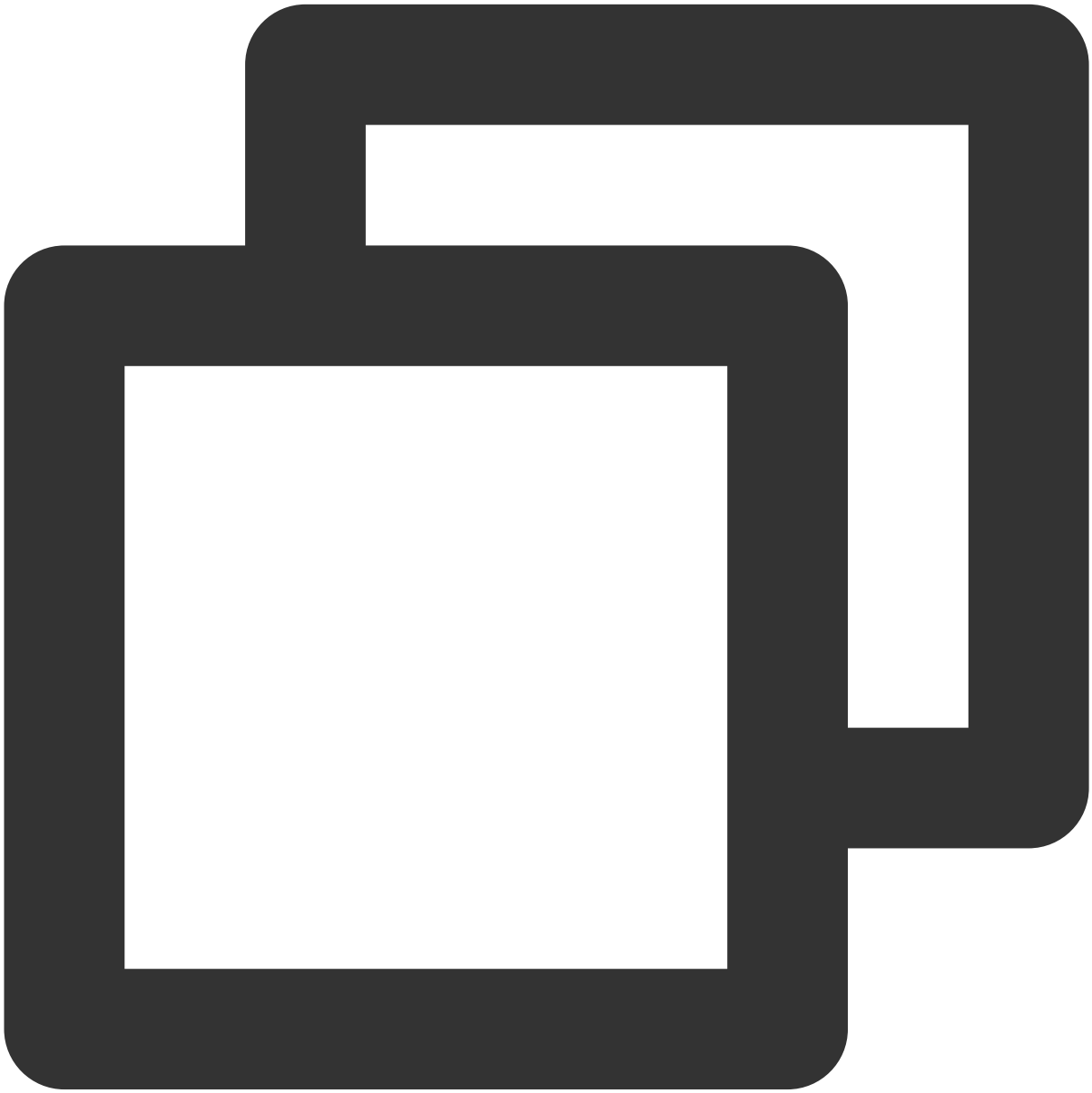
存在以下签名失败的错误码，请根据实际情况处理。

错误码	错误描述
AuthFailure.SignatureExpire	签名过期。Timestamp 与服务器接收到请求的时间相差不得超过五分钟。
AuthFailure.SecretIdNotFound	密钥不存在。请到控制台查看密钥是否被禁用，是否少复制了字符或者多了字符。
AuthFailure.SignatureFailure	签名错误。可能是签名计算错误，或者签名与实际发送的内容不相符合，也有可能是密钥 SecretKey 错误导致的。
AuthFailure.TokenFailure	临时证书 Token 错误。
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。

返回结果

正确返回结果

以云服务器的接口查看实例状态列表（DescribeInstancesStatus）2017-03-12版本为例，若调用成功，其可能的返回如下为：



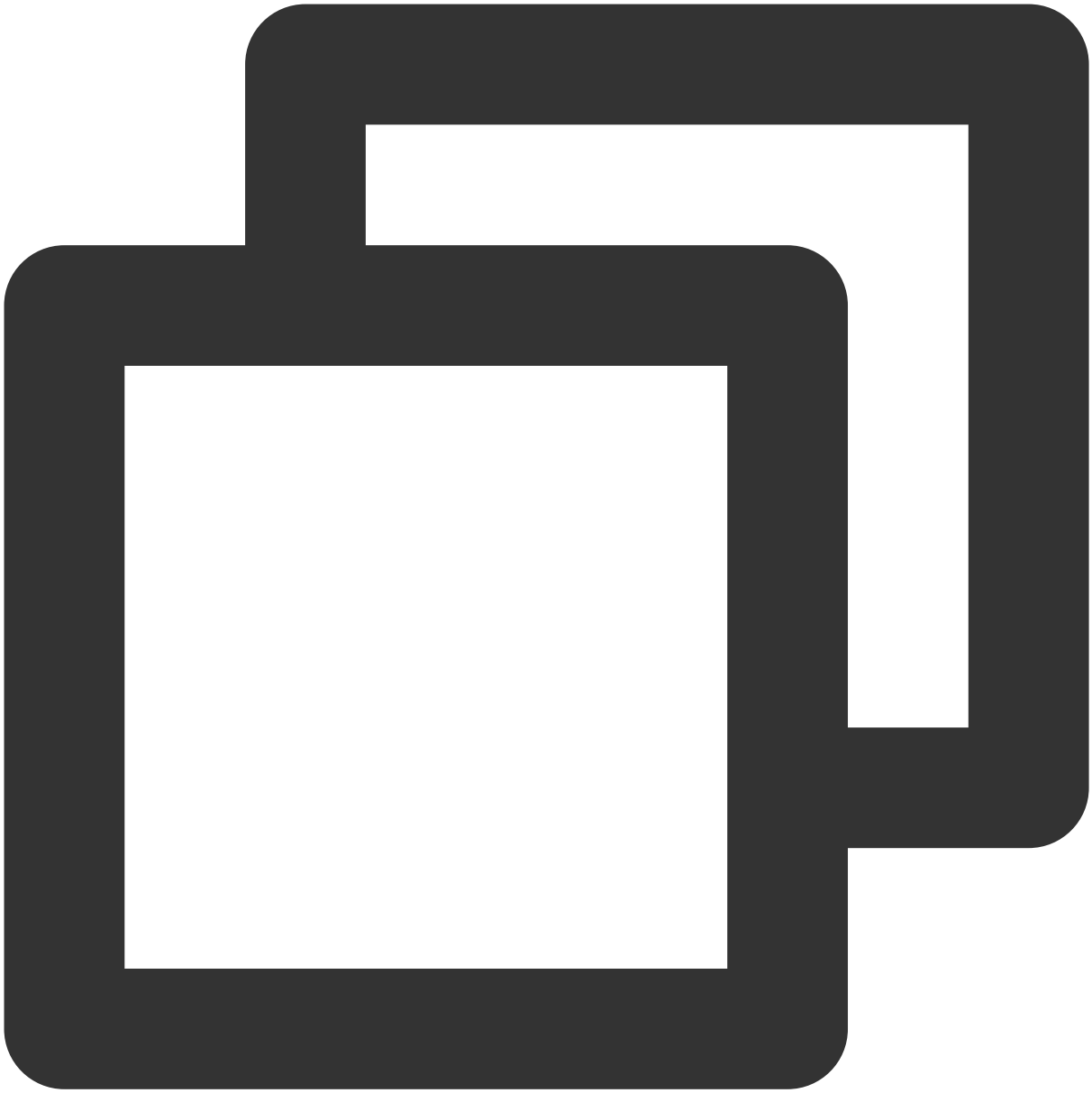
```
{
    "Response": {
        "TotalCount": 0,
        "InstanceStatusSet": [],
```

```
        "RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
    }
}
```

`Response` 及其内部的 `RequestId` 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。
除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 `TotalCount` 和 `InstanceStatusSet` 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 `TotalCount` 在此情况下的返回值为0， `InstanceStatusSet` 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：




```
{
    "Response": {
        "Error": {
            "Code": "AuthFailure.SignatureFailure",
            "Message": "The provided credentials could not be validated. Please che
        },
        "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
    }
}
```

`Error` 的出现代表着该请求调用失败。 `Error` 字段连同其内部的 `Code` 和 `Message` 字段在调用失败时是必定返回的。

`Code` 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。

`Message` 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。

`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。 Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。
LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。

版权所有：腾讯云计算（北京）有限责任公司

ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	HTTPS 请求方法错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

版权所有：腾讯云计算（北京）有限责任公司

PHP API

最近更新时间：2023-03-07 18:16:40

腾讯云 API 全新升级3.0，该版本进行了性能优化且全地域部署、支持就近和按地域接入、访问时延下降显著，接口描述更加详细、错误码描述更加全面、SDK 增加接口级注释，让您更加方便快捷的使用腾讯云产品。这里针对 PHP API 调用方式进行简单说明。
现已支持云服务器（CVM）、云硬盘（CBS）、私有网络（VPC）、云数据库（TencentDB）等 腾讯云产品，后续会支持其他的云产品接入，敬请期待。

了解请求结构

1. 服务地址（endpoint）

API 支持就近地域接入（例如：cvm 产品域名为 `cvm.tencentcloudapi.com`），也支持指定地域域名访问（例如：广州地域的域名为 `cvm.ap-guangzhou.tencentcloudapi.com`），各地域参数见下文公共参数中的地域列表，详情请参见各产品的“请求结构”文档说明判断是否支持该地域。
注意：
对时延敏感的业务，建议指定带地域的域名。

2. 通信协议

腾讯云 API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法：
POST（推荐）
GET
POST 请求支持的 Content-Type 类型：
application/json（推荐），必须使用签名方法 v3（TC3-HMAC-SHA256）。
application/x-www-form-urlencoded，必须使用签名方法 v1（HmacSHA1 或 HmacSHA256）。
multipart/form-data（仅部分接口支持），必须使用签名方法 v3（TC3-HMAC-SHA256）。
GET 请求的请求包大小不得超过32KB。POST 请求使用签名方法 v1（HmacSHA1、HmacSHA256）时不得超过1MB。POST 请求使用签名方法 v3（TC3-HMAC-SHA256）时支持10MB。

4. 字符编码

均使用 UTF-8 编码。

版权所有：腾讯云计算（北京）有限责任公司

公共参数

说明：
公共参数是用于标识用户和接口签名的参数，每次请求均需要携带这些参数，才能正常发起请求。

签名方法 V3 公共参数

签名方法 v3（有时也称作 TC3-HMAC-SHA256）相比签名方法 v1 （部分文档简称为“签名方法”），更安全，支持更大的请求包，支持 POST JSON 格式，性能有一定提升，推荐使用该签名方法计算签名。使用方法见下文 “签名方法介绍” 。

参数名称	类型	必选	描述
X-TC-Action	String	**是**	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
X-TC-Timestamp	Integer	**是**	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702。**注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。**
X-TC-Version	String	**是**	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
Authorization	String	**是**	HTTP 标准身份认证头部字段，例如： TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=72e494ea8**********a96525168 其中： TC3-HMAC-SHA256：签名方法，目前固定取该值。 Credential：签名凭证，AKIDEXAMPLE 是 SecretId。 Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致。 service 为产品名，通常为域名前缀，例如域名 cvm.tencentcloudapi.com 意味着产品名是 cvm。本产品取值为 cvm。 SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部。 Signature：签名摘要，计算过程详见下文。
X-TC-Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

签名方法 V1 公共参数

使用签名方法 v1 （有时会称作 HmacSHA256 和 HmacSHA1），公共参数需要统一放到请求串中。

参数名称	类型	必选	描述
Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如 1529223702，如果与当前时间相差过大，会引起签名过期错误。
Nonce	Integer	是	随机正整数，与 Timestamp 联合起来，用于防止重放攻击。
SecretId	String	是	在 云API密钥 上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey，而 SecretKey 会用来生成请求签名 Signature。
Signature	String	是	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。具体计算方法参见下文“签名方法介绍” 。
Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
SignatureMethod	String	否	签名方式，目前支持 HmacSHA256 和 HmacSHA1。只有指定此参数为 HmacSHA256 时，才使用 HmacSHA256 算法验证签名，其他情况均使用 HmacSHA1 验证签名。
Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token 。

地域列表

由于各个产品支持地域不同，具体详情请参考各产品文档中的地域列表。
例如，您可以参考云服务器的 地域列表。

PHP API 调用方式

腾讯云 API 会对每个请求进行身份验证，用户需要使用安全凭证，经过特定的步骤对请求进行签名（Signature），每个请求都需要在公共请求参数中指定该签名结果并以指定的方式和格式发送请求。

步骤1：申请安全凭证

本文使用的安全凭证为密钥，密钥包括 SecretId 和 SecretKey。每个用户最多可以拥有两对密钥。

SecretId：用于标识 API 调用者身份，可以简单类比为用户名。

SecretKey：用于验证 API 调用者的身份，可以简单类比为密码。

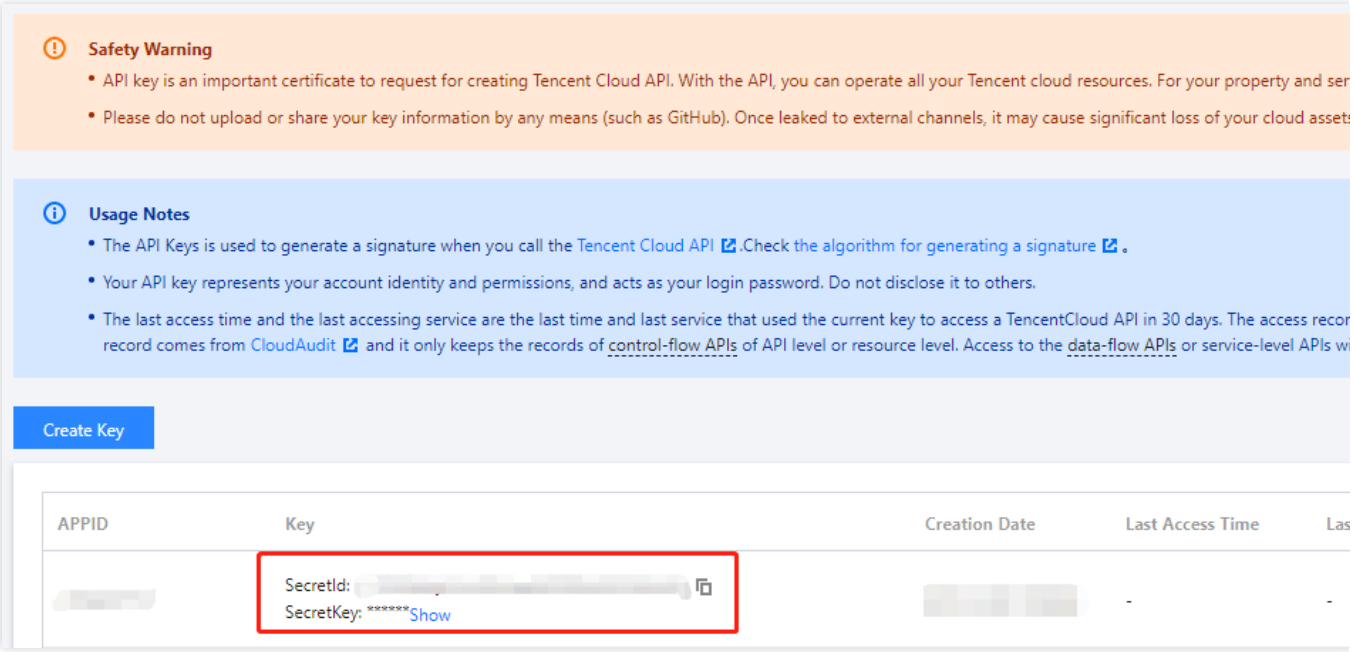
注意：

用户必须严格保管安全凭证，避免泄露，否则将危及财产安全。如已泄漏，请立刻禁用该安全凭证。

假设用户的 SecretId 和 SecretKey 分别是：`AKIDz8krbsJ5*********mLPx3EXAMPLE` 和 `Gu5t9xGAR**********EXAMPLE`。用户想查看广州区云服务器名为“未命名”的主机状态，只返回一条数据。

则请求可能为：

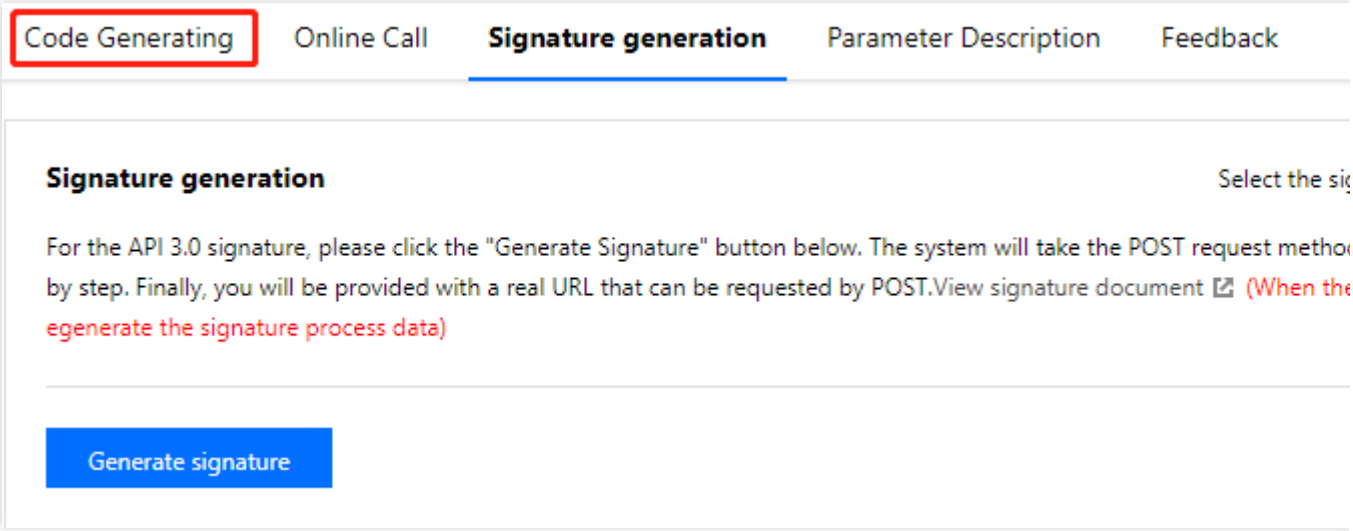
前往 API密钥管理 页面，即可进行获取。如下图所示：



步骤2

1. 获取 API 3.0 V3 版本签名

签名方法 v3（TC3-HMAC-SHA256）功能上覆盖了以前的签名方法 v1，而且更安全，支持更大的请求，支持 json 格式，性能有一定提升，推荐使用该签名方法计算签名。



注意：

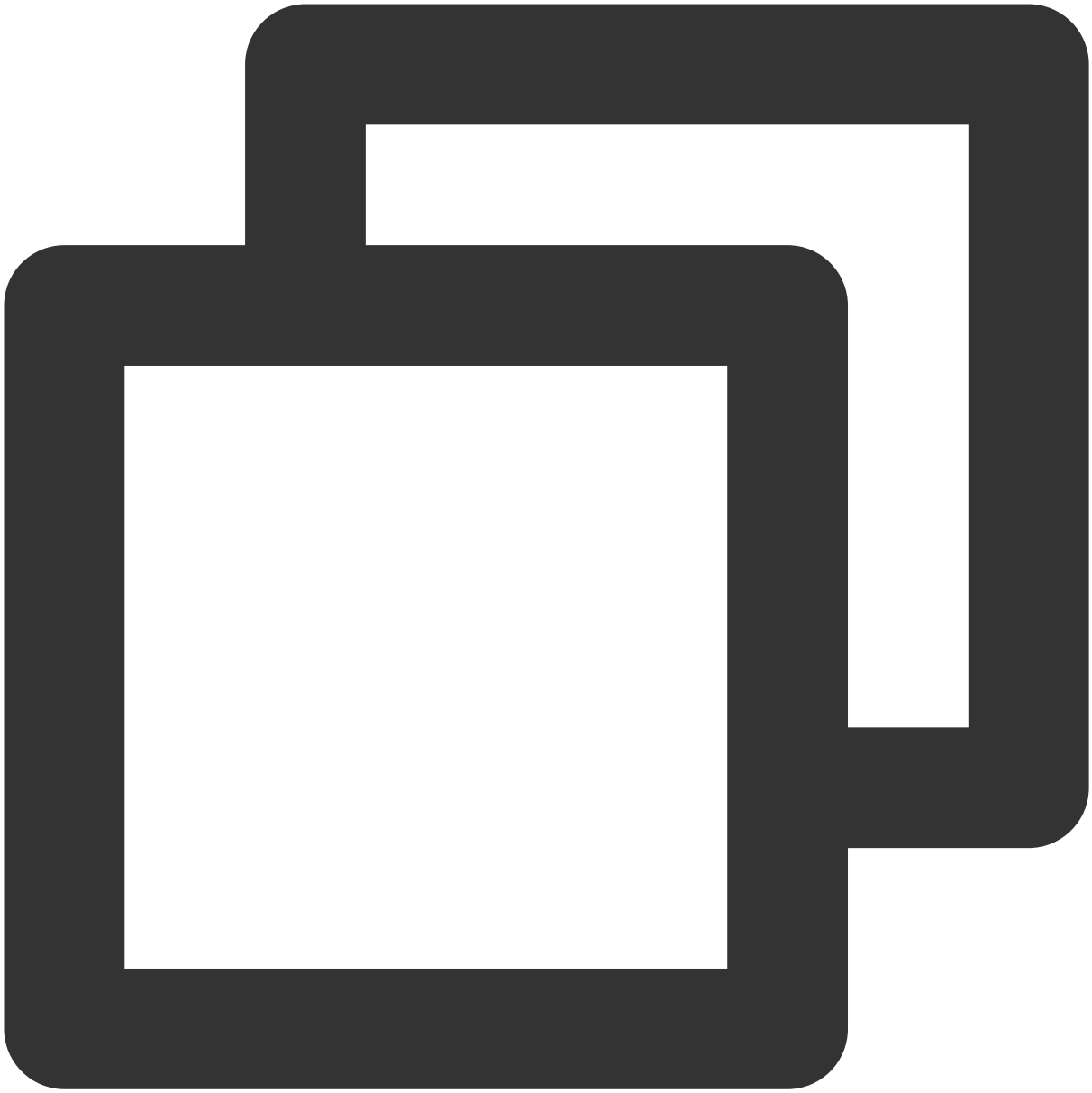
首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“API 3.0 签名 v3”，可以生成签名过程进行验证，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的 7 种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

云 API 支持 GET 和 POST 请求。对于 GET 方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。对于POST方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式绝大多数接口均支持，multipart 格式只有特定接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。推荐使用 POST 请求，因为两者的结果并无差异，但 GET 请求只支持32KB以内的请求包。

下面以 查看实例列表 接口为例，分步骤介绍签名的计算过程。我们选择该接口是因为：

1. 云服务器默认已开通，该接口很常用；
2. 该接口是只读的，不会改变现有资源的状态；
3. 接口覆盖的参数种类较全，可以演示包含数据结构的数组如何使用。

1. 拼接规范请求串

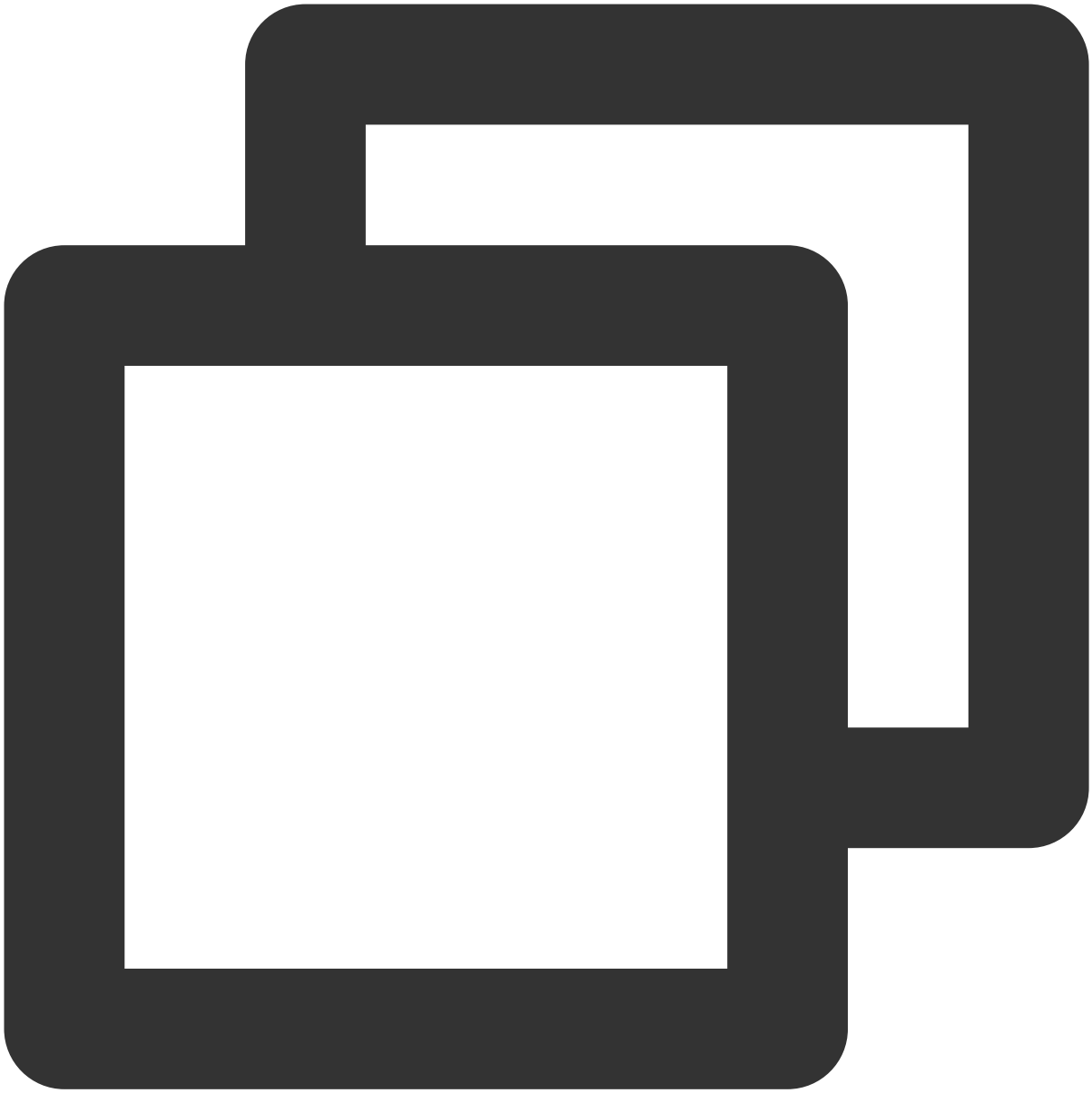


```
CanonicalRequest =
    HTTPRequestMethod + '\\n' +
    CanonicalURI + '\\n' +
    CanonicalQueryString + '\\n' +
    CanonicalHeaders + '\\n' +
    SignedHeaders + '\\n' +
    HashedRequestPayload
```

字段名称	解释

HTTPRequestMethod	HTTP 请求方法（GET、POST ）。此示例取值为 `POST` 。
CanonicalURI	URI 参数，API 3.0 固定为正斜杠（/）。
CanonicalQueryString	发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串""，对于 G 求，则为 URL 中问号（?）后面的字符串内容，例如：Limit=10&Offset=0。 注意：CanonicalQueryString 需要参考 RFC3986 进行 URLEncode，字符集 UTF8，推荐使用 语言标准库，所有特殊字符均需编码，大写形式。
CanonicalHeaders	参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部 签名以提高自身请求的唯一性和安全性。 拼接规则：头部 key 和 value 统一转成小写，掉首尾空格，按照 key:value\\n 格式拼接；多个头部，按照头部 key（小写）的 ASCII 行拼接。此示例计算结果是 `content-type:application/json; charset=ut 8\\nhost:cvm.tencentcloudapi.com\\n` 。 注意：content-type 必须和实际发 符合，有些编程语言网络库即使未指定也会自动添加 charset 值，如果签名时和发送时 致，服务器会返回签名校验失败。
SignedHeaders	参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包 部内容是一一对应的。content-type 和 host 为必选头部。 拼接规则：头部 key 统一转 写；多个头部 key（小写）按照 ASCII 升序进行拼接，并且以分号（;）分隔。此示例 content-type;host
HashedRequestPayload	请求正文（Requestpayload，即 body，此示例为 `{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-name"` 的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(RequestPayload))), HTTP 请求正文做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。 GET 请求，RequestPayload 固定为空字符串。此示例计算结果是 `35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f`

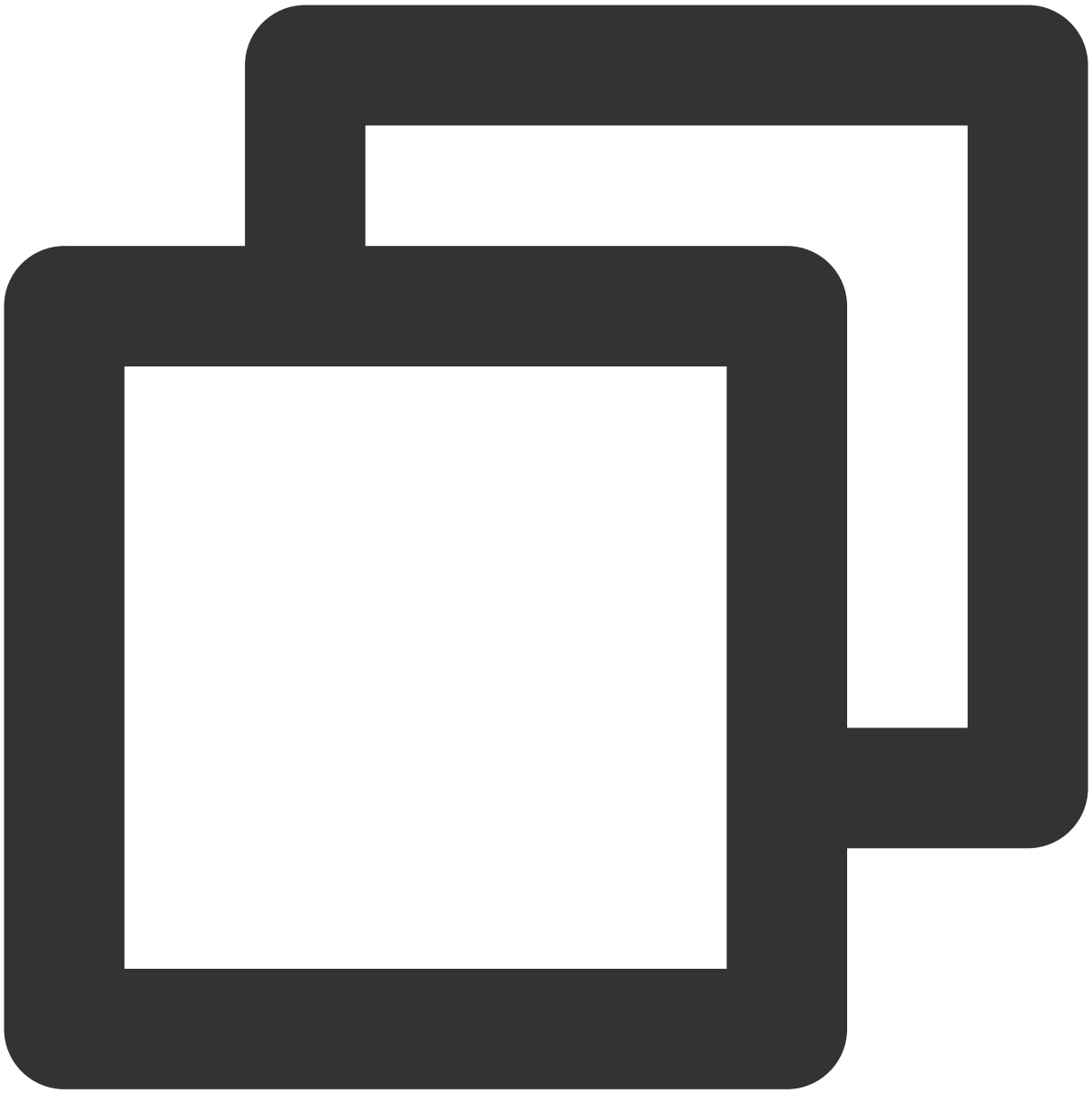
根据以上规则，示例中得到的规范请求串如下：



```
POST
/

content-type:application/json; charset=utf-8
host:cvm.tencentcloudapi.com

content-type;host
35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f064
```

2. 拼接待签名字符串

按如下格式拼接待签名字符串：



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

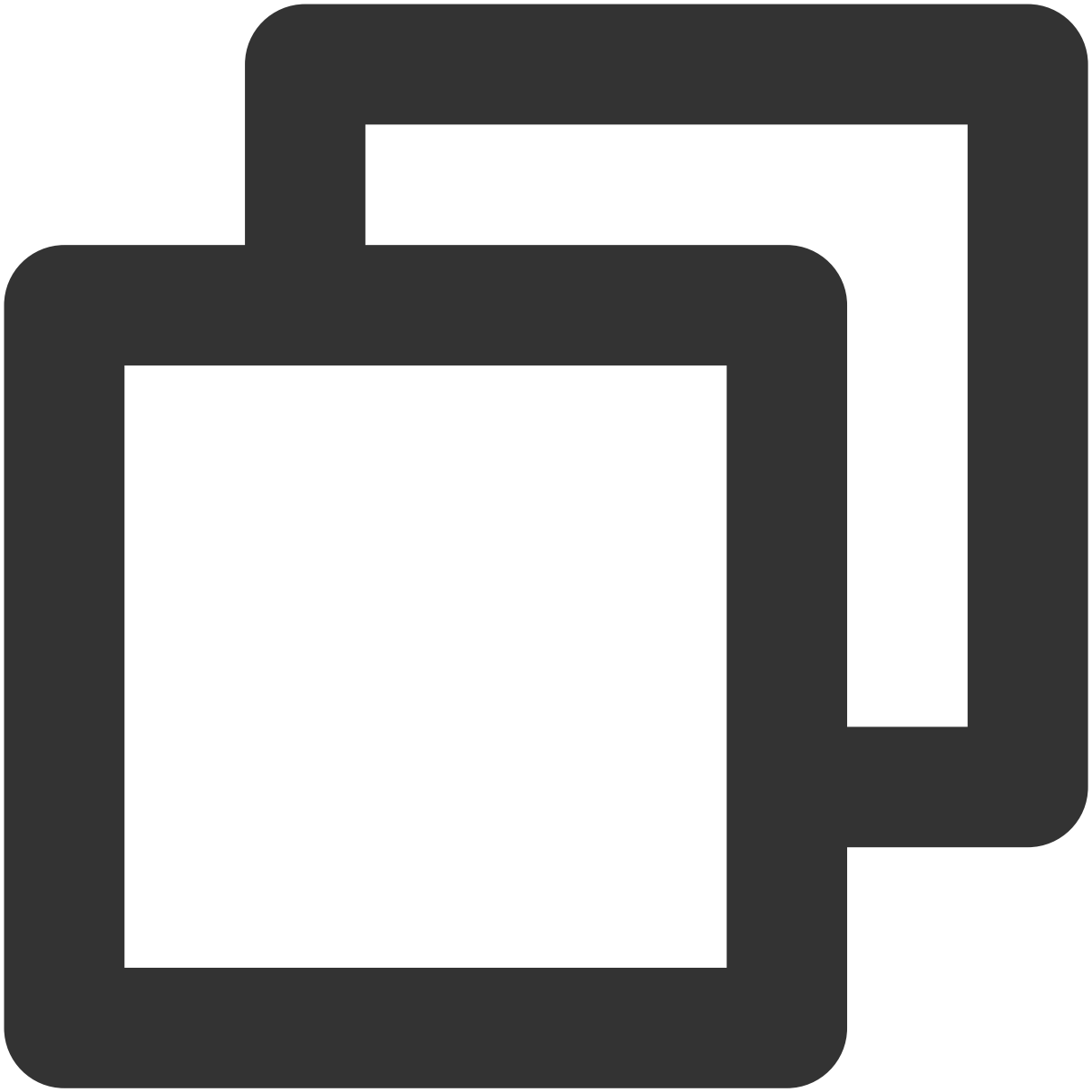
字段名称	解释
Algorithm	签名算法，目前固定为 `TC3-HMAC-SHA256` 。

RequestTimestamp	请求时间戳，即请求头部的公共参数 X-TC-Timestamp 取值，取当前时间 UNIX 时间确到秒。此示例取值为 `1551113065` 。
CredentialScope	凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符串（tc3_request）。**Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Time 换算的 UTC 标准时间日期一致**；service 为产品名，必须与调用的产品域名一致。此算结果是 2019-02-25/cvm/tc3_request。
HashedCanonicalRequest	前述步骤拼接所得规范请求串的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))。此示例计算结果是 `5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d`

注意：

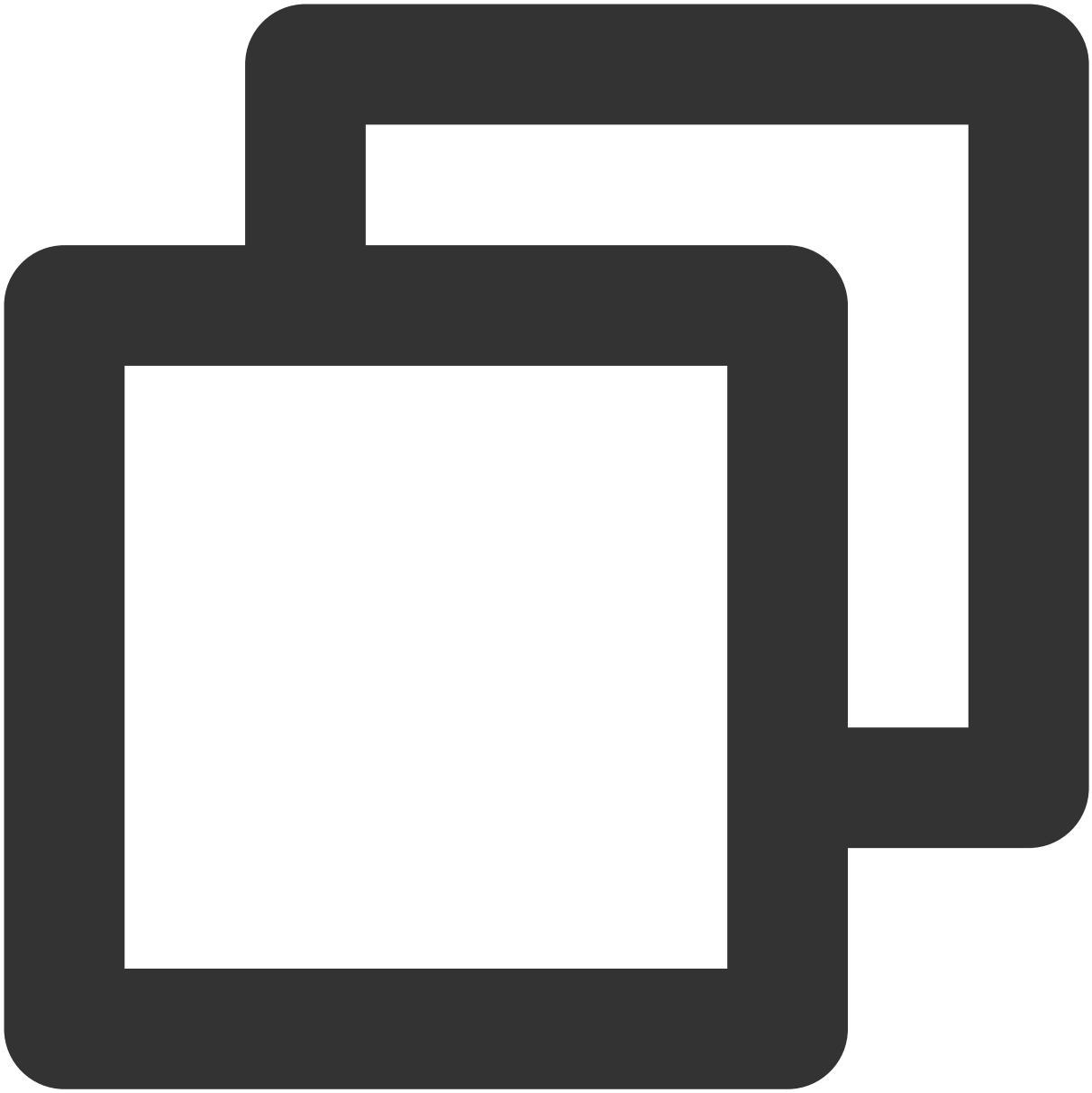
1. Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为 1551113065，在东八区的时间是 2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为 2019-02-25，而不是 2019-02-26。
2. Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败，返回签名过期错误。

根据以上规则，示例中得到的待签名字符串如下：



```
TC3-HMAC-SHA256
1551113065
2019-02-25/cvm/tc3_request
5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d7031
```

3.计算签名(伪代码)



```
$secretDate = hash_hmac("SHA256", $date, "TC3".$secretKey, true);
$secretService = hash_hmac("SHA256", $service, $secretDate, true);
$secretSigning = hash_hmac("SHA256", "tc3_request", $secretService, true);
$signature = hash_hmac("SHA256", $stringToSign, $secretSigning);
echo $signature.PHP_EOL;
```

派生出的密钥 `SecretDate` 、 `SecretService` 和 `SecretSigning` 是二进制的数据，可能包含不可打印字符，此处不展示中间结果。

字段名称	解释

secretKey	原始的 SecretKey，即 `Gu5t9xGAR**********EXAMPLE` 。
date	即 Credential 中的 Date 字段信息。此示例取值为 `2019-02-25` 。
service	即 Credential 中的 Service 字段信息。此示例取值为 `cvm` 。

此示例计算结果是 `72e494ea8****************************************a96525168` 。

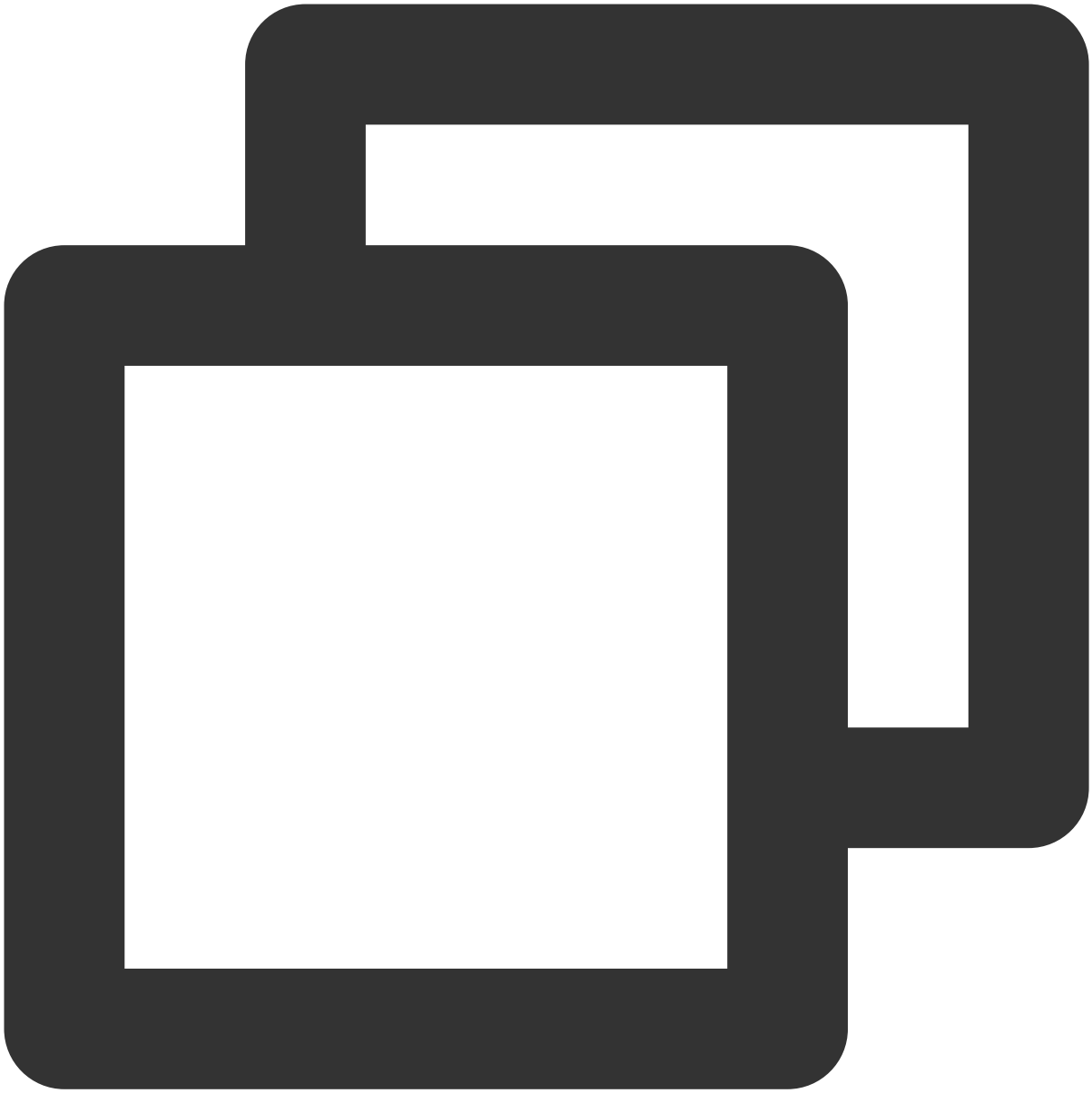
4. 拼接 Authorization

按如下格式拼接 Authorization：

```
$authorization = $algorithm
    ." Credential=".$secretId."/".$credentialScope
    ., SignedHeaders=content-type;host, Signature=".$signature;
echo $authorization.PHP_EOL;
```

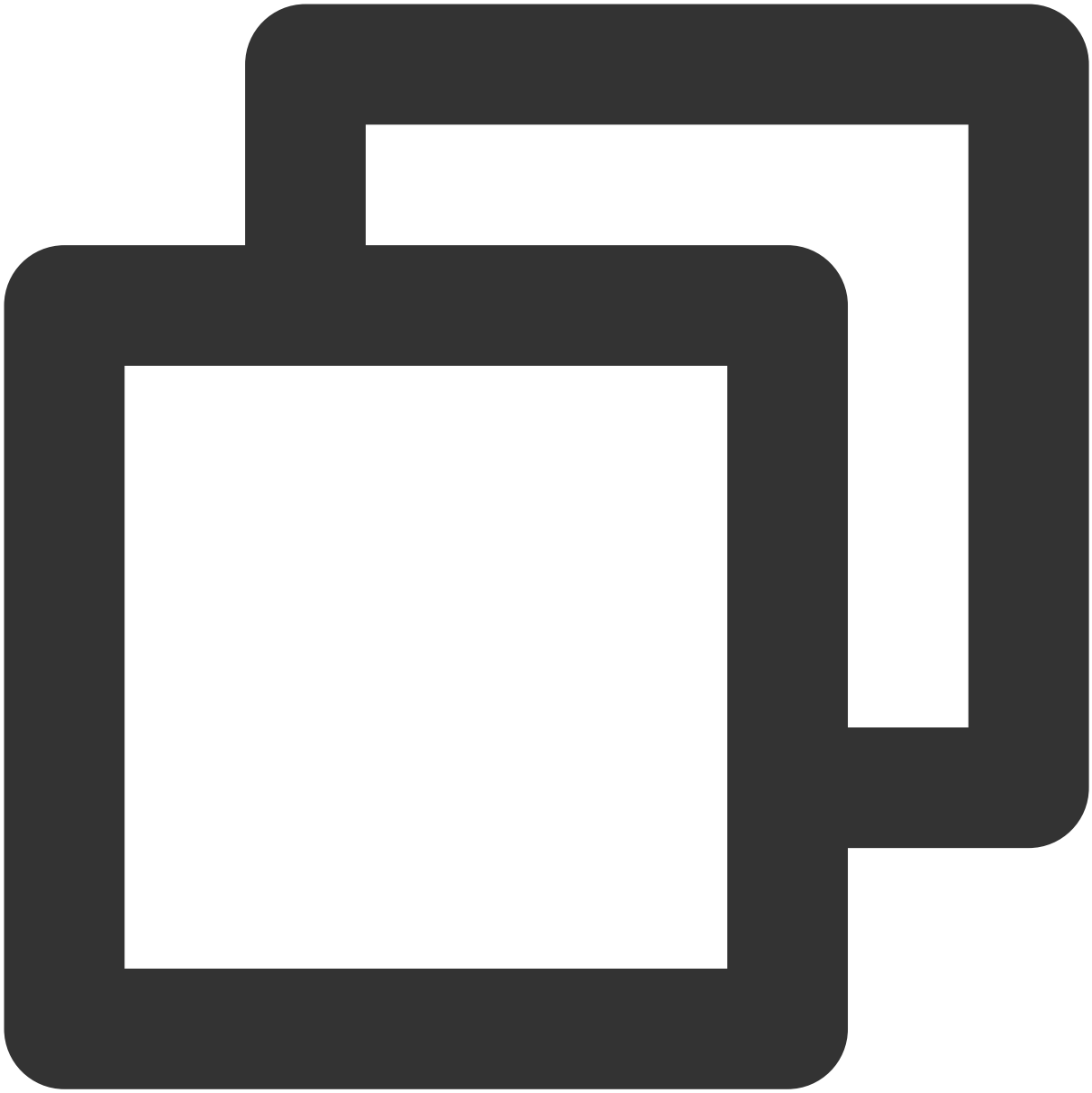
字段名称	解释
algorithm	签名方法，固定为 `TC3-HMAC-SHA256` 。
secretId	密钥对中的 SecretId，即 `AKIDz8krbsJ5*********mLPx3EXAMPLE` 。
credentialScope	见上文，凭证范围。此示例计算结果是 `2019-02-25/cvm/tc3_request` 。
SignedHeaders	见上文，参与签名的头部信息。此示例取值为 `content-type;host` 。
signature	签名值。此示例计算结果是 `72e494ea8*****************************************a96525168` 。

根据以上规则，示例中得到的值为：



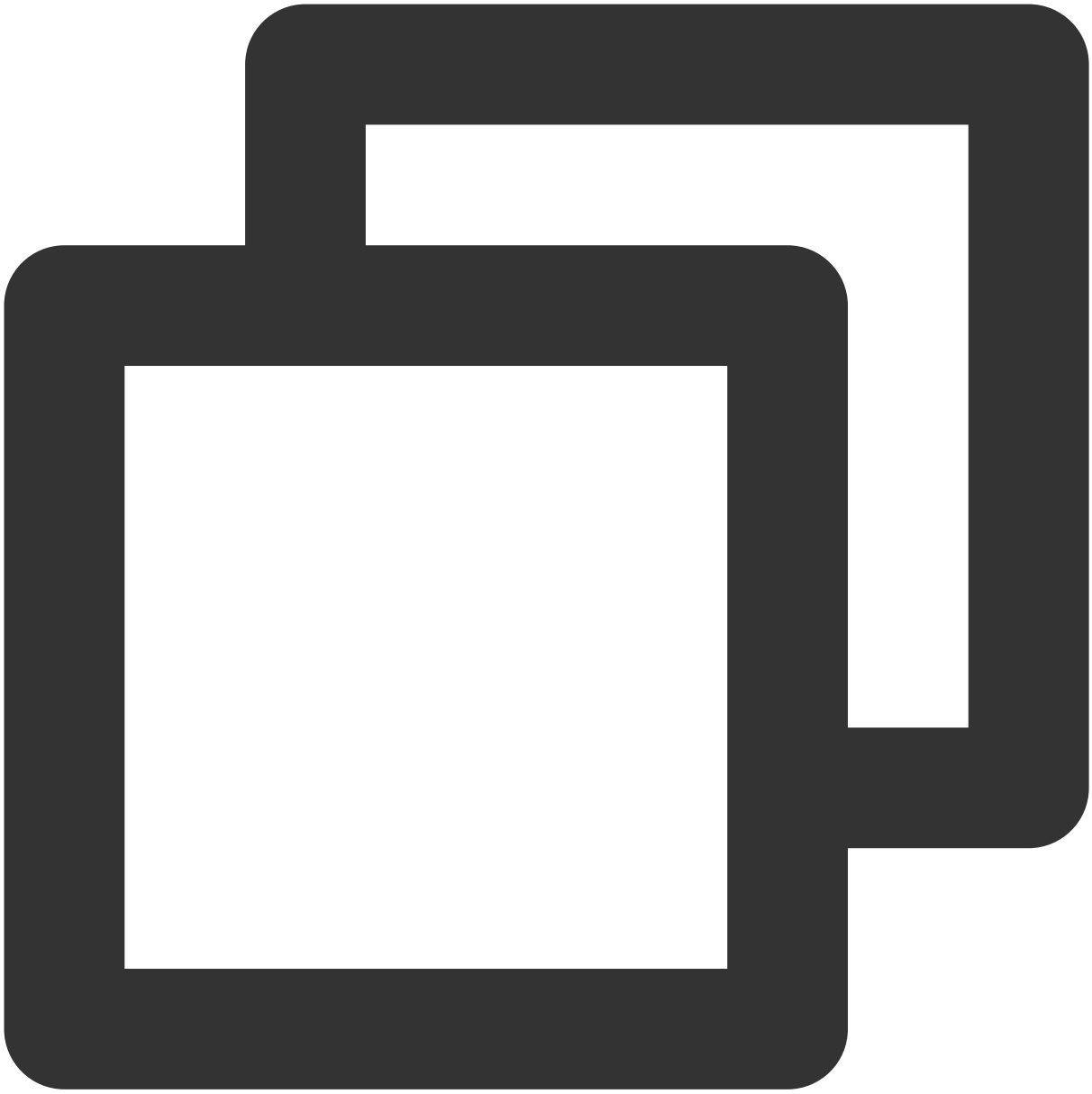
```
TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/2019-02-25/cvm/tc3_re
```

最终完整的调用信息如下：



```
POST https://cvm.tencentcloudapi.com/
Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5**********mLPx3EXAMPLE/2019-0
Content-Type: application/json; charset=utf-8
Host: cvm.tencentcloudapi.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1551113065
X-TC-Region: ap-guangzhou

{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-na
```

5. API 3.0 签名 V3示例



```
<?php
$secretId = "AKIDz8krbsJ5**********mLPx3EXAMPLE";
$secretKey = "Gu5t9xGAR***********EXAMPLE";
$host = "cvm.tencentcloudapi.com";
$service = "cvm";
$version = "2017-03-12";
$action = "DescribeInstances";
$region = "ap-guangzhou";
// $timestamp = time();
```

```
$timestamp = 1551113065;
$algorithm = "TC3-HMAC-SHA256";

// step 1: build canonical request string
$httpRequestMethod = "POST";
$canonicalUri = "/";
$canonicalQueryString = "";
$canonicalHeaders = "content-type:application/json; charset=utf-8\\n"."host:".$host
$signedHeaders = "content-type;host";
$payload = '{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name":
$hashedRequestPayload = hash("SHA256", $payload);
$canonicalRequest = $httpRequestMethod."\\n"
    .$canonicalUri."\\n"
    .$canonicalQueryString."\\n"
    .$canonicalHeaders."\\n"
    .$signedHeaders."\\n"
    .$hashedRequestPayload;
echo $canonicalRequest.PHP_EOL;

// step 2: build string to sign
$date = gmdate("Y-m-d", $timestamp);
$credentialScope = $date."/".$service."/tc3_request";
$hashedCanonicalRequest = hash("SHA256", $canonicalRequest);
$stringToSign = $algorithm."\\n"
    .$timestamp."\\n"
    .$credentialScope."\\n"
    .$hashedCanonicalRequest;
echo $stringToSign.PHP_EOL;

// step 3: sign string
$secretDate = hash_hmac("SHA256", $date, "TC3".$secretKey, true);
$secretService = hash_hmac("SHA256", $service, $secretDate, true);
$secretSigning = hash_hmac("SHA256", "tc3_request", $secretService, true);
$signature = hash_hmac("SHA256", $stringToSign, $secretSigning);
echo $signature.PHP_EOL;

// step 4: build authorization
$authorization = $algorithm
    ." Credential=".$secretId."/".$credentialScope
    ", SignedHeaders=content-type;host, Signature=".$signature;
echo $authorization.PHP_EOL;

$curl = "curl -X POST https://".$host
    .' -H "Authorization: '.$authorization.'"'
    .' -H "Content-Type: application/json; charset=utf-8"'
    .' -H "Host: '.$host.'"'
    .' -H "X-TC-Action: '.$action.'"'
```

```
    .' -H "X-TC-Timestamp: '.$timestamp.'"'
    .' -H "X-TC-Version: '.$version.'"'
    .' -H "X-TC-Region: '.$region.'"'
    ." -d '".$payload."'";
echo $curl.PHP_EOL;
```

2. 获取 API 3.0 V1 版本签名

签名方法 v1 简单易用，但是功能和安全性都不如签名方法 v3，推荐使用签名方法 v3。

注意：

首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“API 3.0 签名 v1”，可以生成签名过程进行验证，并提供了部分编程语言的签名示例，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的 7 种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

以云服务器查看实例列表(DescribeInstances)请求为例，当用户调用这一接口时，其请求参数可能如下:

参数名称	中文	参数值
Action	方法名	DescribeInstances
SecretId	密钥 ID	`AKIDz8krbsJ5**********mLPx3EXAMPLE`
Timestamp	当前时间戳	1465185768
Nonce	随机正整数	11886
Region	实例所在区域	ap-guangzhou
InstanceIds.0	待查询的实例 ID	ins-09dx96dg
Offset	偏移量	0
Limit	最大允许输出	20
Version	接口版本号	2017-03-12

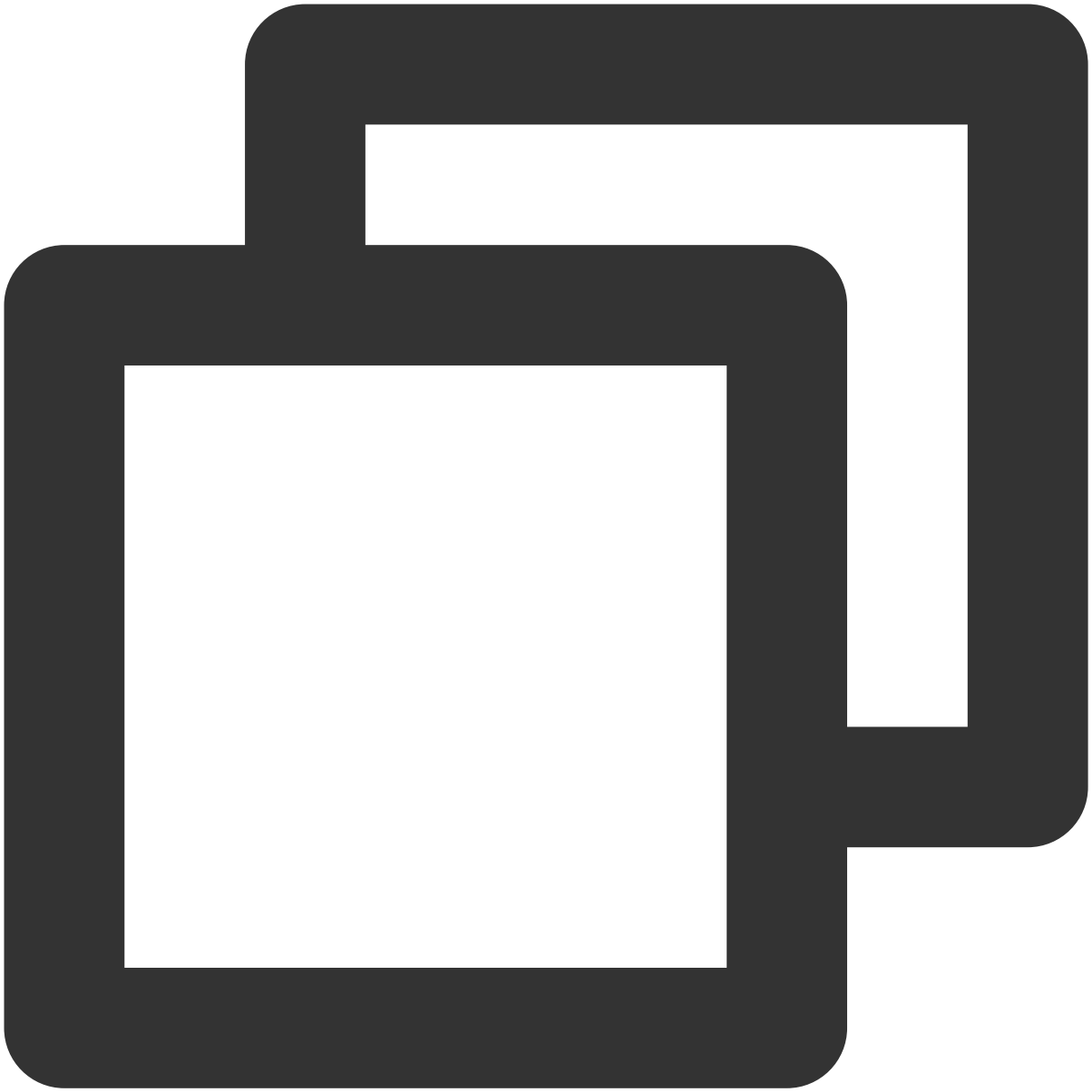
1. 对参数排序

首先对所有请求参数按参数名的字典序（ASCII 码）升序排序。

注意：

1. 只按参数名进行排序，参数值保持对应即可，不参与比大小。
2. 按 ASCII 码比大小，如 InstanceIds.2 要排在 InstanceIds.12 后面，不是按字母表，也不是按数值。用户可以借助编程语言中的相关排序函数来实现这一功能，如 PHP 中的 ksort 函数。

上述示例参数的排序结果如下：



```
{
    'Action' : 'DescribeInstances',
    'InstanceIds.0' : 'ins-09dx96dg',
    'Limit' : 20,
    'Nonce' : 11886,
    'Offset' : 0,
    'Region' : 'ap-guangzhou',
    'SecretId' : 'AKIDz8krbsJ5**********mLPx3EXAMPLE',
    'Timestamp' : 1465185768,
    'Version': '2017-03-12',
}
```

使用程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

2. 拼接规范请求串

此步骤生成请求字符串。 将把上一步排序好的请求参数格式化成“参数名称=参数值”的形式，如对 Action 参数，其参数名称为 "Action" ，参数值为 "DescribeInstances" ，因此格式化后就为 Action=DescribeInstances 。 **注意：“参数值”为原始值而非 url 编码后的值。**

然后将格式化后的各个参数用"&"拼接在一起，最终生成的请求字符串为:

```
Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&R
```

3. 拼接待签名字符串

此步骤生成签名原文字符串。 签名原文字符串由以下几个参数构成:

1. 请求方法：支持 POST 和 GET 方式，这里使用 GET 请求，注意方法为全大写。
2. 请求主机：查看实例列表（DescribeInstances）的请求域名为： `cvm.tencentcloudapi.com` 。实际的请求域名根据接口所属模块的不同而不同，详见各接口说明。
3. 请求路径：当前版本云 API 的请求路径固定为 `/` 。
4. 请求字符串：即上一步生成的请求字符串。

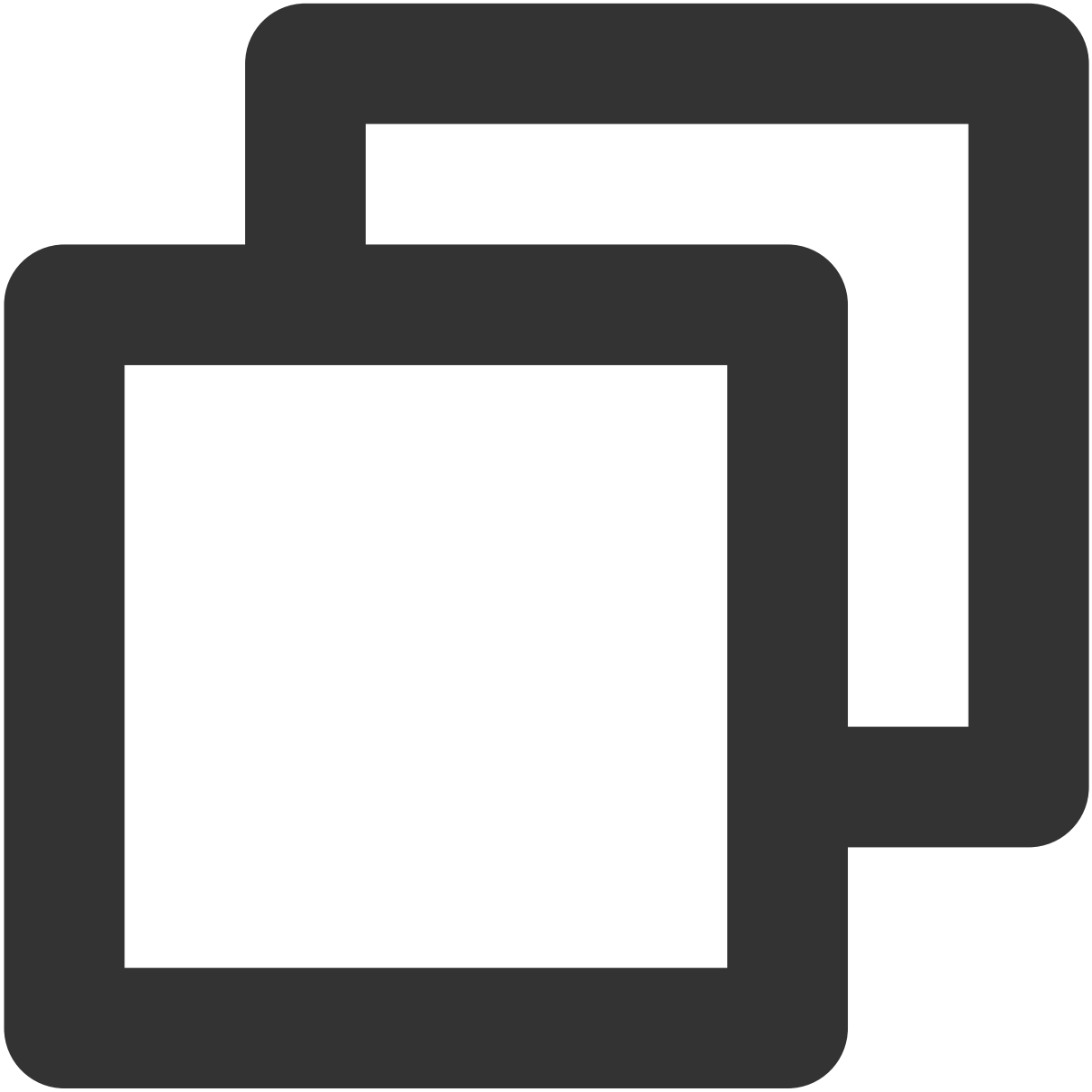
签名原文串的拼接规则为： `请求方法 + 请求主机 +请求路径 + ? + 请求字符串` 。

示例的拼接结果为：

```
GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Lim
```

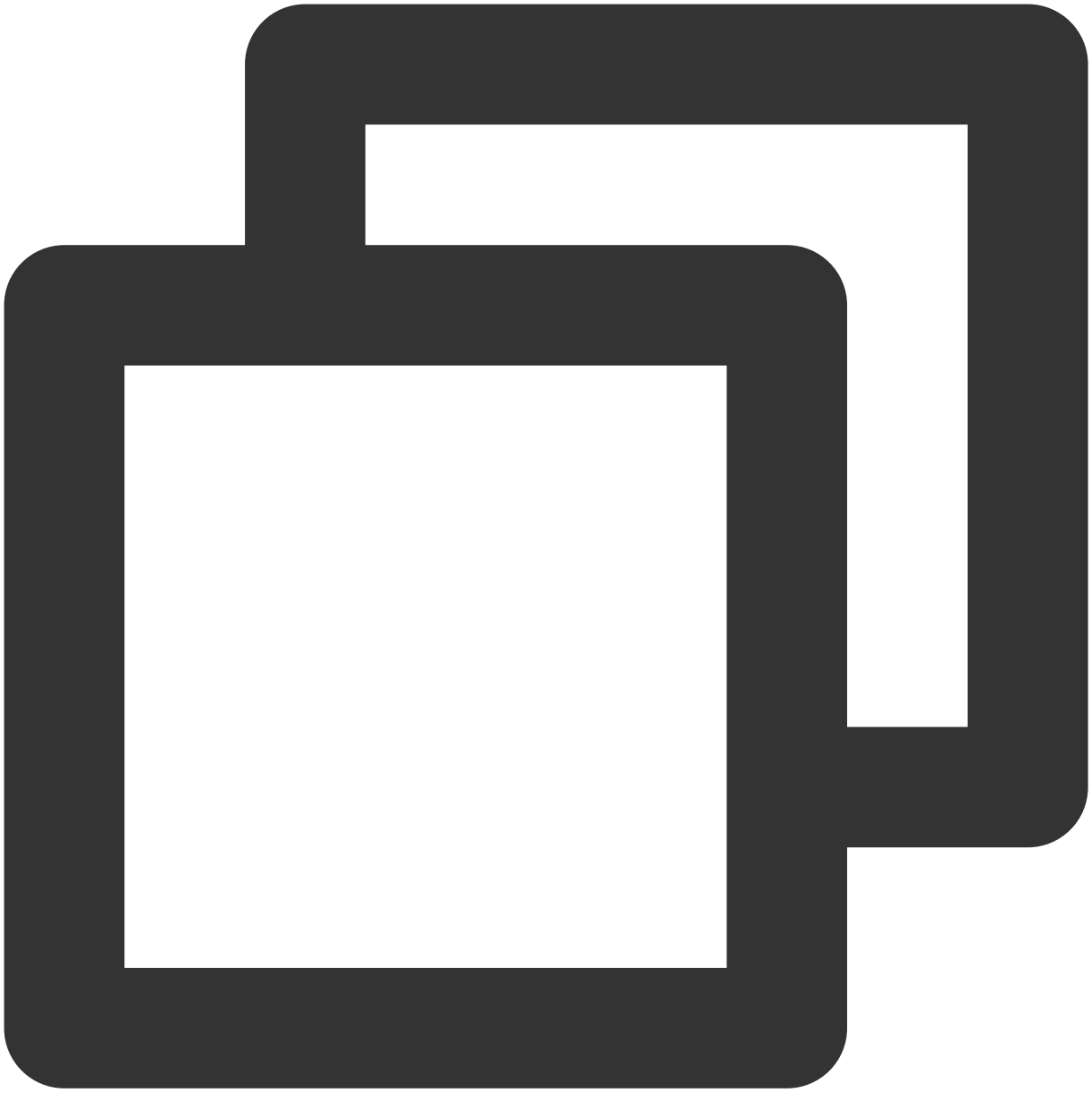
4. 计算签名（伪代码）

此步骤生成签名串。 首先使用 HMAC-SHA1 算法对上一步中获得的**签名原文字符串**进行签名，然后将生成的签名串使用 Base64 进行编码，即可获得最终的签名串。



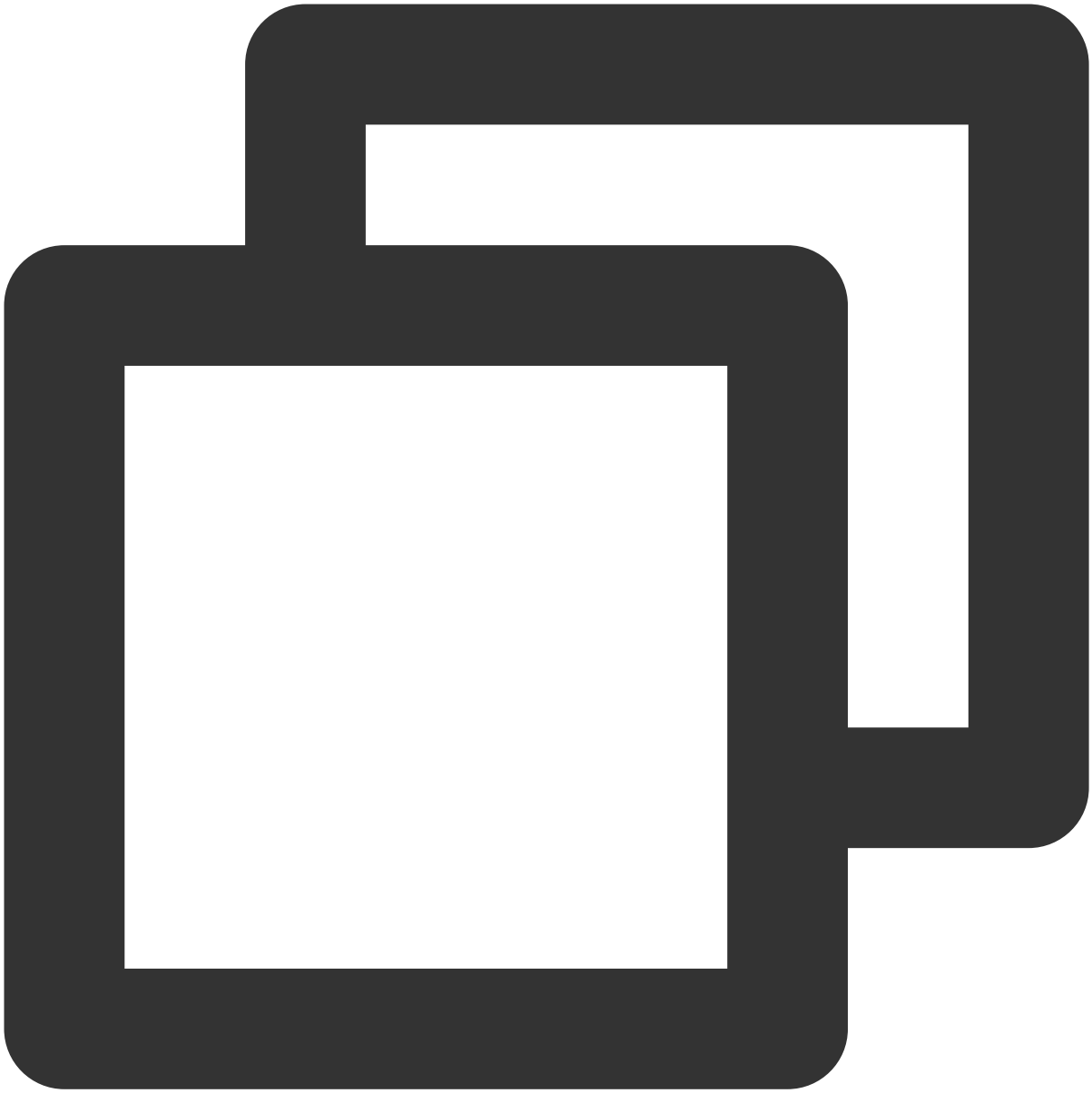
```
$secretKey = 'Gu5t9xGAR***********EXAMPLE';
$srcStr = 'GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-0
$signStr = base64_encode(hash_hmac('sha1', $srcStr, $secretKey, true));
echo $signStr;
```

最终得到的签名串为：

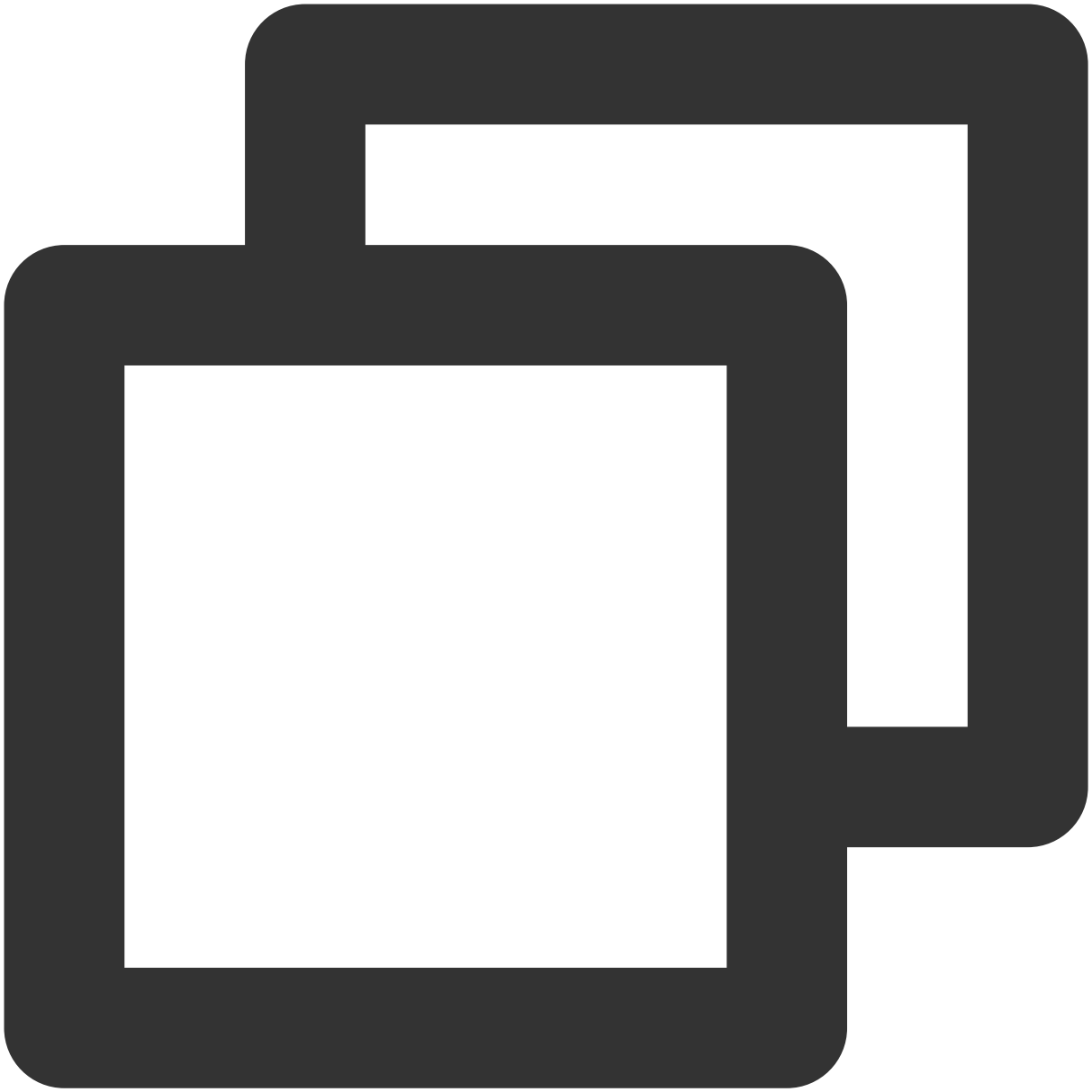


```
EliP9YW3pW28FpsEdkXt/+WcGeI=
```

5. 获取调用信息并发送请求



```
# 实际调用，成功后可能如果是消费接口会产生计费(此处以Python语言为例发送get请求)
resp = requests.get("https://" + endpoint, params=data)
# 打印发送请求的请求串
print(resp.url)
```



最终得到的请求串为
`https://cvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96d`

字段名称	解释
endpoint	服务地址， 例如： `cvm.tencentcloudapi.com`
data	API 3.0 签名 V1 所举示例接口参数，**注意这里需要将计算的签名已键值对的形式加入 data 中**

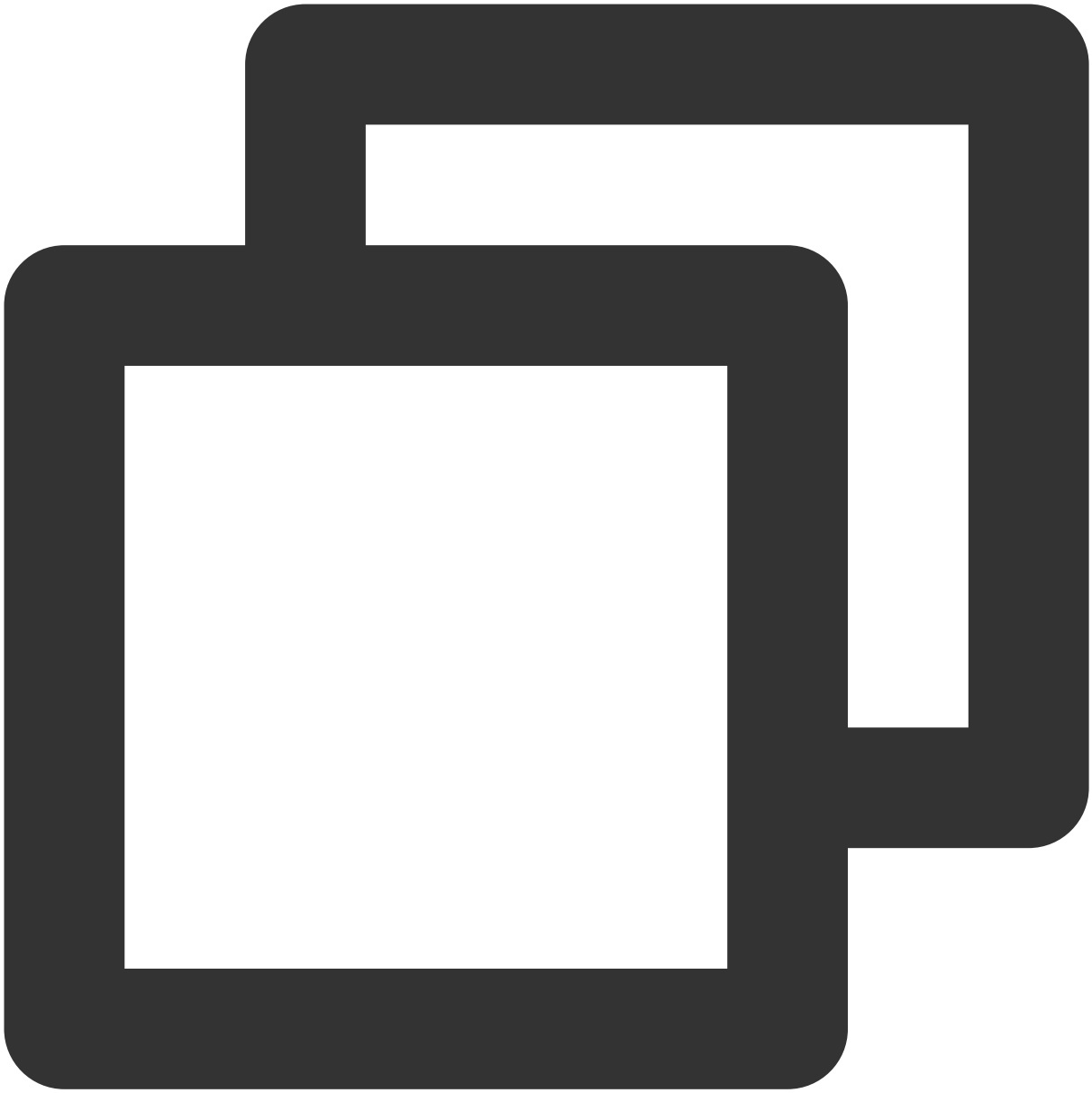
注意：

由于示例中的密钥是虚构的，时间戳也不是系统当前时间，因此如果将此 url 在浏览器中打开或者用 curl 等命令调用时会返回鉴权错误：签名过期。为了得到一个可以正常返回的 url ，需要修改示例中的 SecretId 和 SecretKey 为真实的密钥，并使用系统当前时间戳作为 Timestamp。
为了更清楚的解释签名过程，下面以PHP语言为例，将上述的签名过程具体实现。请求的域名、调用的接口和参数的取值都以上述签名过程为准，代码只为解释签名过程，并不具备通用性，实际开发请尽量使用 SDK 。

6. 签名串编码

生成的签名串并不能直接作为请求参数，需要对其进行 URL 编码。
如上一步生成的签名串为 `Eli*****************cGeI=` ，最终得到的签名串请求参数（Signature）为： `EliP**********************eI%3D` ，它将用于生成最终的请求 URL。
注意：
如果用户的请求方法是 GET，或者请求方法为 POST 同时 Content-Type 为 application/x-www-form-urlencoded，则发送请求时所有请求参数的值均需要做 URL 编码，参数键和=符号不需要编码。非 ASCII 字符在 URL 编码前需要先以 UTF-8 进行编码。
有些编程语言的网络库会自动为所有参数进行 urlencode，在这种情况下，就不需要对签名串进行 URL 编码了，否则两次 URL 编码会导致签名失败。
其他参数值也需要进行编码，编码采用 RFC 3986。使用 %XY 对特殊字符例如汉字进行百分比编码，其中“X”和“Y”为十六进制字符（0-9 和大写字母 A-F），使用小写将引发错误。

7. API 3.0 签名 V1示例



```
<?php
$secretId = "AKIDz8krbsJ5**********mLPx3EXAMPLE";
$secretKey = "Gu5t9xGAR***********EXAMPLE";
$param["Nonce"] = 11886;//rand();
$param["Timestamp"] = 1465185768;//time();
$param["Region"] = "ap-guangzhou";
$param["SecretId"] = $secretId;
$param["Version"] = "2017-03-12";
$param["Action"] = "DescribeInstances";
$param["InstanceIds.0"] = "ins-09dx96dg";
$param["Limit"] = 20;
```

```
$param["Offset"] = 0;

ksort($param);

$signStr = "GETcvm.tencentcloudapi.com/?";
foreach ( $param as $key => $value ) {
    $signStr = $signStr . $key . "=" . $value . "&";
}
$signStr = substr($signStr, 0, -1);

$signature = base64_encode(hash_hmac("sha1", $signStr, $secretKey, true));
echo $signature.PHP_EOL;
// need to install and enable curl extension in php.ini
// $param["Signature"] = $signature;
// $url = "https://cvm.tencentcloudapi.com/?".http_build_query($param);
// echo $url.PHP_EOL;
// $ch = curl_init();
// curl_setopt($ch, CURLOPT_URL, $url);
// $output = curl_exec($ch);
// curl_close($ch);
// echo json_decode($output);
```

API 2.0 签名

此签名现已不在维护，建议使用性能更优的 **API 3.0 签名**，如需使用，请直接在 API Explorer 的**签名串生成>选择 API 2.0 签名版本**进行操作。

签名失败

存在以下签名失败的错误码，请根据实际情况处理。

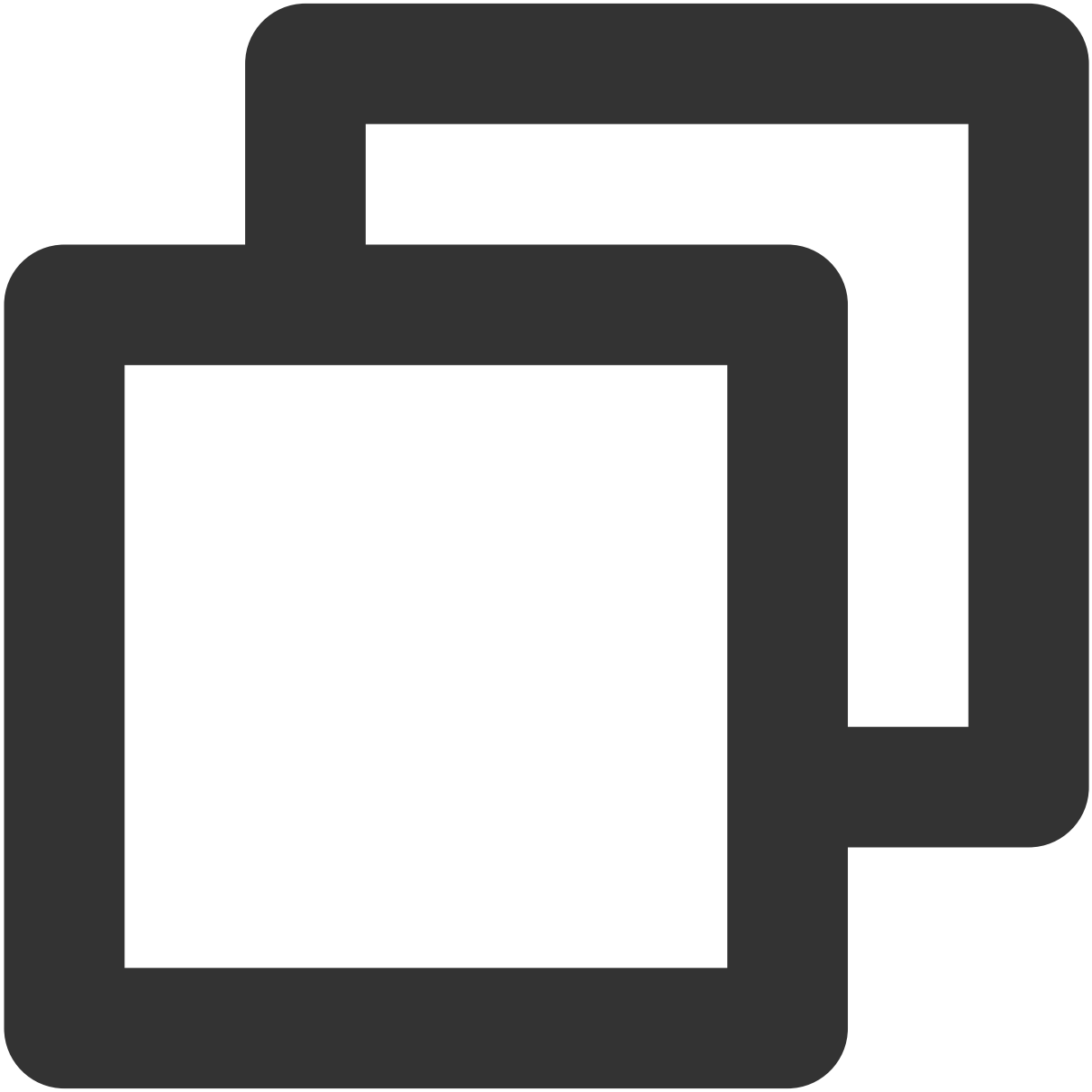
错误码	错误描述
AuthFailure.SignatureExpire	签名过期。Timestamp 与服务器接收到请求的时间相差不得超过五分钟。
AuthFailure.SecretIdNotFound	密钥不存在。请到控制台查看密钥是否被禁用，是否少复制了字符或者多了字符。
AuthFailure.SignatureFailure	签名错误。可能是签名计算错误，或者签名与实际发送的内容不相符合，也有可能是密钥 SecretKey 错误导致的。
AuthFailure.TokenFailure	临时证书 Token 错误。

AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。

返回结果

正确返回结果

以云服务器的接口查看实例列表 (DescribeInstances) 2017-03-12 版本为例，若调用成功，其可能的返回如下为：



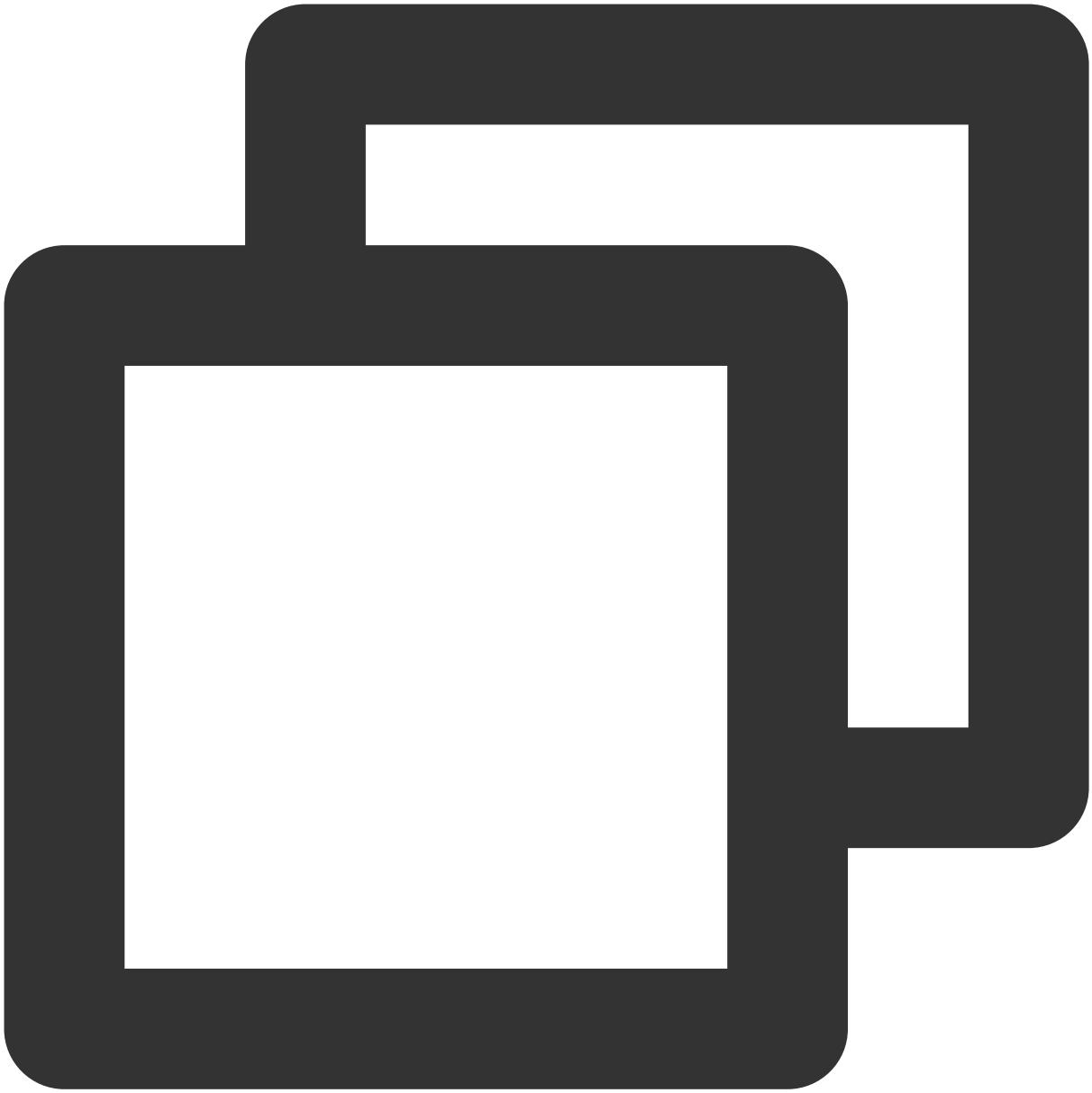
```
{
    "Response": {
```

```
        "TotalCount": 0,
        "InstanceStatusSet": [],
        "RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
    }
}
```

Response 及其内部的 RequestId 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。RequestId 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。
除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 TotalCount 和 InstanceStatusSet 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 TotalCount 在此情况下的返回值为 0， InstanceStatusSet 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：




```
{
    "Response": {
        "Error": {
            "Code": "AuthFailure.SignatureFailure",
            "Message": "The provided credentials could not be validated. Please che
        },
        "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
    }
}
```

`Error` 的出现代表着该请求调用失败。 `Error` 字段连同其内部的 `Code` 和 `Message` 字段在调用失败时是必定返回的。

`Code` 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。

`Message` 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。

`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。 Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。
LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。

版权所有：腾讯云计算（北京）有限责任公司

ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	HTTPS 请求方法错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

版权所有：腾讯云计算（北京）有限责任公司

Python API

最近更新时间：2023-03-07 18:16:40

腾讯云 API 全新升级3.0，该版本进行了性能优化且全地域部署、支持就近和按地域接入、访问时延下降显著，接口描述更加详细、错误码描述更加全面、SDK 增加接口级注释，让您更加方便快捷的使用腾讯云产品。这里针对 Python API 调用方式进行简单说明。
现已支持云服务器（CVM）、云硬盘（CBS）、私有网络（VPC）、云数据库（TencentDB）等 腾讯云产品，后续会支持其他的云产品接入，敬请期待。

了解请求结构

1. 服务地址（endpoint）

API 支持就近地域接入（例如：cvm 产品域名为 `cvm.tencentcloudapi.com`），也支持指定地域域名访问（例如：广州地域的域名为 `cvm.ap-guangzhou.tencentcloudapi.com`），各地域参数见下文公共参数中的地域列表，详情请参见各产品的“请求结构”文档说明判断是否支持该地域。

注意：

对时延敏感的业务，建议指定带地域的域名。

2. 通信协议

腾讯云 API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法：
POST（推荐）
GET
POST 请求支持的 Content-Type 类型：
application/json（推荐），必须使用签名方法 v3（TC3-HMAC-SHA256）。
application/x-www-form-urlencoded，必须使用签名方法 v1（HmacSHA1 或 HmacSHA256）。
multipart/form-data（仅部分接口支持），必须使用签名方法 v3（TC3-HMAC-SHA256）。
GET 请求的请求包大小不得超过32KB。POST 请求使用签名方法 v1（HmacSHA1、HmacSHA256）时不得超过1MB。POST 请求使用签名方法 v3（TC3-HMAC-SHA256）时支持10MB。

4. 字符编码

均使用 UTF-8 编码。

版权所有：腾讯云计算（北京）有限责任公司

公共参数

说明：
公共参数是用于标识用户和接口签名的参数，每次请求均需要携带这些参数，才能正常发起请求。

签名方法 V3 公共参数

签名方法 v3 （有时也称作 TC3-HMAC-SHA256）相比签名方法 v1 （有些文档可能会简称“签名方法”），更安全，支持更大的请求包，支持 POST JSON 格式，性能有一定提升，推荐使用该签名方法计算签名。使用方法见下文 “签名方法介绍”。

参数名称	类型	必选	描述
X-TC-Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
X-TC-Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如 1529223702。**注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。**
X-TC-Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
Authorization	String	是	HTTP 标准身份认证头部字段，例如： TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=72e494ea8**********a96525168 其中： TC3-HMAC-SHA256：签名方法，目前固定取该值。 Credential：签名凭证，AKIDEXAMPLE 是 SecretId。 Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致。 service 为产品名，通常为域名前缀，例如域名 cvm.tencentcloudapi.com 意味着产品名是 cvm。本产品取值为 cvm。 SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部。 Signature：签名摘要，计算过程详见下文。
X-TC-Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

签名方法 V1 公共参数

使用签名方法 v1 （有时会称作 HmacSHA256 和 HmacSHA1），公共参数需要统一放到请求串中。

参数名称	类型	必选	描述
Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
Region	String	-	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。**注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。**
Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如 1529223702，如果与当前时间相差过大，会引起签名过期错误。
Nonce	Integer	是	随机正整数，与 Timestamp 联合起来，用于防止重放攻击。
SecretId	String	是	在 云API密钥 上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey，而 SecretKey 会用来生成请求签名 Signature。
Signature	String	是	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。具体计算方法参见下文“签名方法介绍”。
Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
SignatureMethod	String	否	签名方式，目前支持 HmacSHA256 和 HmacSHA1。只有指定此参数为 HmacSHA256 时，才使用 HmacSHA256 算法验证签名，其他情况均使用 HmacSHA1 验证签名。
Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token 。

地域列表

由于各个产品支持地域不同，具体详情请参考各产品文档中的地域列表。
例如，您可以参考云服务器的 地域列表。

Python API 调用方式

腾讯云 API 会对每个请求进行身份验证，用户需要使用安全凭证，经过特定的步骤对请求进行签名（Signature），每个请求都需要在公共请求参数中指定该签名结果并以指定的方式和格式发送请求。

假设用户的 SecretId 和 SecretKey 分别是： `AKIDz8krbsJ5*********mLPx3EXAMPLE` 和 `Gu5t9xGAR**********EXAMPLE` 。用户想查看广州区云服务器名为 “未命名” 的主机状态，只返回一条数据。则请求可能为：

```
curl -X POST https://cvm.tencentcloudapi.com \\
-H "Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5*********mLPx3EXAMPLE/20
-H "Content-Type: application/json; charset=utf-8" \\
-H "Host: cvm.tencentcloudapi.com" \\
```

```
-H "X-TC-Action: DescribeInstances" \\
-H "X-TC-Timestamp: 1551113065" \\
-H "X-TC-Version: 2017-03-12" \\
-H "X-TC-Region: ap-guangzhou" \\
-d '{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instanc
```

步骤1：申请安全凭证

本文使用的安全凭证为密钥，密钥包括 SecretId 和 SecretKey。每个用户最多可以拥有两对密钥。

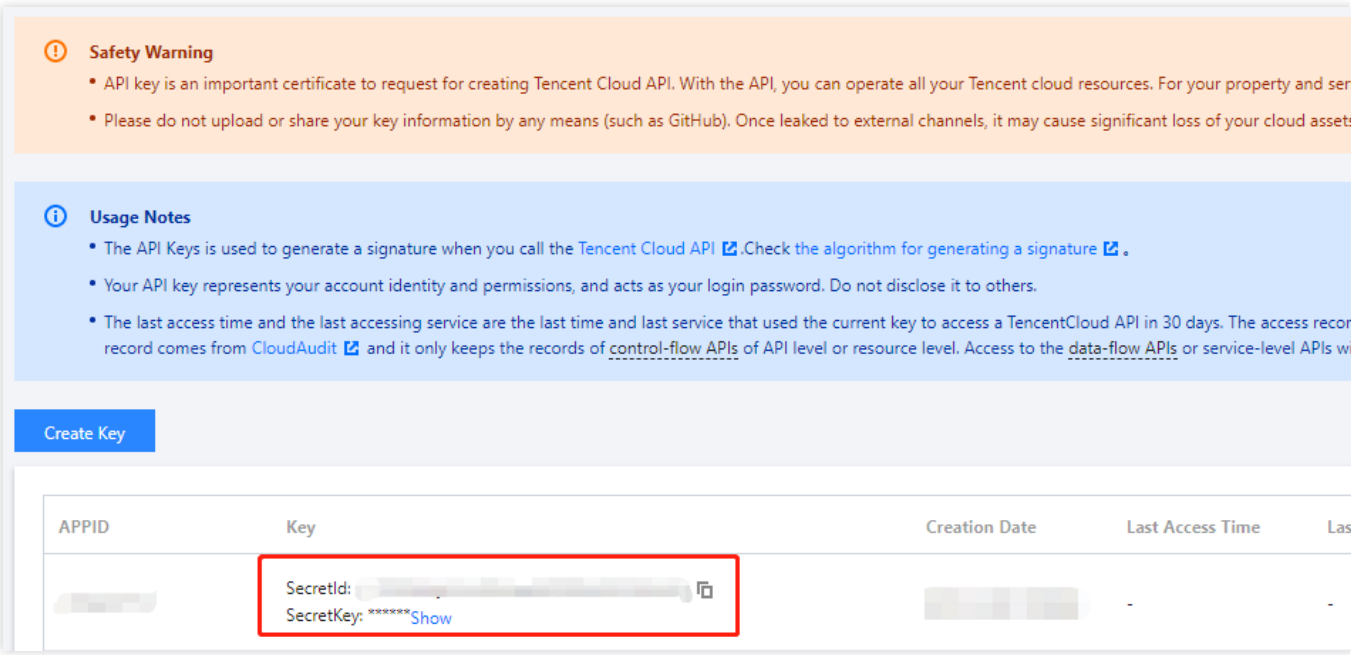
SecretId：用于标识 API 调用者身份，可以简单类比为用户名。

SecretKey：用于验证 API 调用者的身份，可以简单类比为密码。

注意：

用户必须严格保管安全凭证，避免泄露，否则将危及财产安全。如已泄漏，请立刻禁用该安全凭证。

前往 API 密钥管理 页面，即可进行获取。如下图所示：



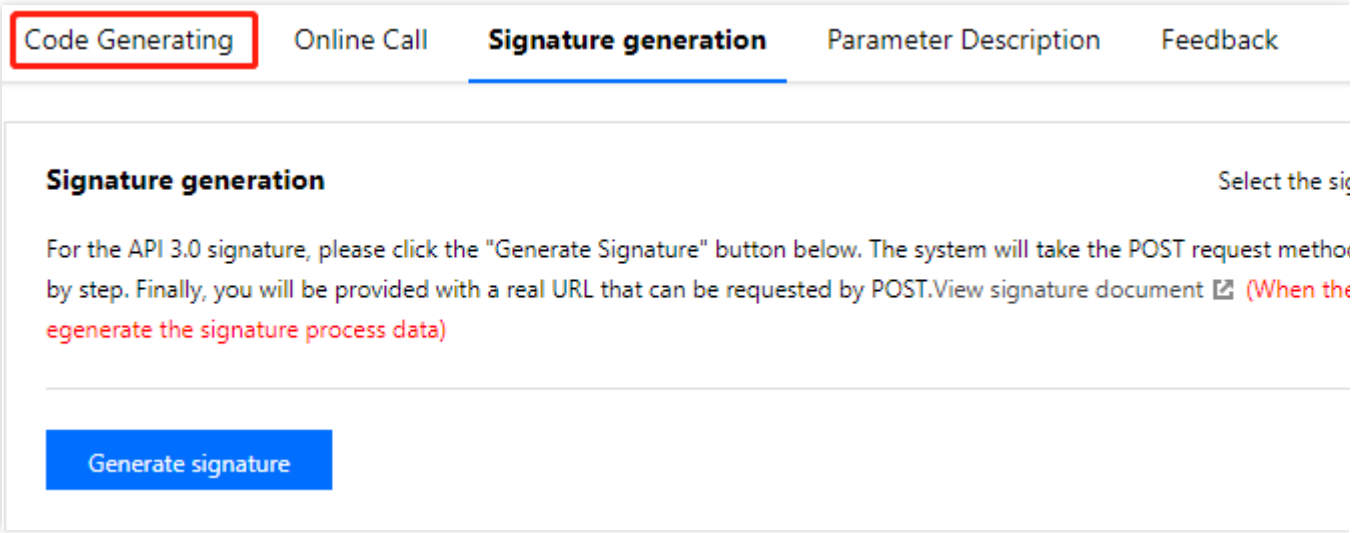
步骤2

1. 获取 API 3.0 V3 版本签名

签名方法 v3 （TC3-HMAC-SHA256）功能上覆盖了以前的签名方法 v1，而且更安全，支持更大的请求，支持 json 格式，性能有一定提升，推荐使用该签名方法计算签名。如下图所示：

说明：

首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为 “API 3.0 签名 v3”，可以生成签名过程进行验证，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的7种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

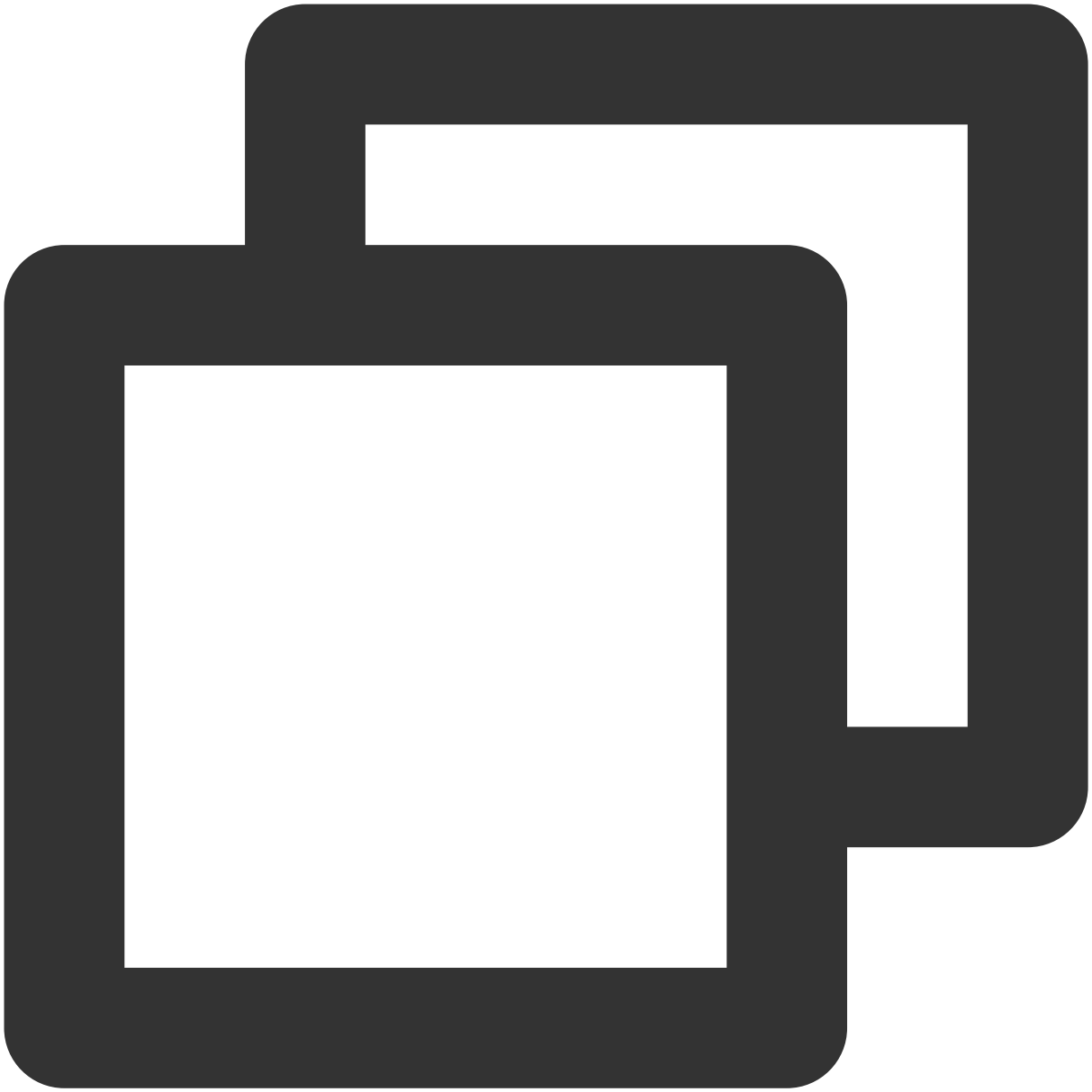


云 API 支持 GET 和 POST 请求。对于 GET 方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。对于 POST 方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式绝大多数接口均支持，multipart 格式只有特定接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。推荐使用 POST 请求，因为两者的结果并无差异，但 GET 请求只支持32KB以内的请求包。

下面以 查看实例列表 接口为例，分步骤介绍签名的计算过程。我们选择该接口原因是：

1. 云服务器默认已开通，该接口很常用。
2. 该接口是只读的，不会改变现有资源的状态。
3. 接口覆盖的参数种类较全，可以演示包含数据结构的数组如何使用。

1. 拼接规范请求串

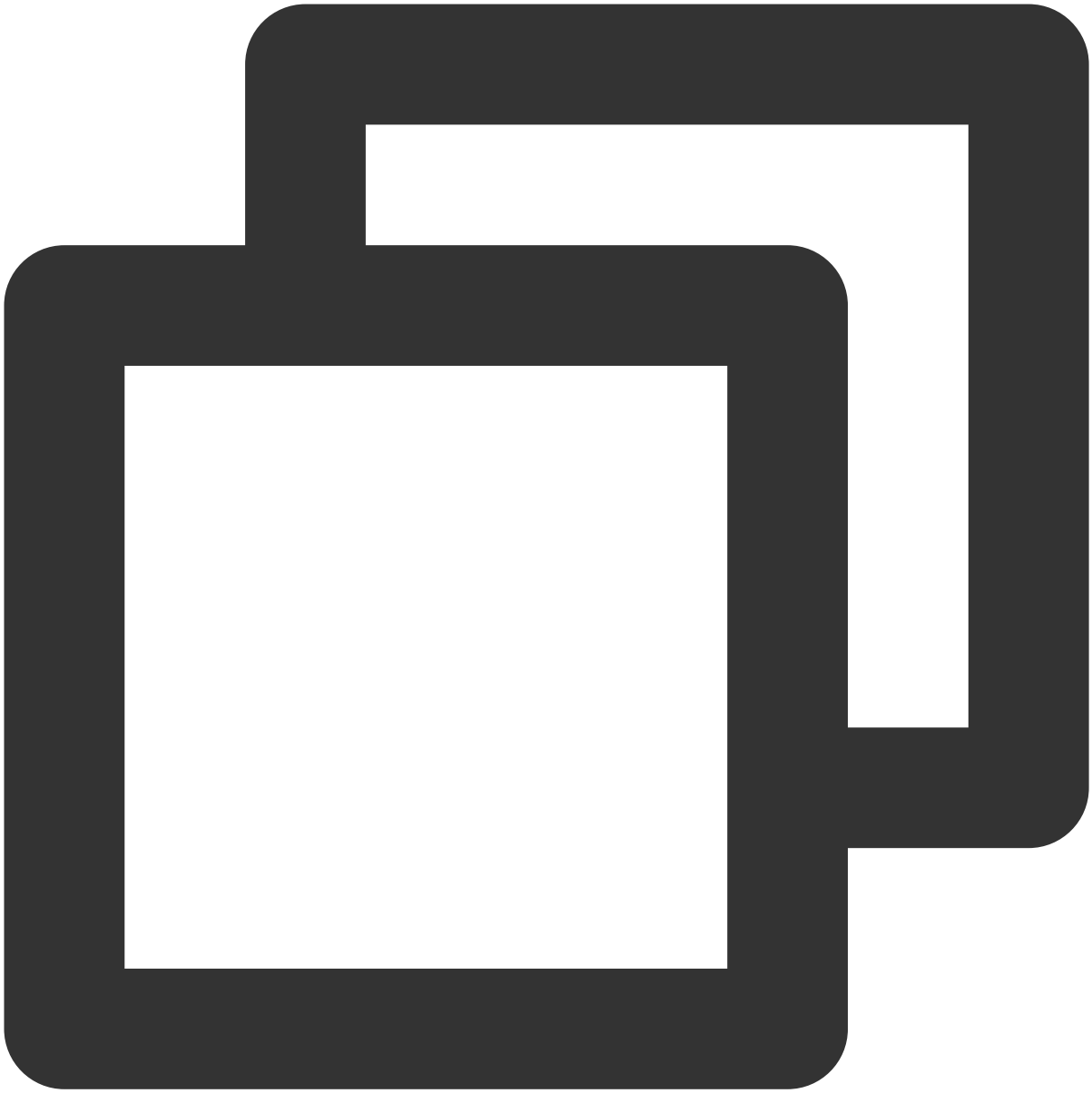


```
CanonicalRequest =
    HTTPRequestMethod + '\\n' +
    CanonicalURI + '\\n' +
    CanonicalQueryString + '\\n' +
    CanonicalHeaders + '\\n' +
    SignedHeaders + '\\n' +
    HashedRequestPayload
```

字段名称	解释

HTTPRequestMethod	HTTP 请求方法（GET、POST ）。此示例取值为 `POST` 。
CanonicalURI	URI 参数，API 3.0 固定为正斜杠（/）。
CanonicalQueryString	发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串""，对于 G 求，则为 URL 中问号（?）后面的字符串内容，例如：Limit=10&Offset=0。 注意：CanonicalQueryString 需要参考 RFC3986 进行 URLEncode，字符集 UTF8，推荐使用 语言标准库，所有特殊字符均需编码，大写形式。
CanonicalHeaders	参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部 签名以提高自身请求的唯一性和安全性。 拼接规则：头部 key 和 value 统一转成小写，掉首尾空格，按照 key:value\\n 格式拼接；多个头部，按照头部 key（小写）的 ASCII 行拼接。此示例计算结果是 `content-type:application/json; charset=ut 8\\nhost:cvm.tencentcloudapi.com\\n` 。 注意：content-type 必须和实际发 符合，有些编程语言网络库即使未指定也会自动添加 charset 值，如果签名时和发送时 致，服务器会返回签名校验失败。
SignedHeaders	参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包 部内容是一一对应的。content-type 和 host 为必选头部。 拼接规则：头部 key 统一转 写；多个头部 key（小写）按照 ASCII 升序进行拼接，并且以分号（;）分隔。此示例 content-type;host
HashedRequestPayload	请求正文（Requestpayload，即 body，此示例为 `{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-name"` 的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(RequestPayload))), HTTP 请求正文做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。 GET 请求，RequestPayload 固定为空字符串。此示例计算结果是 `35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f`

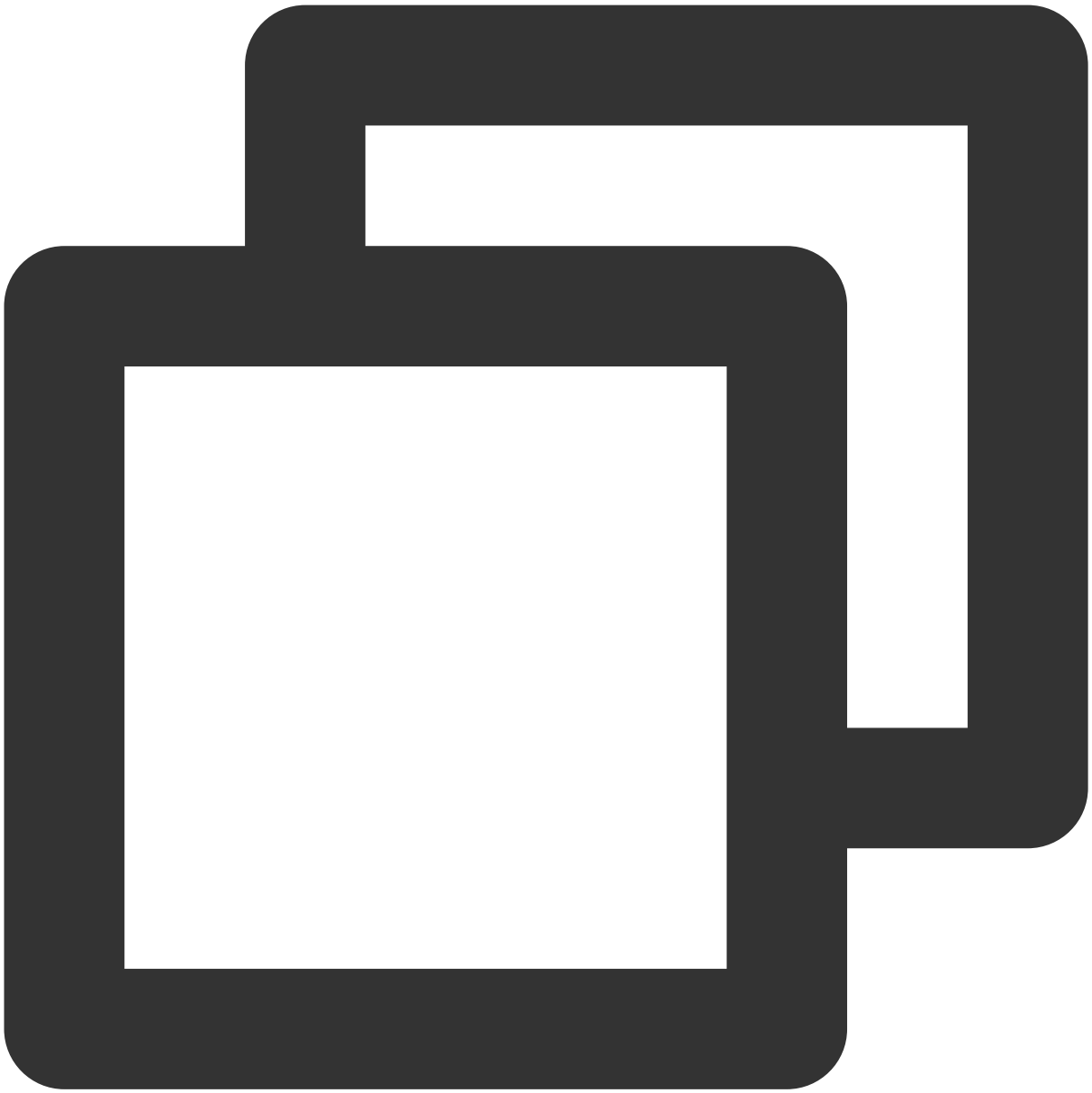
根据以上规则，示例中得到的规范请求串如下：



```
POST
/

content-type:application/json; charset=utf-8
host:cvm.tencentcloudapi.com

content-type;host
35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f064
```

2. 拼接待签名字符串

按如下格式拼接待签名字符串：



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

字段名称	解释
Algorithm	签名算法，目前固定为 `TC3-HMAC-SHA256` 。

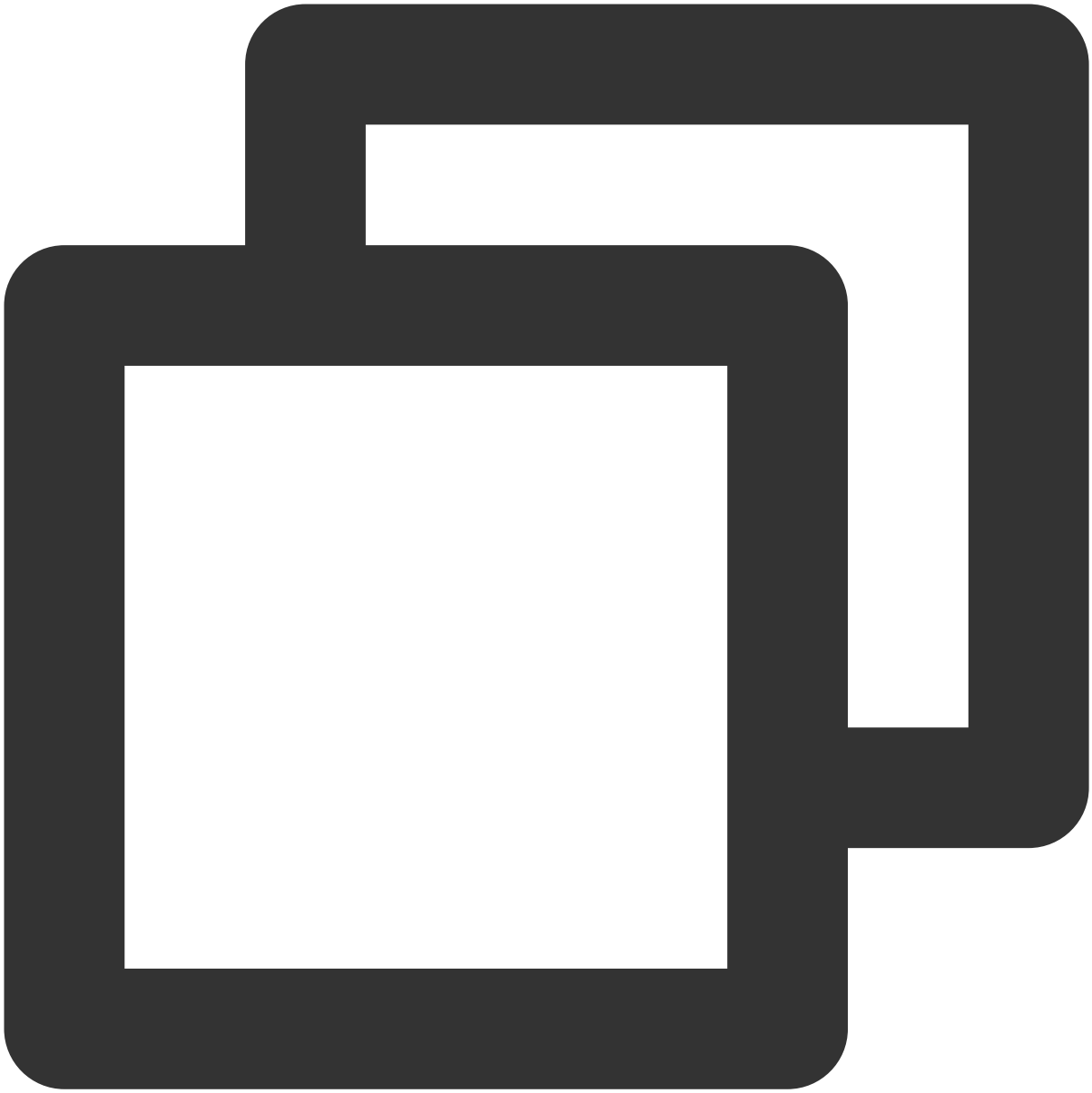
RequestTimestamp	请求时间戳，即请求头部的公共参数 X-TC-Timestamp 取值，取当前时间 UNIX 时间确到秒。此示例取值为 `1551113065` 。
CredentialScope	凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符串（tc3_request）。**Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Time 换算的 UTC 标准时间日期一致**；service 为产品名，必须与调用的产品域名一致。此算结果是 2019-02-25/cvm/tc3_request。
HashedCanonicalRequest	前述步骤拼接所得规范请求串的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))。此示例计算结果是 `5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d`

注意：

Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为 1551113065，在东八区的时间是 2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为 2019-02-25，而不是 2019-02-26。

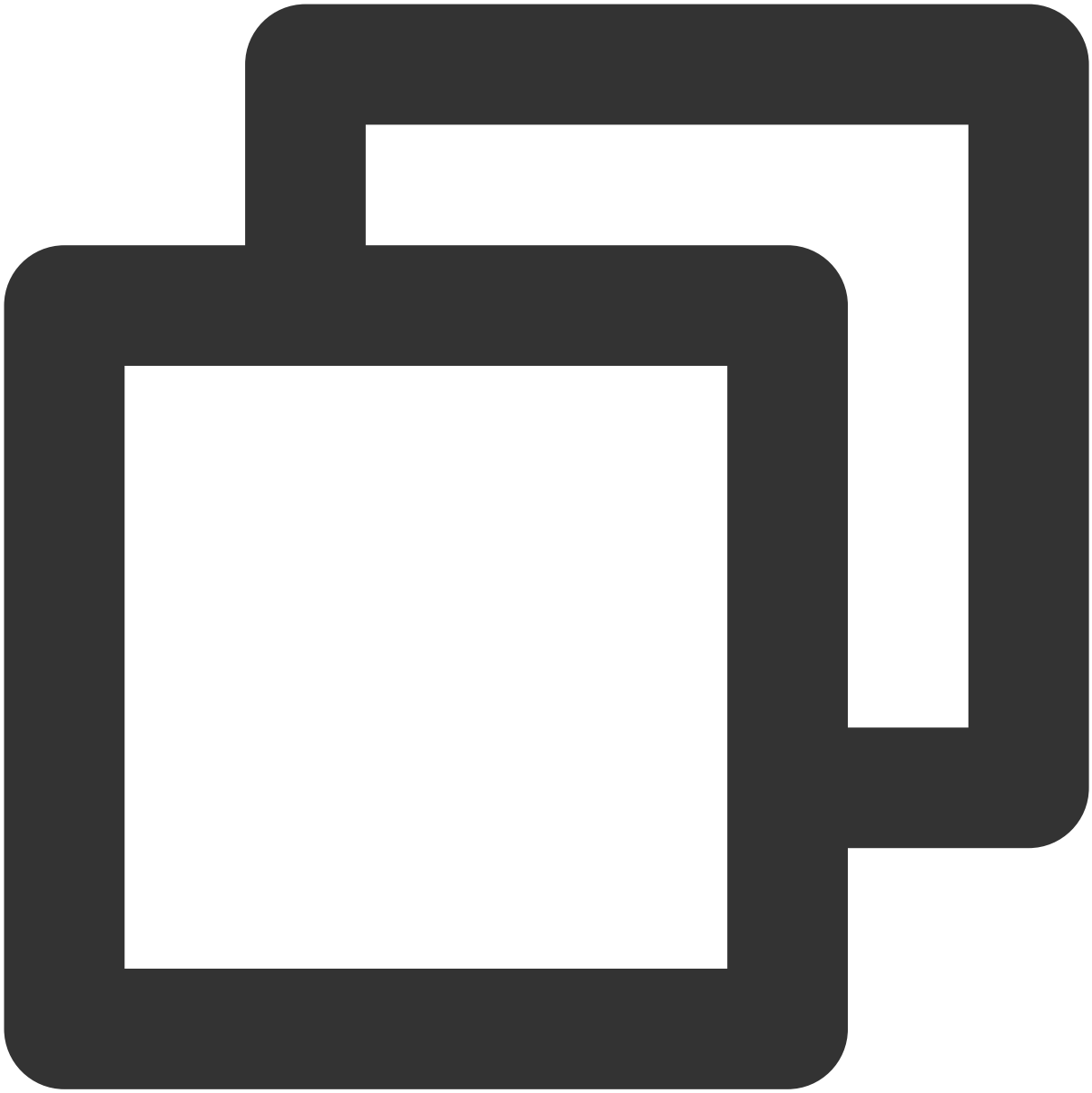
Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败，返回签名过期错误。

根据以上规则，示例中得到的待签名字符串如下：



```
TC3-HMAC-SHA256
1551113065
2019-02-25/cvm/tc3_request
5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d7031
```

3. 计算签名(伪代码)

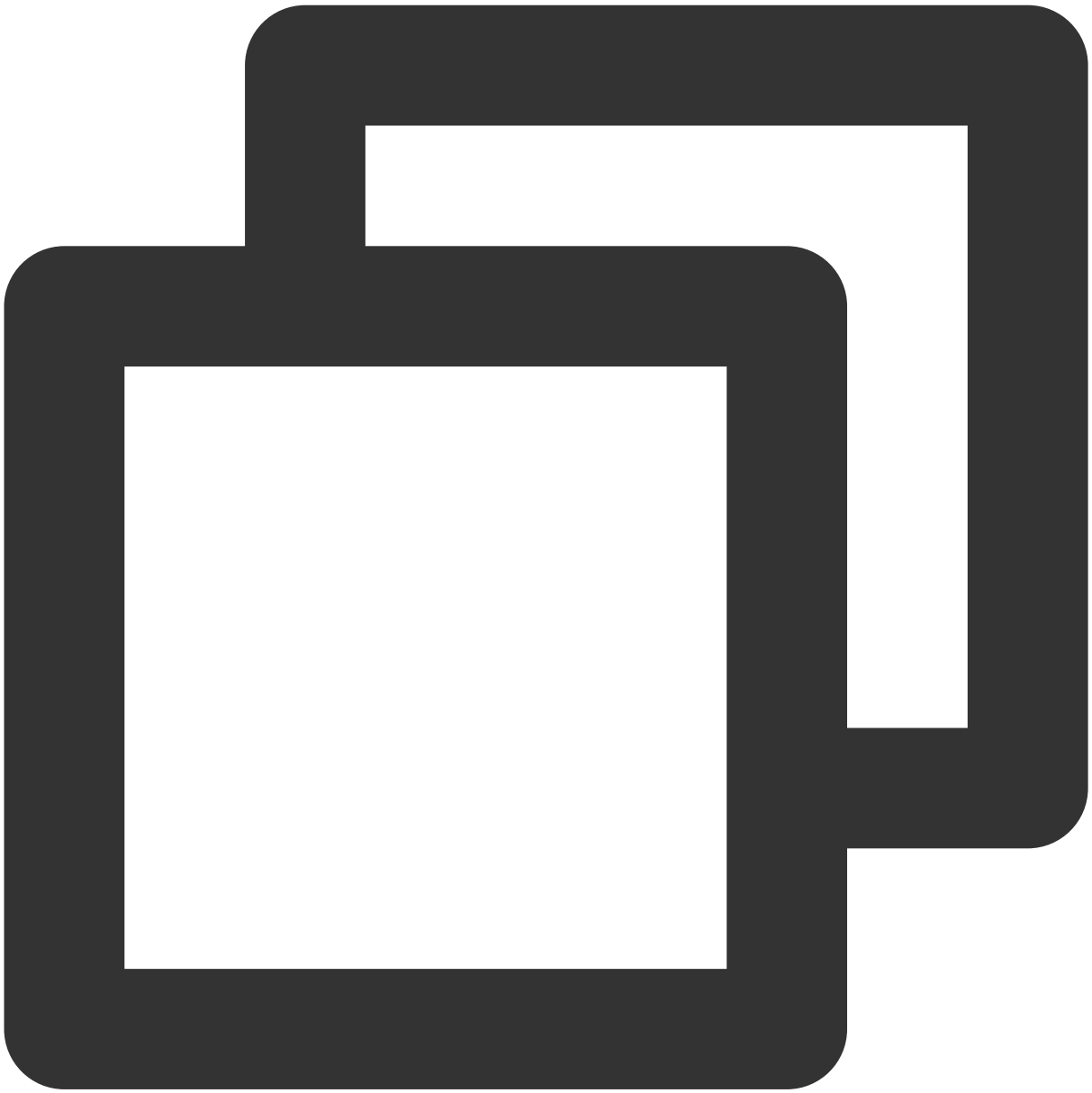


```
SecretKey = "Gu5t9xGARN**********QYCN3EXAMPLE"
SecretDate = HMAC_SHA256("TC3" + SecretKey, Date)
SecretService = HMAC_SHA256(SecretDate, Service)
SecretSigning = HMAC_SHA256(SecretService, "tc3_request")
```

字段名称	解释
SecretKey	原始的 SecretKey，即 `Gu5t9xGARN**********QYCN3EXAMPLE` 。
Date	即 Credential 中的 Date 字段信息。此示例取值为 `2019-02-25` 。

Service	即 Credential 中的 Service 字段信息。此示例取值为 `cvm` 。

签名结果如下：

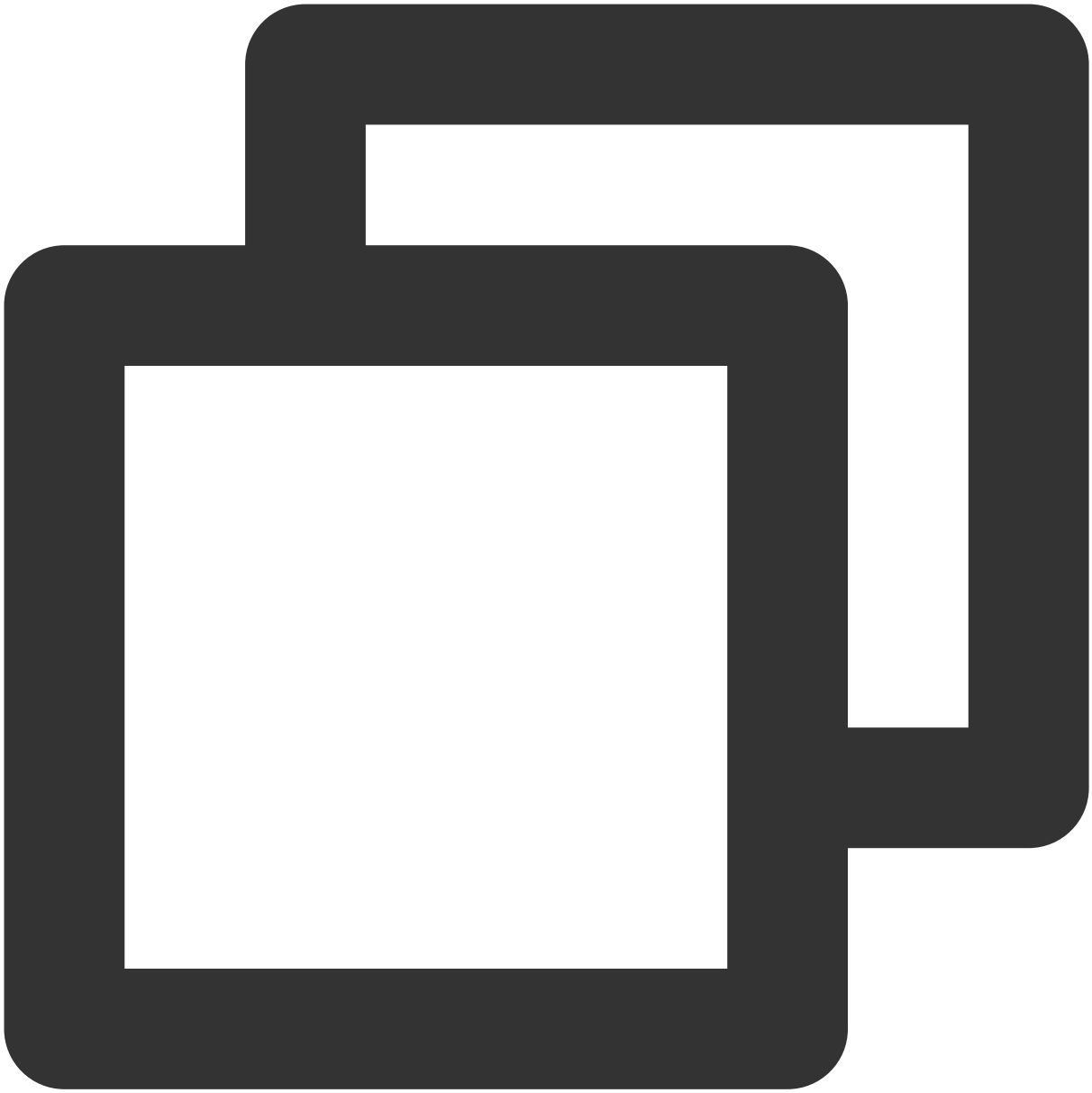


```
Signature = HexEncode(HMAC_SHA256(SecretSigning, StringToSign))
```

此示例计算结果是 `72e494ea8*****************************c5a96525168` 。

4. 获取调用信息

按如下格式拼接 Authorization：



```
Authorization =
    Algorithm + ' ' +
    'Credential=' + SecretId + '/' + CredentialScope + ', ' +
    'SignedHeaders=' + SignedHeaders + ', ' +
    'Signature=' + Signature
```

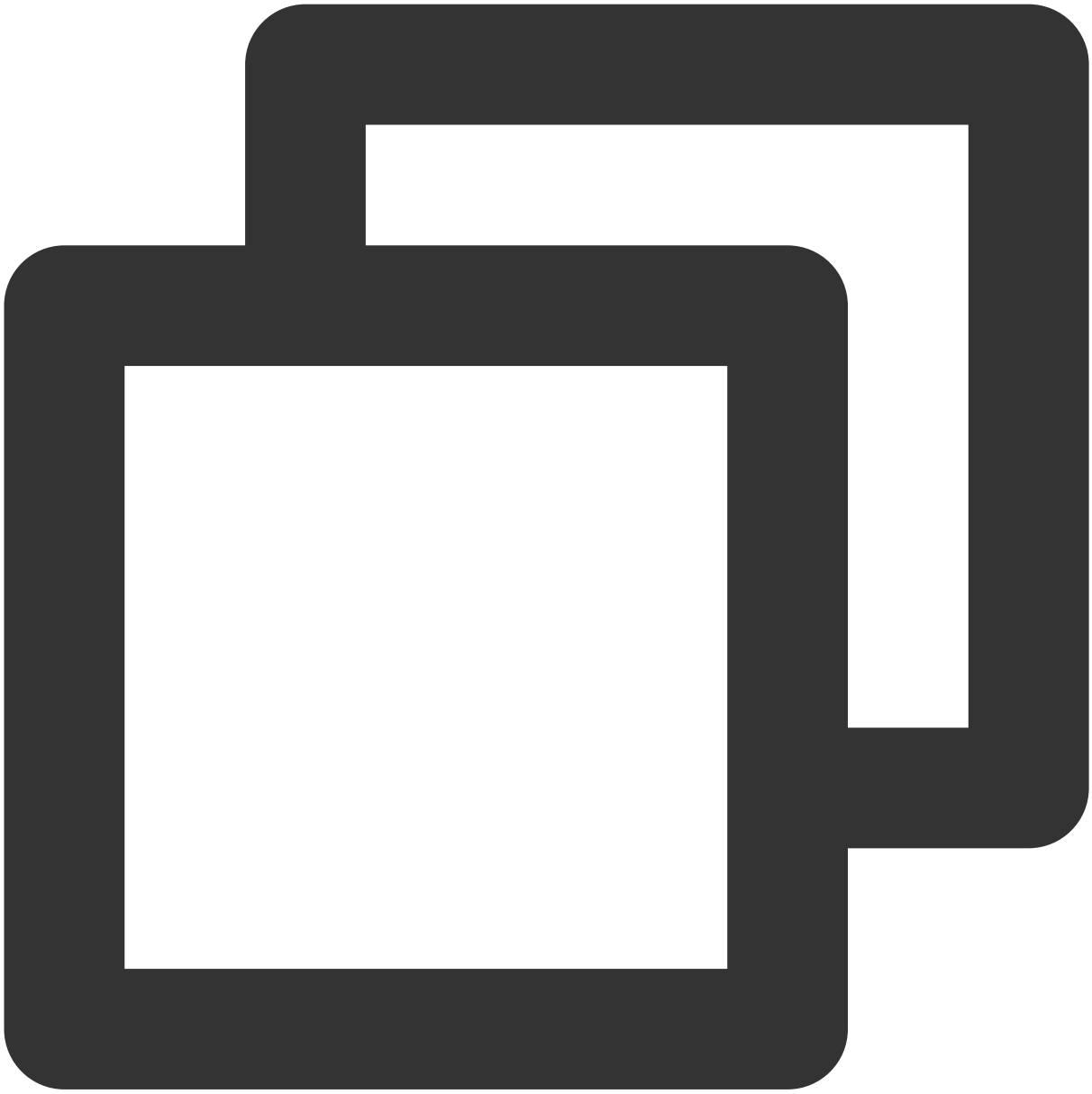
字段名称	解释
Algorithm	签名方法，固定为 `TC3-HMAC-SHA256` 。
SecretId	

	密钥对中的 SecretId，即 `AKIDz8krbsJ5yK*********mLPx3EXAMPLE` 。
CredentialScope	见上文，凭证范围。此示例计算结果是 `2019-02-25/cvm/tc3_request` 。
SignedHeaders	见上文，参与签名的头部信息。此示例取值为 `content-type;host` 。
Signature	签名值。此示例计算结果是 `72e494ea809ad7a8c8f7a450**************f516e8da2f66e2c5a96525168` 。

根据以上规则，示例中得到的值为：

```
TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5yK**********mLPx3EXAMPLE/2019-02-25/cvm/tc3_
```

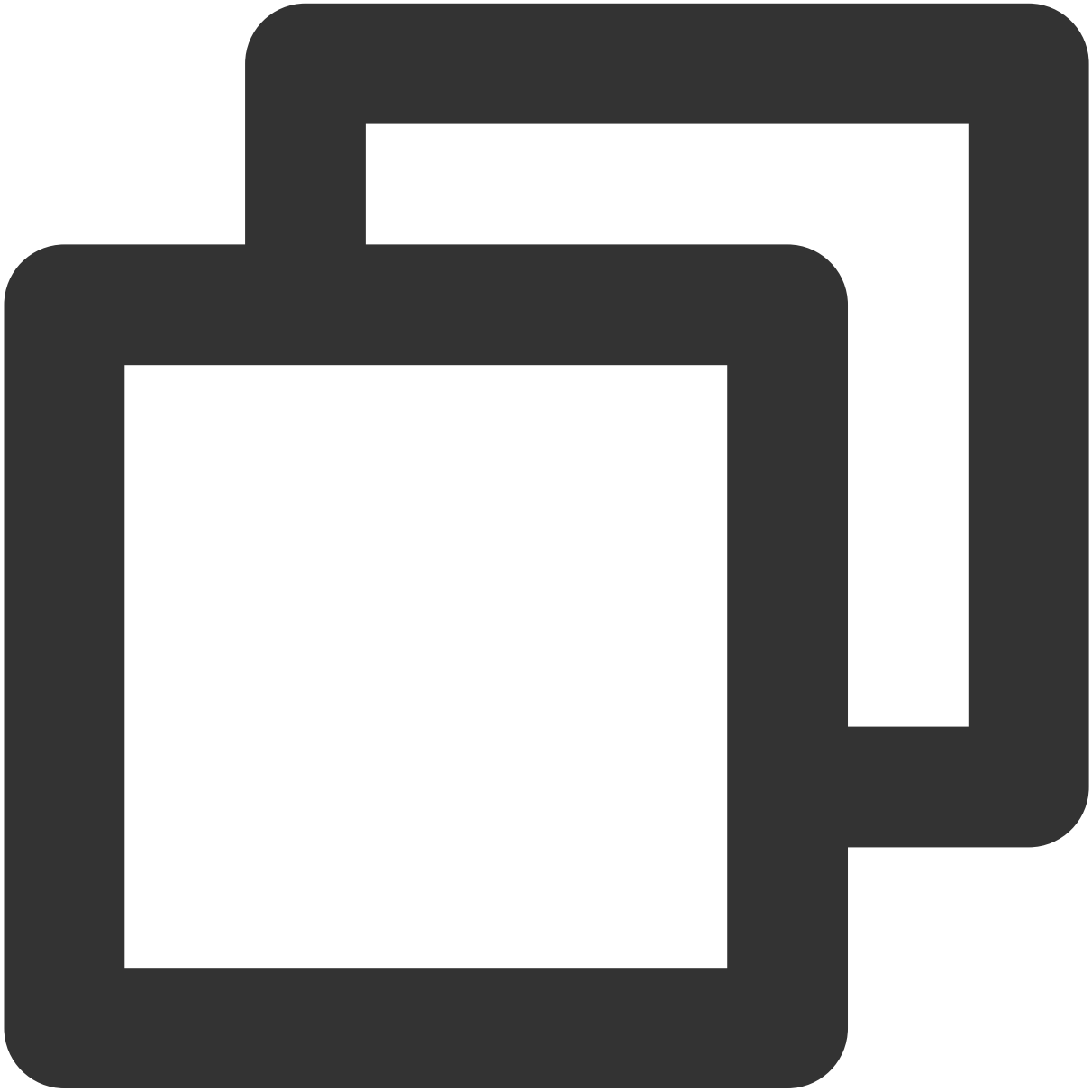
最终完整的调用信息如下：



```
POST https://cvm.tencentcloudapi.com/
Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5yK**********mLPx3EXAMPLE/2019
Content-Type: application/json; charset=utf-8
Host: cvm.tencentcloudapi.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1551113065
```

```
X-TC-Region: ap-guangzhou

{"Limit": 1, "Filters": [{"Values": ["\\u672a\\u547d\\u540d"], "Name": "instance-na
```

5. API 3.0 签名 V3 示例



```
# -*- coding: utf-8 -*-
import hashlib, hmac, json, os, sys, time
from datetime import datetime

# 密钥参数
```

```
secret_id = "`AKIDz8krbsJ5yK*********mLPx3EXAMPLE`"
secret_key = "`Gu5t9xGARN*********QYCN3EXAMPLE`"

service = "cvm"
host = "cvm.tencentcloudapi.com"
endpoint = "https://" + host
region = "ap-guangzhou"
action = "DescribeInstances"
version = "2017-03-12"
algorithm = "TC3-HMAC-SHA256"
#timestamp = int(time.time())
timestamp = 1551113065
date = datetime.utcfromtimestamp(timestamp).strftime("%Y-%m-%d")
params = {"Limit": 1, "Filters": [{"Name": "instance-name", "Values": [u"未命名"]}]}

# ************* 步骤 1：拼接规范请求串 *************
http_request_method = "POST"
canonical_uri = "/"
canonical_querystring = ""
ct = "application/json; charset=utf-8"
payload = json.dumps(params)
canonical_headers = "content-type:%s\\nhost:%s\\n" % (ct, host)
signed_headers = "content-type;host"
hashed_request_payload = hashlib.sha256(payload.encode("utf-8")).hexdigest()
canonical_request = (http_request_method + "\\n" +
                     canonical_uri + "\\n" +
                     canonical_querystring + "\\n" +
                     canonical_headers + "\\n" +
                     signed_headers + "\\n" +
                     hashed_request_payload)
print(canonical_request)

# ************* 步骤 2：拼接待签名字符串 *************
credential_scope = date + "/" + service + "/" + "tc3_request"
hashed_canonical_request = hashlib.sha256(canonical_request.encode("utf-8")).hexdig
string_to_sign = (algorithm + "\\n" +
                  str(timestamp) + "\\n" +
                  credential_scope + "\\n" +
                  hashed_canonical_request)
print(string_to_sign)

# ************* 步骤 3：计算签名 *************
# 计算签名摘要函数
def sign(key, msg):
    return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()
secret_date = sign(("TC3" + secret_key).encode("utf-8"), date)
secret_service = sign(secret_date, service)
```

```
secret_signing = sign(secret_service, "tc3_request")
signature = hmac.new(secret_signing, string_to_sign.encode("utf-8"), hashlib.sha256
print(signature)

# ************* 步骤 4：拼接 Authorization *************
authorization = (algorithm + " " +
                 "Credential=" + secret_id + "/" + credential_scope + ", " +
                 "SignedHeaders=" + signed_headers + ", " +
                 "Signature=" + signature)
print(authorization)

print('curl -X POST ' + endpoint
      + ' -H "Authorization: ' + authorization + '"'
      + ' -H "Content-Type: application/json; charset=utf-8"'
      + ' -H "Host: ' + host + '"'
      + ' -H "X-TC-Action: ' + action + '"'
      + ' -H "X-TC-Timestamp: ' + str(timestamp) + '"'
      + ' -H "X-TC-Version: ' + version + '"'
      + ' -H "X-TC-Region: ' + region + '"'
      + " -d '" + payload + "'")
```

2. 获取 API 3.0 V1 版本签名

签名方法 v1（HmacSHA1、HmacSHA256）简单易用，但是功能和安全性都不如签名方法 v3，推荐使用签名方法 v3。

首次接触，建议使用 API Explorer 中的“签名串生成”功能，选择签名版本为“API 3.0 签名 v1”，可以生成签名过程进行验证，并提供了部分编程语言的签名示例，也可直接生成 SDK 代码。推荐使用腾讯云 API 配套的 7 种常见的编程语言 SDK，已经封装了签名和请求过程，均已开源，支持 Python、Java、PHP、Go、NodeJS、.NET、C++。

以云服务器查看实例列表(DescribeInstances)请求为例，当用户调用这一接口时，其请求参数可能如下:

参数名称	中文	参数值
Action	方法名	DescribeInstances
SecretId	密钥 ID	`AKIDz8krbsJ5**********mLPx3EXAMPLE`
Timestamp	当前时间戳	1465185768
Nonce	随机正整数	11886
Region	实例所在区域	ap-guangzhou
InstanceIds.0	待查询的实例 ID	ins-09dx96dg
Offset	偏移量	0
Limit	最大允许输出	20

Version	接口版本号	2017-03-12

1. 对参数排序

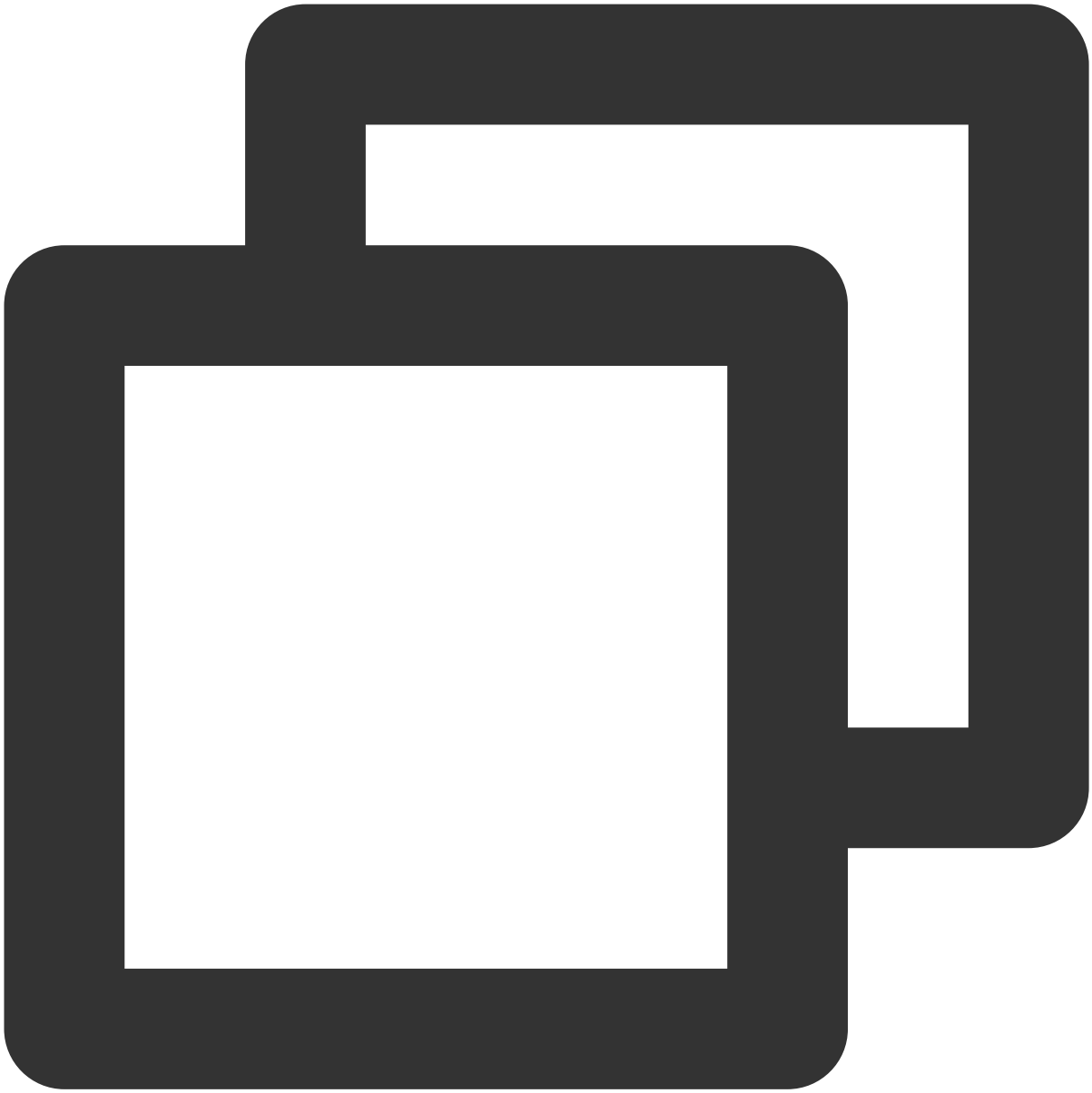
首先对所有请求参数按参数名的字典序（ASCII 码）升序排序。

注意：

1. 只按参数名进行排序，参数值保持对应即可，不参与比大小。

2. 按 ASCII 码比大小，如 InstanceIds.2 要排在 InstanceIds.12 后面，不是按字母表，也不是按数值。用户可以借助编程语言中的相关排序函数来实现这一功能，如 PHP 中的 ksort 函数。

上述示例参数的排序结果如下：



```
{
    'Action' : 'DescribeInstances',
    'InstanceIds.0' : 'ins-09dx96dg',
    'Limit' : 20,
    'Nonce' : 11886,
    'Offset' : 0,
    'Region' : 'ap-guangzhou',
    'SecretId' : 'AKIDz8krbsJ5**********mLPx3EXAMPLE',
    'Timestamp' : 1465185768,
    'Version': '2017-03-12',
}
```

使用程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

2. 拼接规范请求串

此步骤生成请求字符串。 将把上一步排序好的请求参数格式化成“参数名称=参数值”的形式，如对 Action 参数，其参数名称为 "Action" ，参数值为 "DescribeInstances" ，因此格式化后就为 Action=DescribeInstances。

注意：

“参数值”为原始值而非 url 编码后的值。

然后将格式化后的各个参数用"&"拼接在一起，最终生成的请求字符串为:

```
Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&R
```

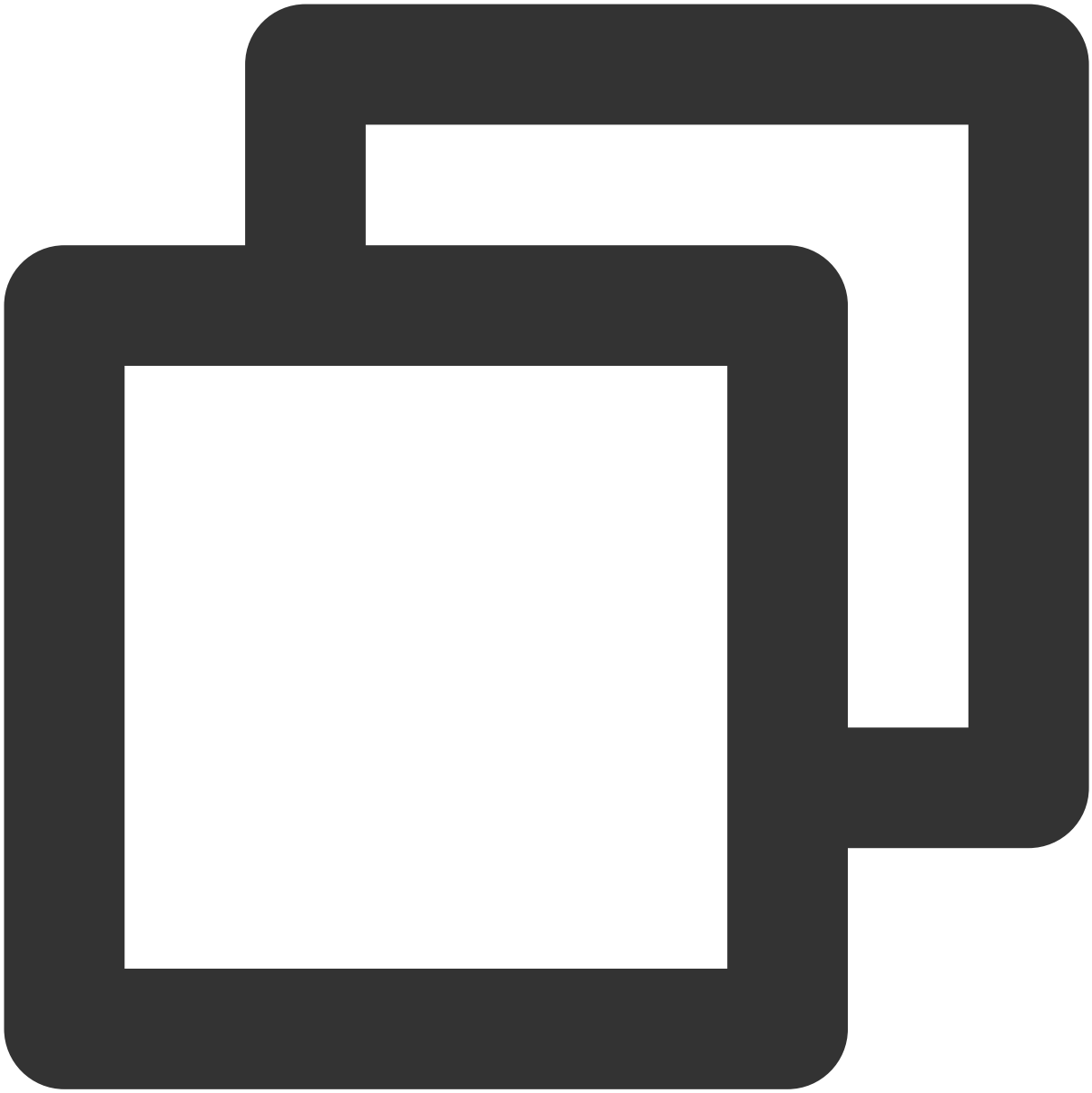
3. 拼接待签名字符串

此步骤生成签名原文字符串。 签名原文字符串由以下几个参数构成：

1. 请求方法: 支持 POST 和 GET 方式，这里使用 GET 请求，注意方法为全大写。
2. 请求主机:查看实例列表(DescribeInstances)的请求域名为：cvm.tencentcloudapi.com。实际的请求域名根据接口所属模块的不同而不同，详见各接口说明。
3. 请求路径: 当前版本云API的请求路径固定为 / 。
4. 请求字符串: 即上一步生成的请求字符串。

签名原文串的拼接规则为： `请求方法 + 请求主机 +请求路径 + ? + 请求字符串` 。

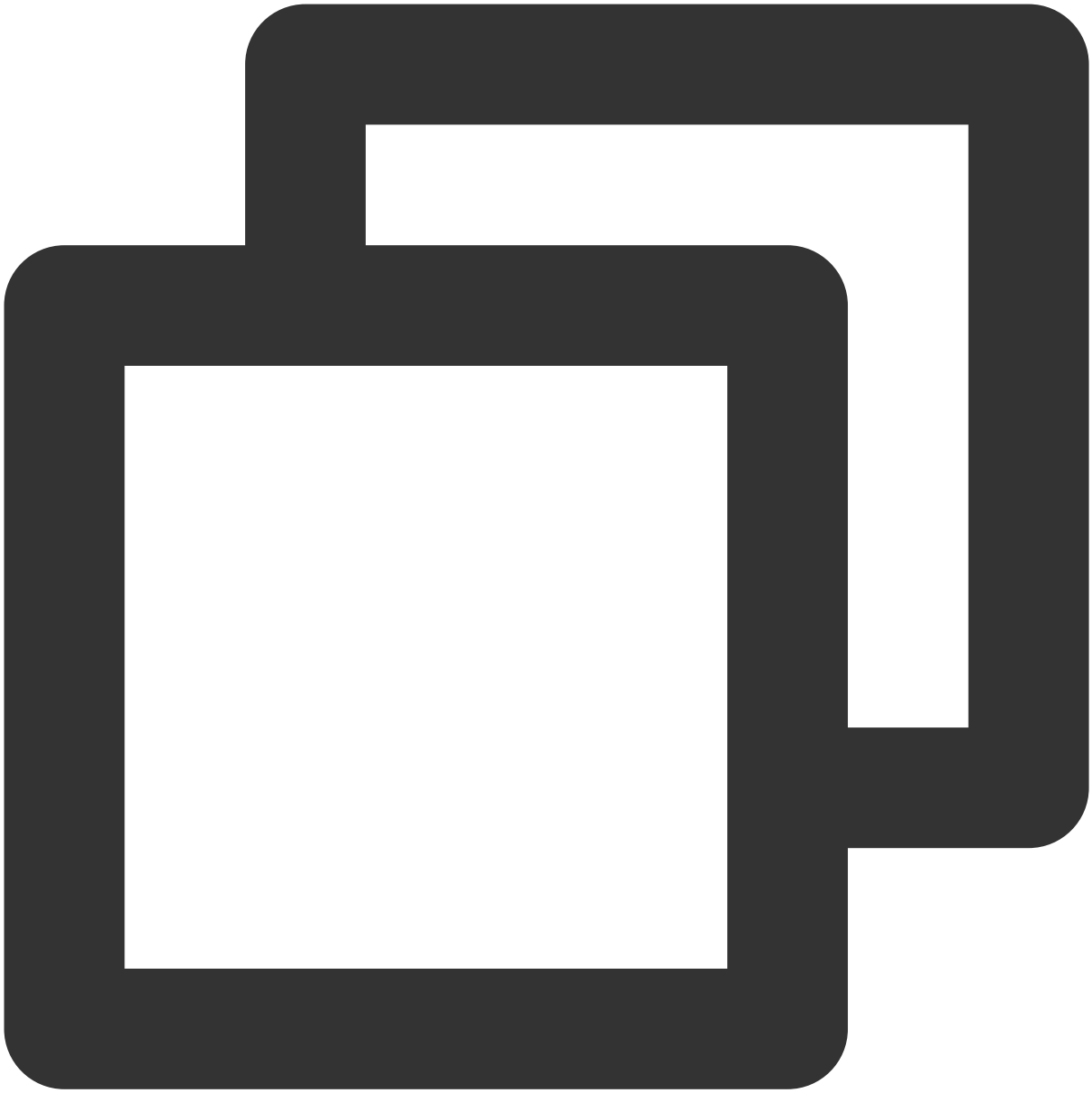
示例的拼接结果为：



```
GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Lim
```

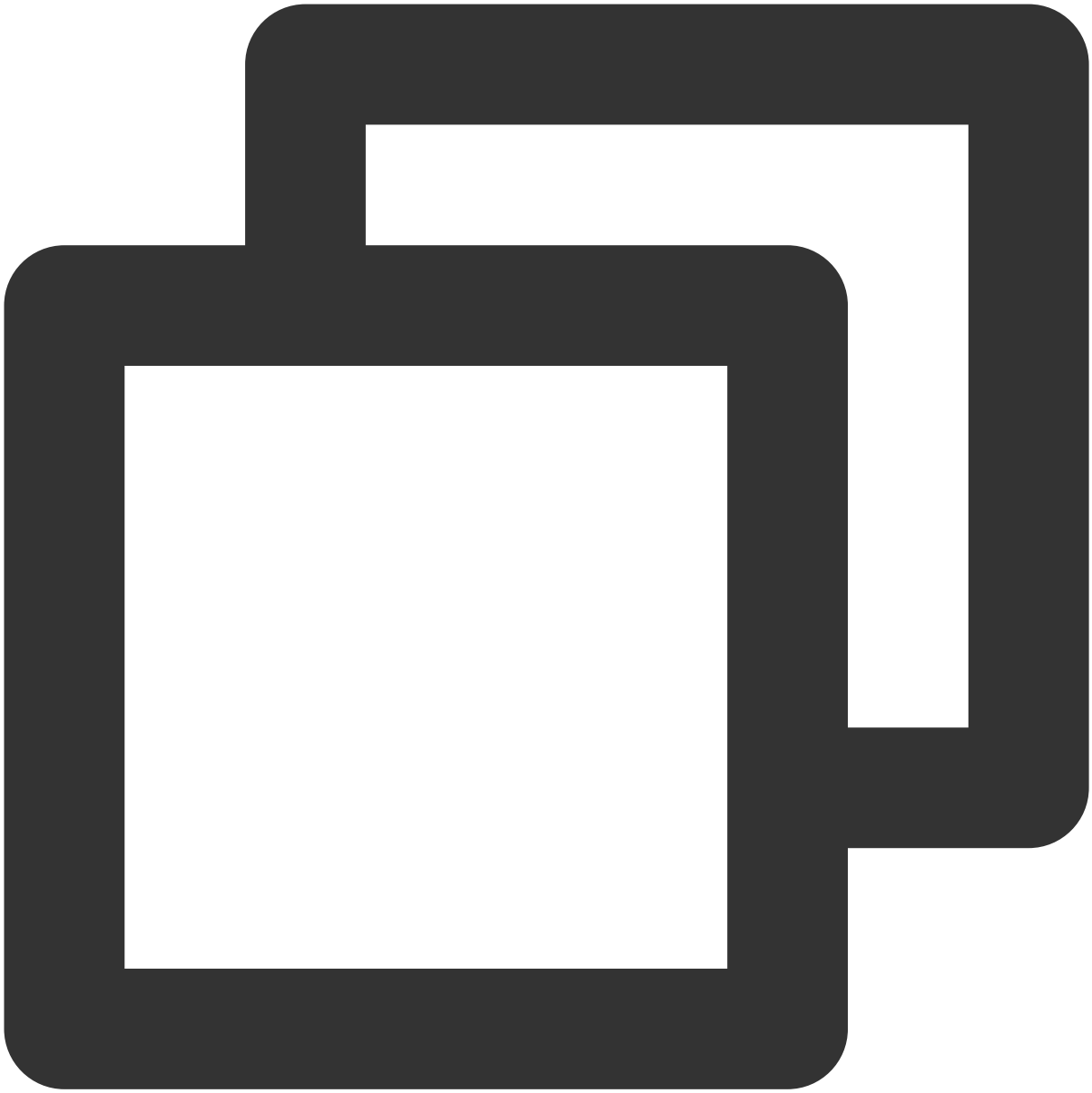
4. 计算签名（伪代码）

此步骤生成签名串。 首先使用 HMAC-SHA1 算法对上一步中获得的**签名原文字符串**进行签名，然后将生成的签名串使用 Base64 进行编码，即可获得最终的签名串。



```
secret_key = "Gu5t9xGAR********QYCN3EXAMPLE"
s = "GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96d
hmac_str = hmac.new(secret_key.encode("utf8"), s.encode("utf8"),hashlib.sha1).diges
# 最终签名串
Signature = base64.b64encode(hmac_str)
```

5. 获取调用信息并发送请求



```
data["Signature"] = base64.b64encode(hmac_str)
print(data["Signature"])  # 最终签名串
# 此处会实际调用，成功后可能产生计费
resp = requests.get("https://" + endpoint, params=data)
print(resp.url)
```

字段名称	解释
endpoint	服务地址， 例如： `cvm.tencentcloudapi.com`
data	API 3.0 签名 V1 所举示例接口参数，**注意** 这里需要将计算的签名已键值对的形式加入 data 中

注意：

由于示例中的密钥是虚构的，时间戳也不是系统当前时间，因此如果将此 url 在浏览器中打开或者用 curl 等命令调用时会返回鉴权错误：签名过期。为了得到一个可以正常返回的 url ，需要修改示例中的 SecretId 和 SecretKey 为真实的密钥，并使用系统当前时间戳作为 Timestamp 。

为了更清楚的解释签名过程，下面以 Python 语言为例，将上述的签名过程具体实现。请求的域名、调用的接口和参数的取值都以上述签名过程为准，代码只为解释签名过程，并不具备通用性，实际开发请尽量使用 SDK 。

6. 签名串编码

生成的签名串并不能直接作为请求参数，需要对其进行 URL 编码。

如上一步生成的签名串为 `Eli**********cGeI=` ，最终得到的签名串请求参数（Signature）为： `EliP**********eI%3D` ，它将用于生成最终的请求 URL。

注意：

如果用户的请求方法是 GET，或者请求方法为 POST 同时 Content-Type 为 application/x-www-form-urlencoded，则发送请求时所有请求参数的值均需要做 URL 编码，参数键和=符号不需要编码。非 ASCII 字符在 URL 编码前需要先以 UTF-8 进行编码。

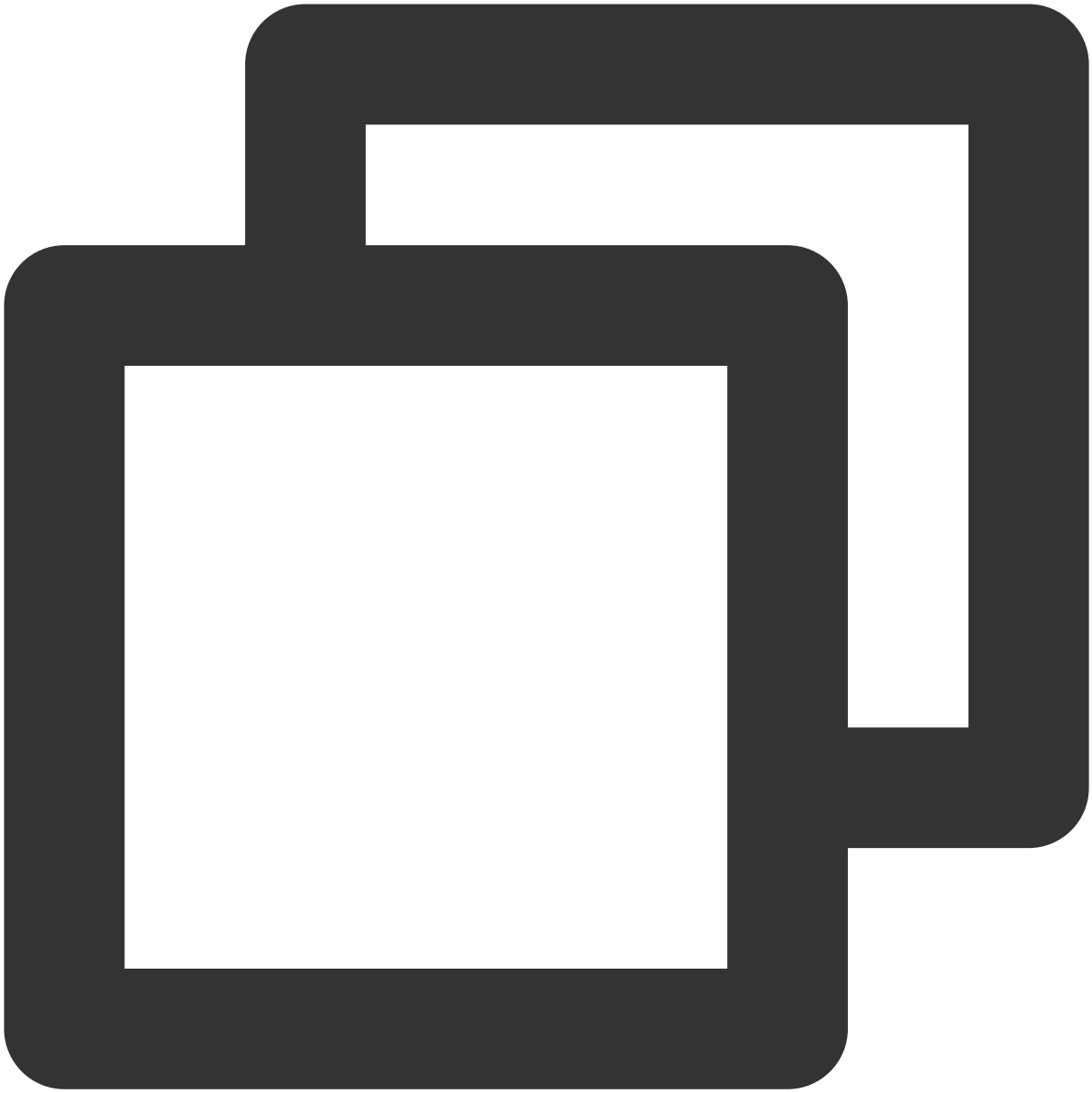
有些编程语言的网络库会自动为所有参数进行 urlencode，在这种情况下，就不需要对签名串进行 URL 编码了，否则两次 URL 编码会导致签名失败。

其他参数值也需要进行编码，编码采用 RFC 3986。使用 %XY 对特殊字符例如汉字进行百分比编码，其中“X”和“Y”为十六进制字符（0-9 和大写字母 A-F），使用小写将引发错误。

7. API 3.0 签名 V1示例

注意：

如果是在 Python 2环境中运行，需要先安装 requests 依赖包： `pip install requests` 。



```
# -*- coding: utf8 -*-
import base64
import hashlib
import hmac
import time

import requests

secret_id = "AKIDz8k*********LPx3EXAMPLE"
secret_key = "Gu5t9xGA*********YCN3EXAMPLE"
```

```
def get_string_to_sign(method, endpoint, params):
    s = method + endpoint + "/?"
    query_str = "&".join("%s=%s" % (k, params[k]) for k in sorted(params))
    return s + query_str

def sign_str(key, s, method):
    hmac_str = hmac.new(key.encode("utf8"), s.encode("utf8"), method).digest()
    return base64.b64encode(hmac_str)

if __name__ == '__main__':
    endpoint = "cvm.tencentcloudapi.com"
    data = {
        'Action' : 'DescribeInstances',
        'InstanceIds.0' : 'ins-09dx96dg',
        'Limit' : 20,
        'Nonce' : 11886,
        'Offset' : 0,
        'Region' : 'ap-guangzhou',
        'SecretId' : secret_id,
        'Timestamp' : 1465185768, # int(time.time())
        'Version': '2017-03-12'
    }
    s = get_string_to_sign("GET", endpoint, data)
    data["Signature"] = sign_str(secret_key, s, hashlib.sha1)
    print(data["Signature"])
    # 此处会实际调用，成功后可能产生计费
    # resp = requests.get("https://" + endpoint, params=data)
    # print(resp.url)
```

API 2.0 签名

此签名现已不在维护，建议使用性能更优的 **API 3.0 签名**，如需使用，请直接在 API Explorer 的**签名串生成>选择 API 2.0 签名版本**进行操作。

签名失败

存在以下签名失败的错误码，请根据实际情况处理。

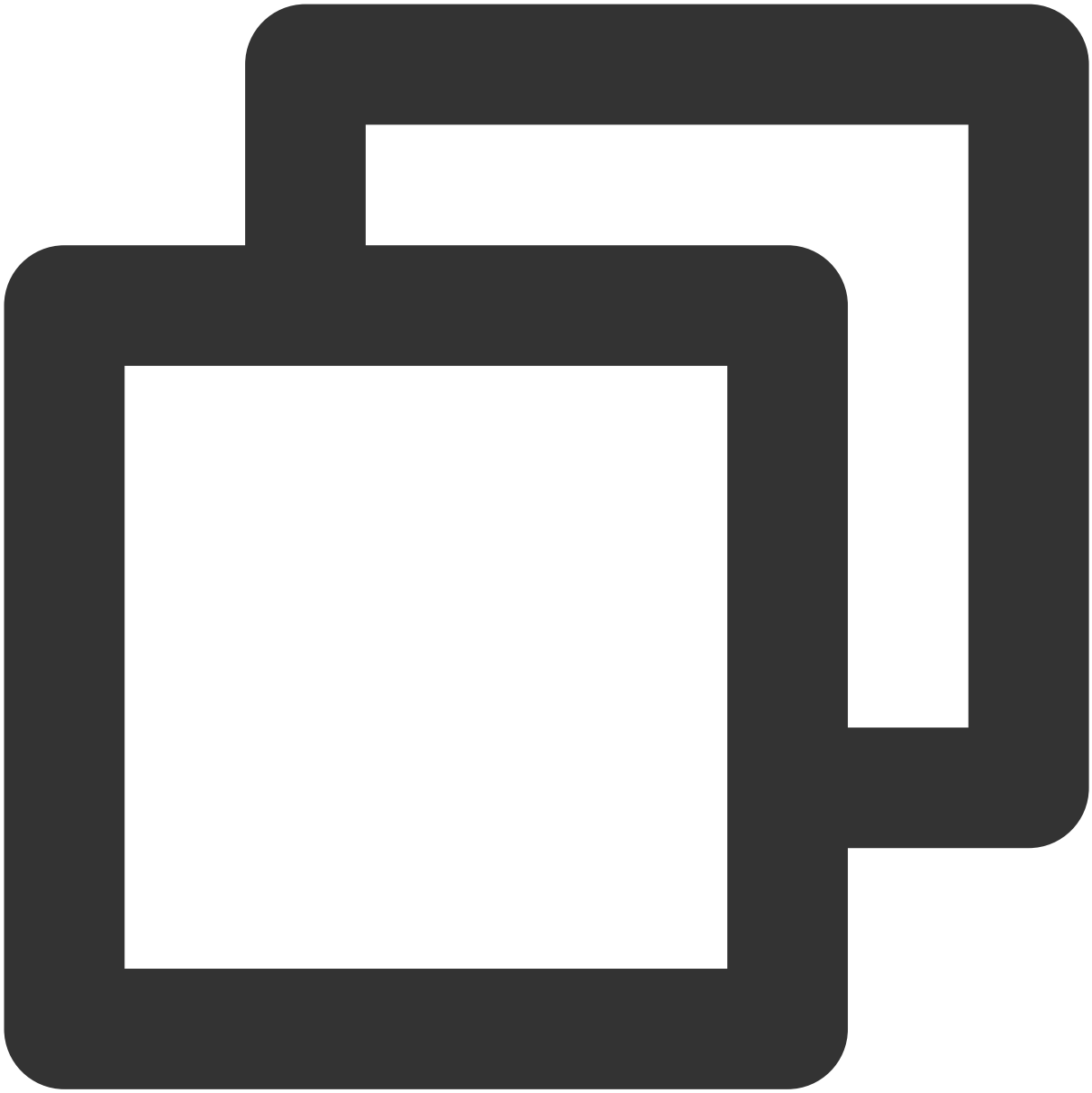
错误码	错误描述
AuthFailure.SignatureExpire	签名过期。Timestamp 与服务器接收到请求的时间相差不得超过五分钟。
AuthFailure.SecretIdNotFound	密钥不存在。请到控制台查看密钥是否被禁用，是否少复制了字符或者多了

	字符。
AuthFailure.SignatureFailure	签名错误。可能是签名计算错误，或者签名与实际发送的内容不相符合，也有可能是密钥 SecretKey 错误导致的。
AuthFailure.TokenFailure	临时证书 Token 错误。
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。

返回结果

正确返回结果

以云服务器的接口查看实例状态列表 (DescribeInstancesStatus) 2017-03-12 版本为例，若调用成功，其可能的返回如下为：



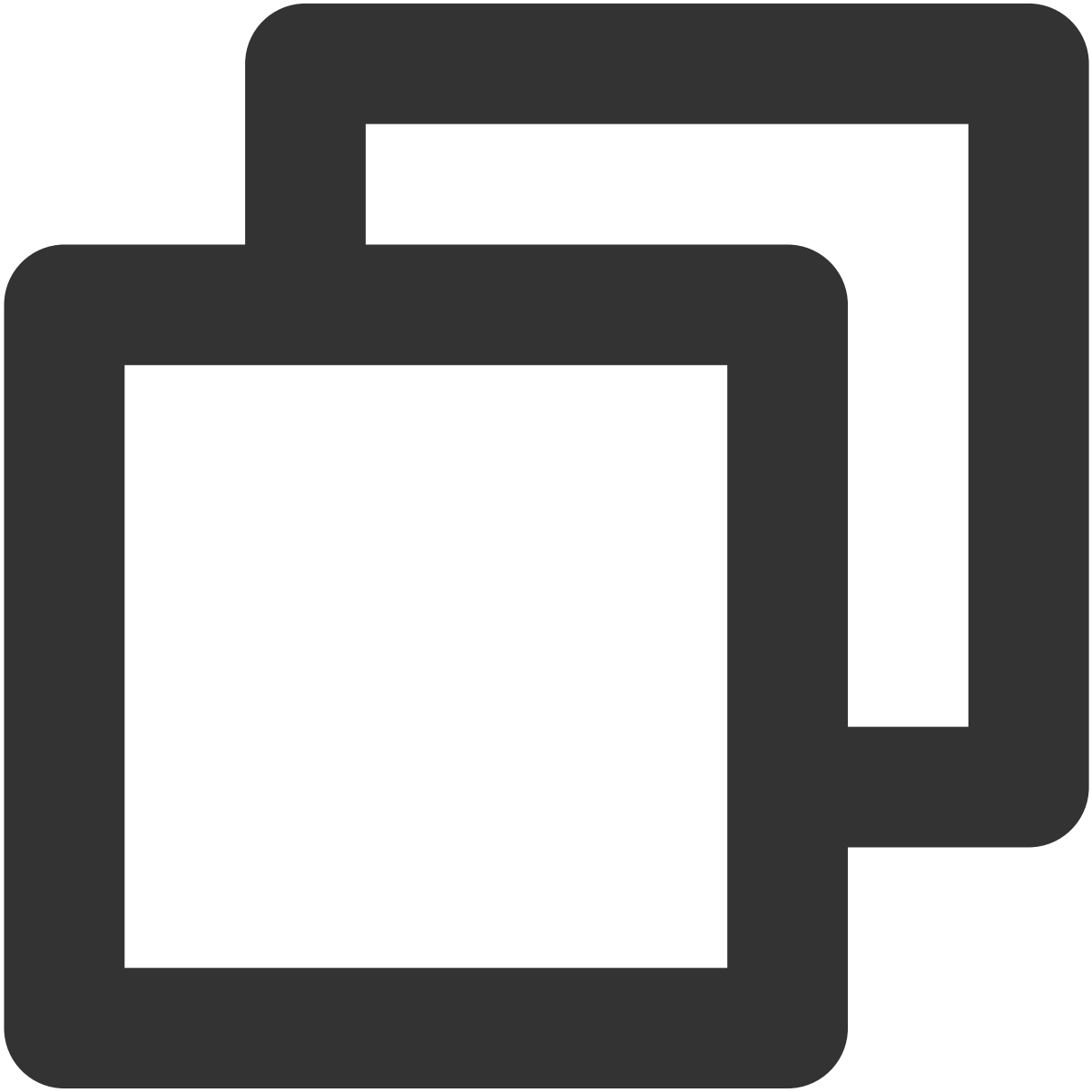
```
{
    "Response": {``
        "TotalCount": 0,
        "InstanceStatusSet": [],
        "RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
    }
}
```

`Response` 及其内部的 `RequestId` 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 `TotalCount` 和 `InstanceStatusSet` 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 `TotalCount` 在此情况下的返回值为 0， `InstanceStatusSet` 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：



```
{
    "Response": {
        "Error": {
```

```
            "Code": "AuthFailure.SignatureFailure",
            "Message": "The provided credentials could not be validated. Please che
        },
        "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
    }
}
```

`Error` 的出现代表着该请求调用失败。 `Error` 字段连同其内部的 `Code` 和 `Message` 字段在调用失败时是必定返回的。

`Code` 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。

`Message` 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。

`RequestId` 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。 Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。

LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	HTTPS 请求方法错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

版权所有：腾讯云计算（北京）有限责任公司