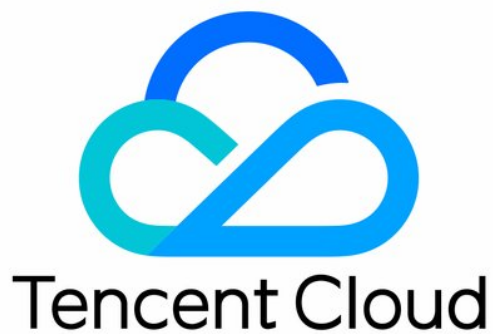


API Gateway

Verification and Security

Product Documentation



Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Verification and Security

Overview

Application-Enabled Authentication Method

Authentication-Free Mode

OAuth2.0

CAM Policy

Verification and Security

Overview

Last updated : 2023-12-22 09:53:53

Various API verification methods and protection policies are provided to protect your APIs, helping avoid data and asset losses caused by malicious access, unauthorized access, application vulnerabilities, and attacks.

Currently, API Gateway supports [application-enabled](#), [OAuth2.0](#), and [key pair](#) authentication methods. The key pair method is a historical feature, and the application-enabled method is recommended.

Application-Enabled Authentication Method

Last updated : 2023-12-22 09:54:07

If a published API uses the application-enabled authentication method (`ApiAppKey` and `ApiAppSecret`), when a client calls the API, it needs to use the signature key to perform signature calculation on the request content and transfer the signature to the server for signature verification. This document describes how to implement the signature calculation process on the client.

Note:

For application-enabled authentication signature demos for common programming languages, see [Development Guide - Generating Application Authentication Signature](#).

Overview

API Gateway provides a frontend signature calculation and verification feature, which can:

Verify the validity of a client request and confirm that the request carries the signature generated by the authorized `ApiAppKey` .

Prevent the request data from being tampered with during network transfer.

The owner of an API can generate an application on the application management page in the API Gateway console. Each application carries a signature key pair (`ApiAppKey` and `ApiAppSecret`). After the API owner authorizes the API to the specified application (which can be issued by the API owner or owned by an API caller), the API caller can use the application's signature key to call the API.

When the client calls the API, it needs to use the authorized signature key to perform encrypted signature calculation on the critical request content, put the `ApiAppKey` and the encrypted signature string in the header of the request, and transfer it to API Gateway. API Gateway will read the header information of the `ApiAppKey` in the request, query the value of the `ApiAppSecret` corresponding to the value of the `ApiAppKey` , use the `ApiAppSecret` to perform signature calculation on the critical data in the received request, and compare the generated signature with the signature sent by the client to verify the correctness of the signature. Only if the request passes the signature verification will it be sent to the backend service; otherwise, API Gateway will deem the request invalid and directly return an error.

Signature Generation and Verification Process

Prerequisites

The security authentication type of the called API is "application authentication".

The API caller needs to get the authorization granted to the application by the API before calling the API.

Signature generation on client

1. Extract the critical data from the original request and get a string for signing.
2. Use the encryption algorithm plus `ApiAppSecret` to encrypt the critical data signature string to get a signature.
3. Add all headers related to the signature to the original HTTP request to get the final HTTP request.

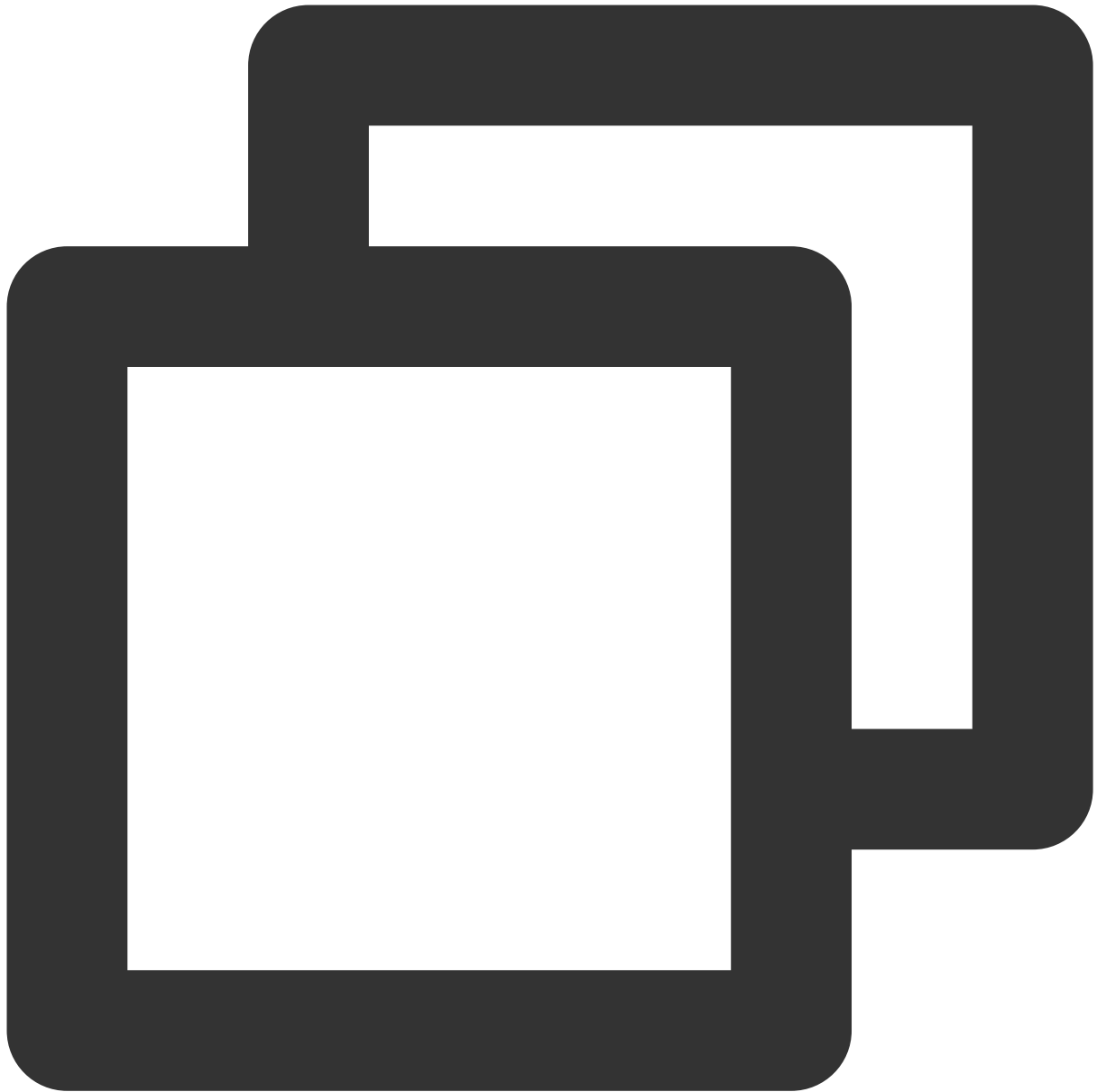
Signature verification on server

1. Extract the critical data from the received request and get a string for signing.
2. Read the `ApiAppKey` from the received request and query the corresponding `ApiAppSecret` through the `ApiAppKey`.
3. Use the encryption algorithm plus `ApiAppSecret` to encrypt the critical data signature string to get a signature.
4. Read the client signature from the received request and check whether the server signature and the client signature are the same.

Signature Generation and Transfer

Signature string extraction

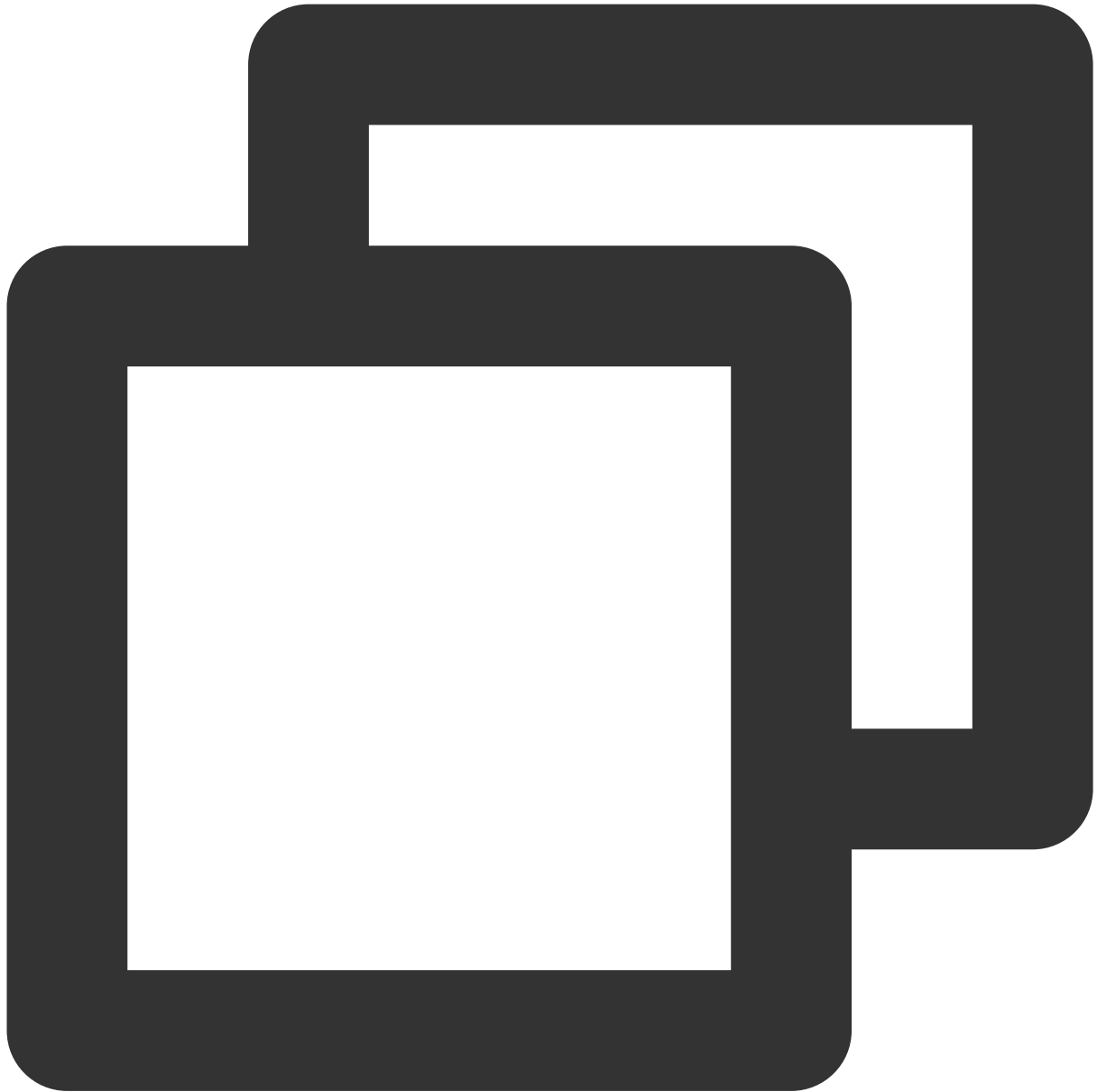
The client needs to extract the critical data from the HTTP request and splice it into a signature string in the following format:



```
Headers
HTTPMethod
Accept
Content-Type
Content-MD5
PathAndParameters
```

The above 6 fields constitute the entire signature string. They need to be separated with `\\n`. `Headers` must contain `X-Date`. There is no need to add `\\n` after `PathAndParameters`. Even if other fields are empty, `\\n` should still be retained. The signature is case sensitive. The extraction rules for each field are as follows:

Headers: you can select specified headers to participate in the signature calculation. The keys of the selected headers are sorted in lexicographical order and then spliced as follows:



```
HeaderKey1 + ": " + HeaderValue1 + "\\n"\\n"+
HeaderKey2 + ": " + HeaderValue2 + "\\n"\\n"+
...
HeaderKeyN + ": " + HeaderValueN + "\\n"
```

The headers in `Authorization` are the ones involved in signature calculation. We recommend you convert them to the lowercase and separate them by ASCII spaces. For example, if the headers involved in the calculation are

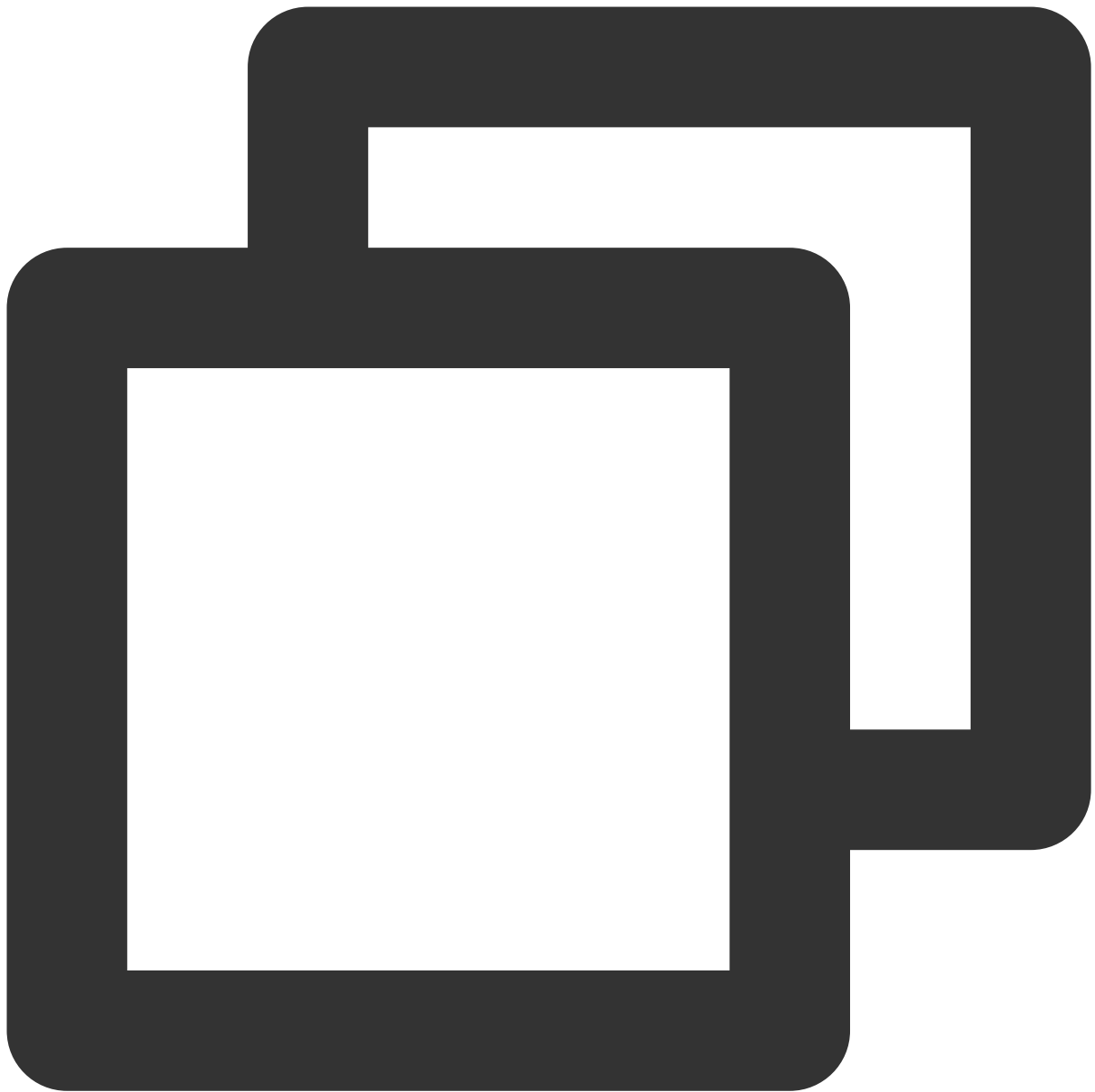
`date` and `source` , the format should be `headers="date source"` ; if only the `x-date` header participates in the calculation, the format should be `headers="x-date"` .

HTTPMethod: HTTP method. The value should be in all caps (such as POST).

Accept: value of the `Accept` header in the request, which can be empty. We recommend you explicitly set the `Accept` header. If it is empty, some HTTP clients will set the default value of `/` for it, causing signature verification to fail.

Content-Type: value of the `Content-Type` header in the request, which can be empty.

Content-MD5: value of the `Content-MD5` header in the request, which can be empty. The `Content-MD5` header is calculated only when the request has a `Body` in a `non-Form` format. The calculation method of the `Content-MD5` value in Java is as follows:



```
String content-MD5 = Base64.encodeBase64(MD5(bodyStream.getBytes("UTF-8")));
```

PathAndParameters: it contains all the parameters in `Path` , `Query` , and `Form` in the following format:

`path` does not contain release environment (release, prepub, test) information.

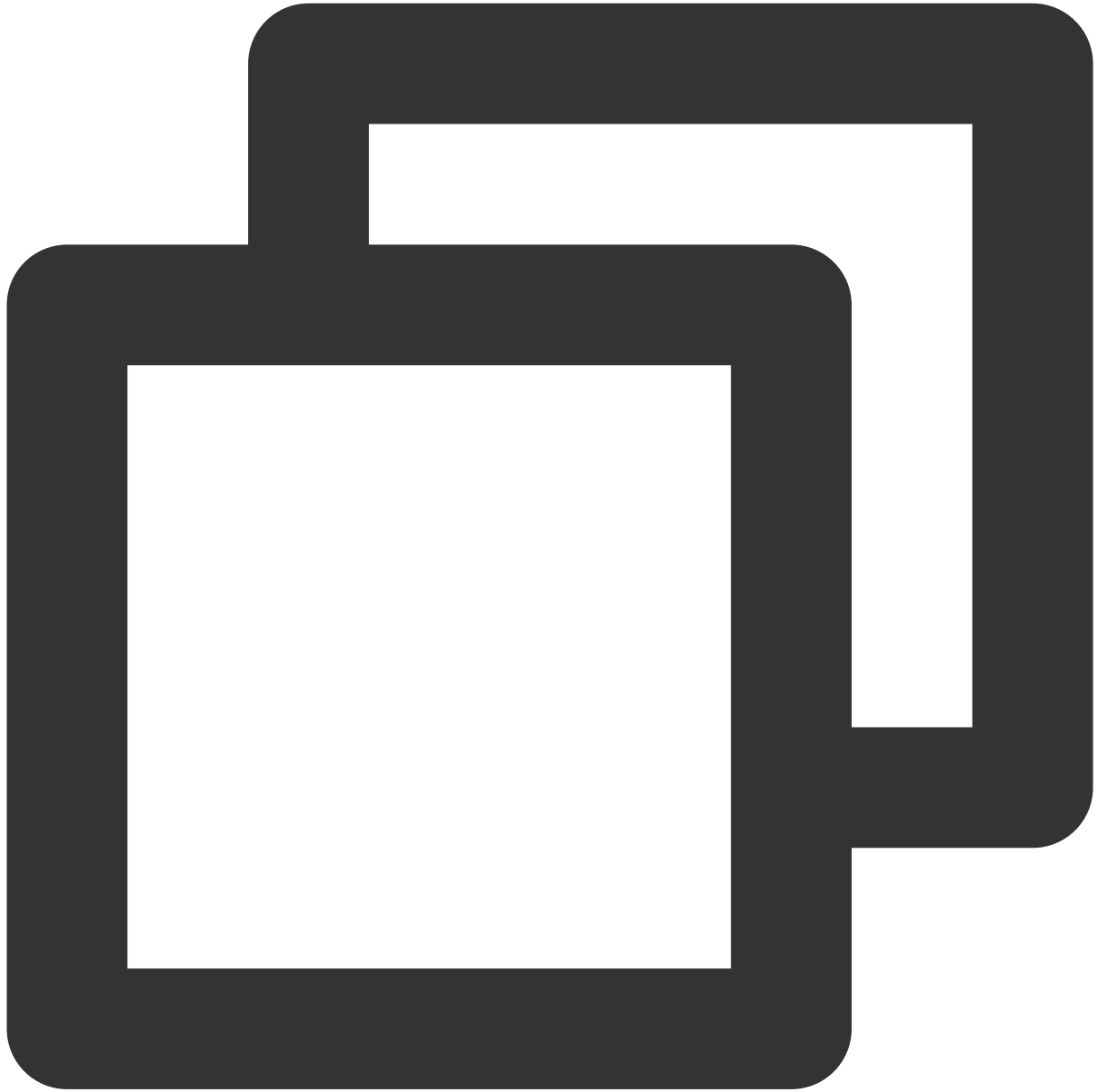
The keys of the `Query` and `Form` parameter pair are sorted in lexicographical order and then spliced in the above-mentioned method.

If `Query` and `Form` parameters are empty, use `Path` directly without adding `?` .

If the value of a parameter is empty, only the key is kept to participate in the signature calculation, and the equal sign does not need to be added in the signature.

If there are array parameters in `Query` and `Form` (i.e., parameters with the same key but different values), the values need to be sorted in lexicographical order and then spliced in the above-mentioned method.

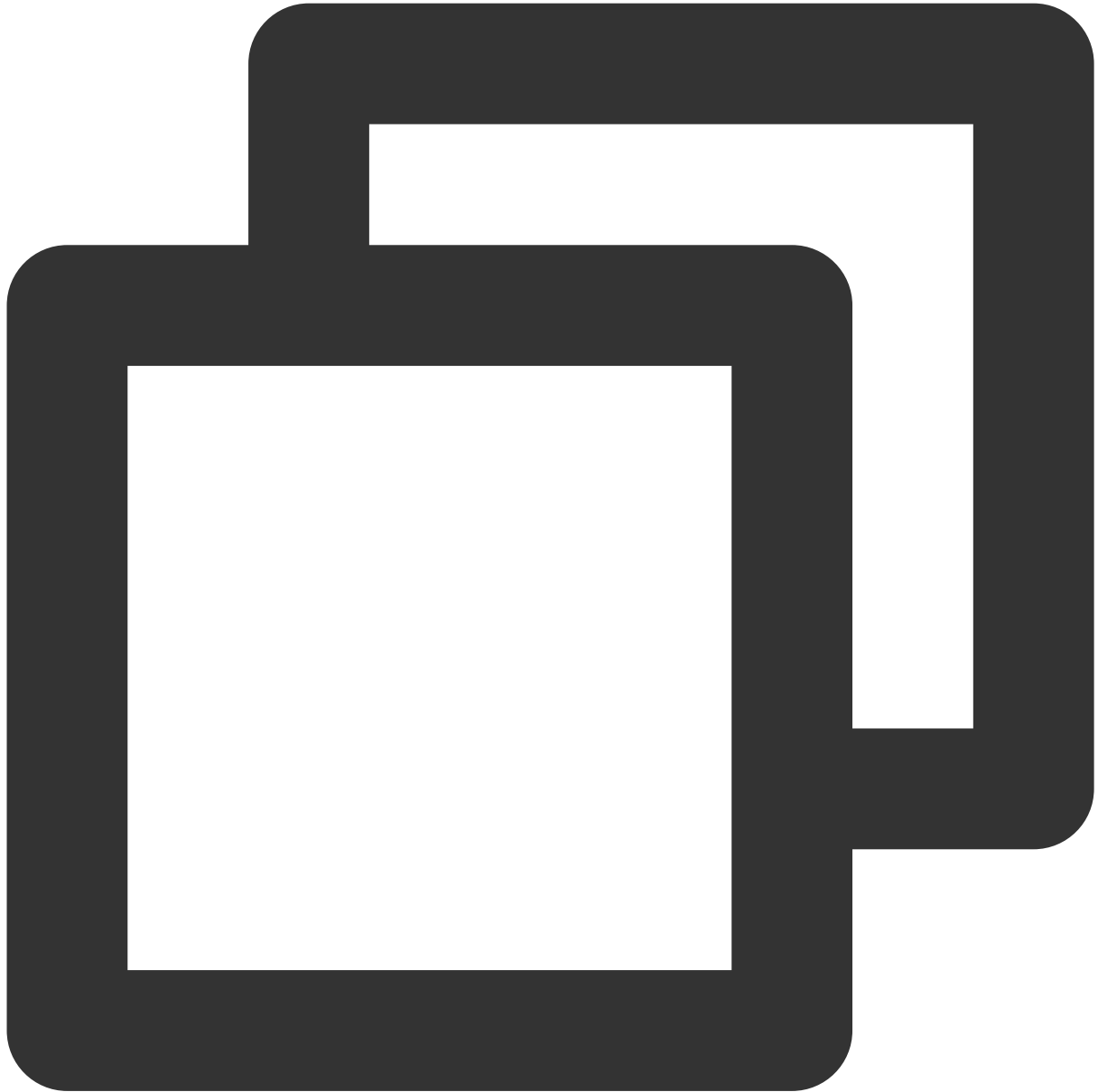
Take a general HTTP request as an example:



```
POST / HTTP/1.1
host:service-3rmwxxxx-1255968888.cq.apigw.tencentcs.com
accept:application/json
content-type:application/x-www-form-urlencoded
source:apigw test
x-date:Thu, 11 Mar 2021 08:29:58 GMT
content-length:8
```

```
p=test
```

The generated correct signature string is as follows:



```
source: apigw test
x-date: Thu, 11 Mar 2021 08:29:58 GMT
POST
application/json
application/x-www-form-urlencoded

/?p=test
```

Signature calculation

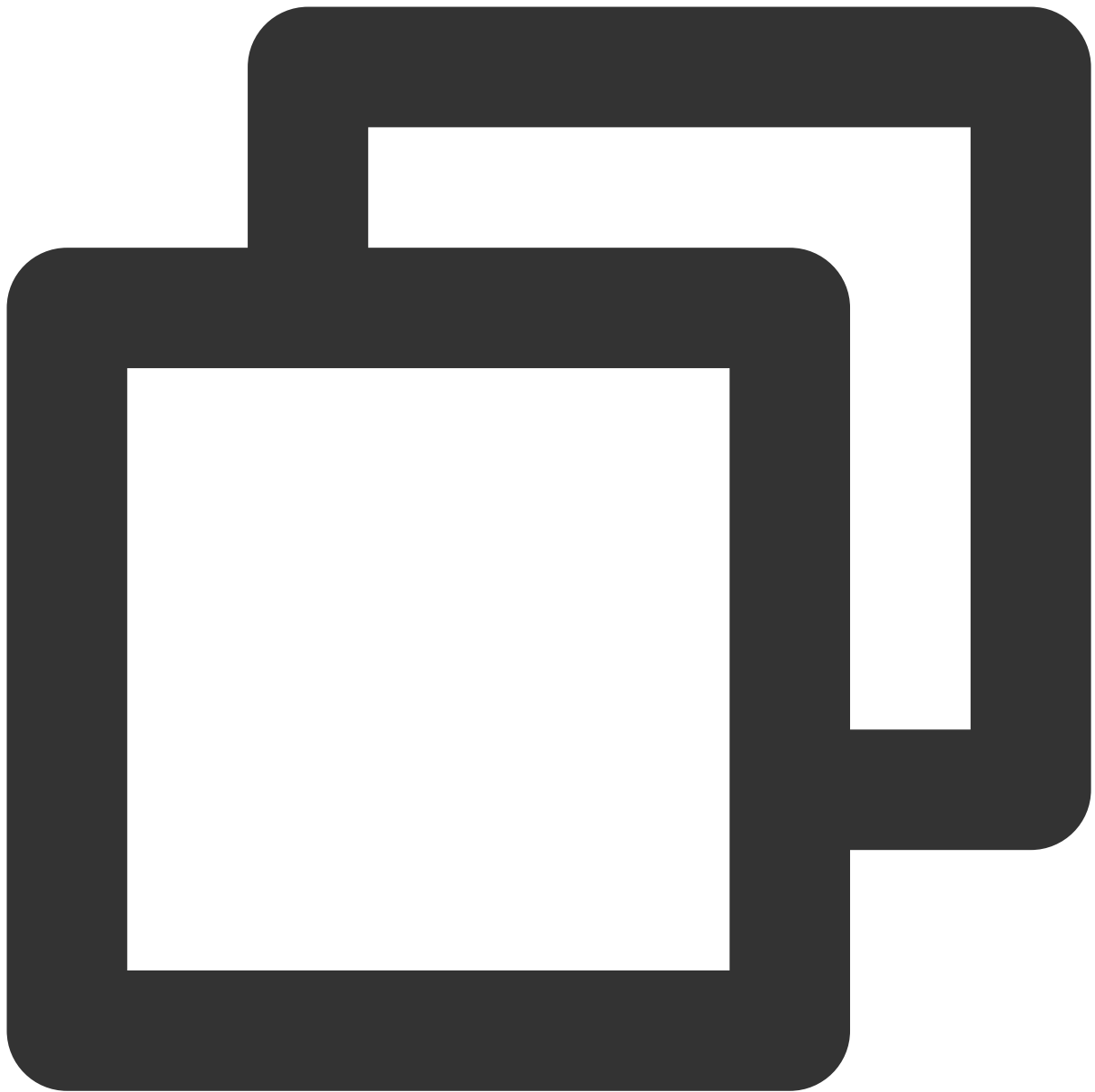
After the client extracts the critical data from the HTTP request and splices it into a signature string, it needs to encrypt and encode the signature string to form the final signature in the following steps:

1. Use UTF-8 to decode the signature string (`signing_str` signing information) to get a byte array.
2. Use the encryption algorithm to encrypt the byte array.
3. Base64-encode the encrypted byte array to form the final signature.

Signature transfer

The client needs to put the `Authorization` in the HTTP request and transfer it to API Gateway for signature verification.

The format of the `Authorization` header is as follows:



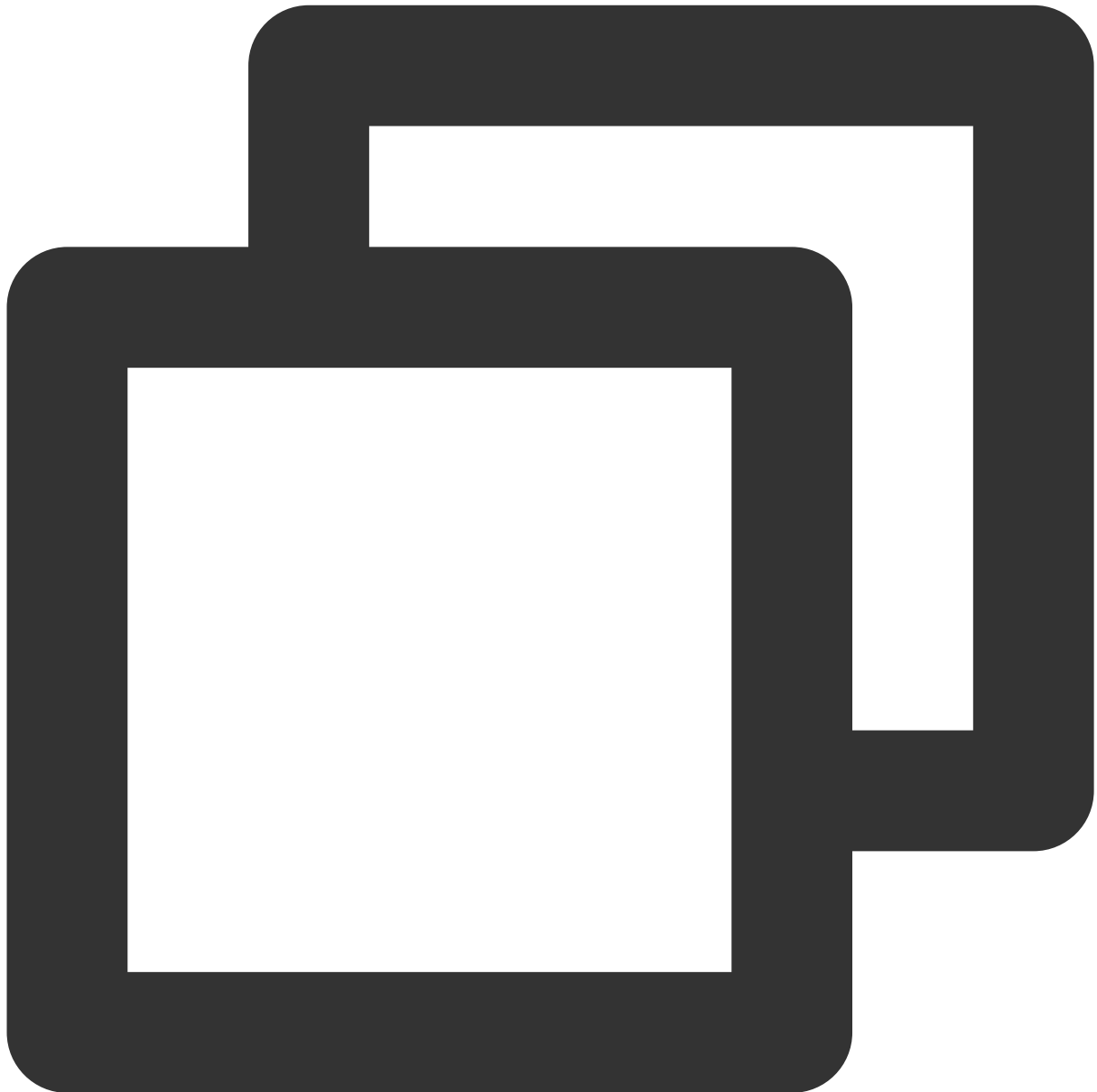
```
Authorization: hmac id="secret_id", algorithm="hmac-sha1", headers="date source", s
```

The parameters in `Authorization` are described as follows:

Parameter	Description
hmac	Fixed and used to indicate the calculation method
ID	Value of the <code>secret_id</code> in the key
algorithm	Encryption algorithm. HMAC-SHA1 and HMAC-SHA256 are supported currently

headers	Headers involved in the signature calculation
signature	Signature obtained after signature calculation is completed, with <code>signing_str</code> as its content

Below is an example of an HTTP request with a signature:



```
POST / HTTP/1.1
host:service-3rmwxxxx-1255968888.cq.apigw.tencentcs.com
accept:application/json
content-type:application/x-www-form-urlencoded
source:apigw test
```

```
x-date:Thu, 11 Mar 2021 08:29:58 GMT
Authorization: hmac id="xxxxxxx", algorithm="hmac-sha1", headers="source x-date", si
content-length:8
p=test
```

Signature Troubleshooting

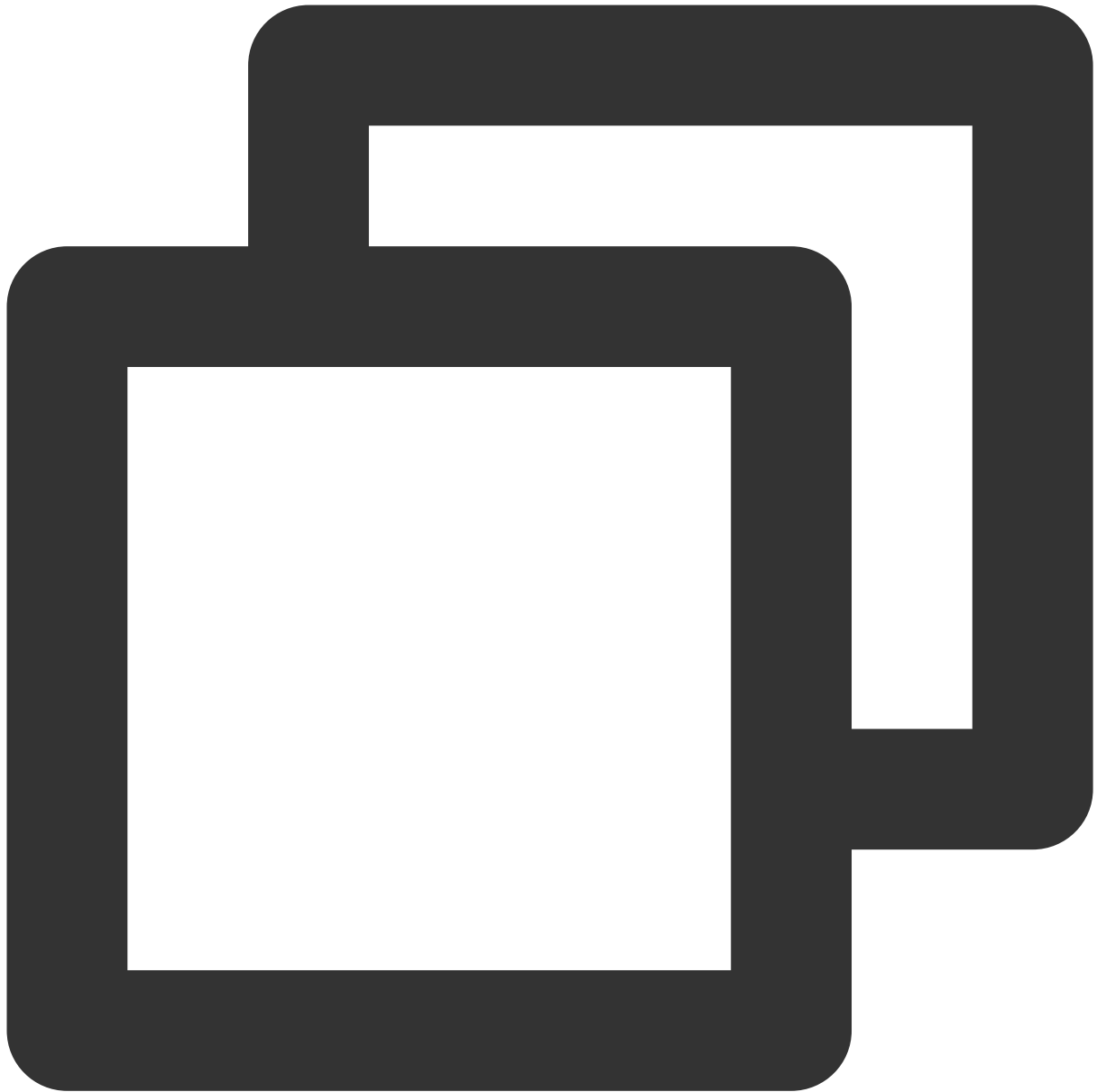
Question:

If the API Gateway signature verification fails, the server's signature string (StringToSign) will be put in the HTTP response header and returned to the client with an error code of 401.

Solution:

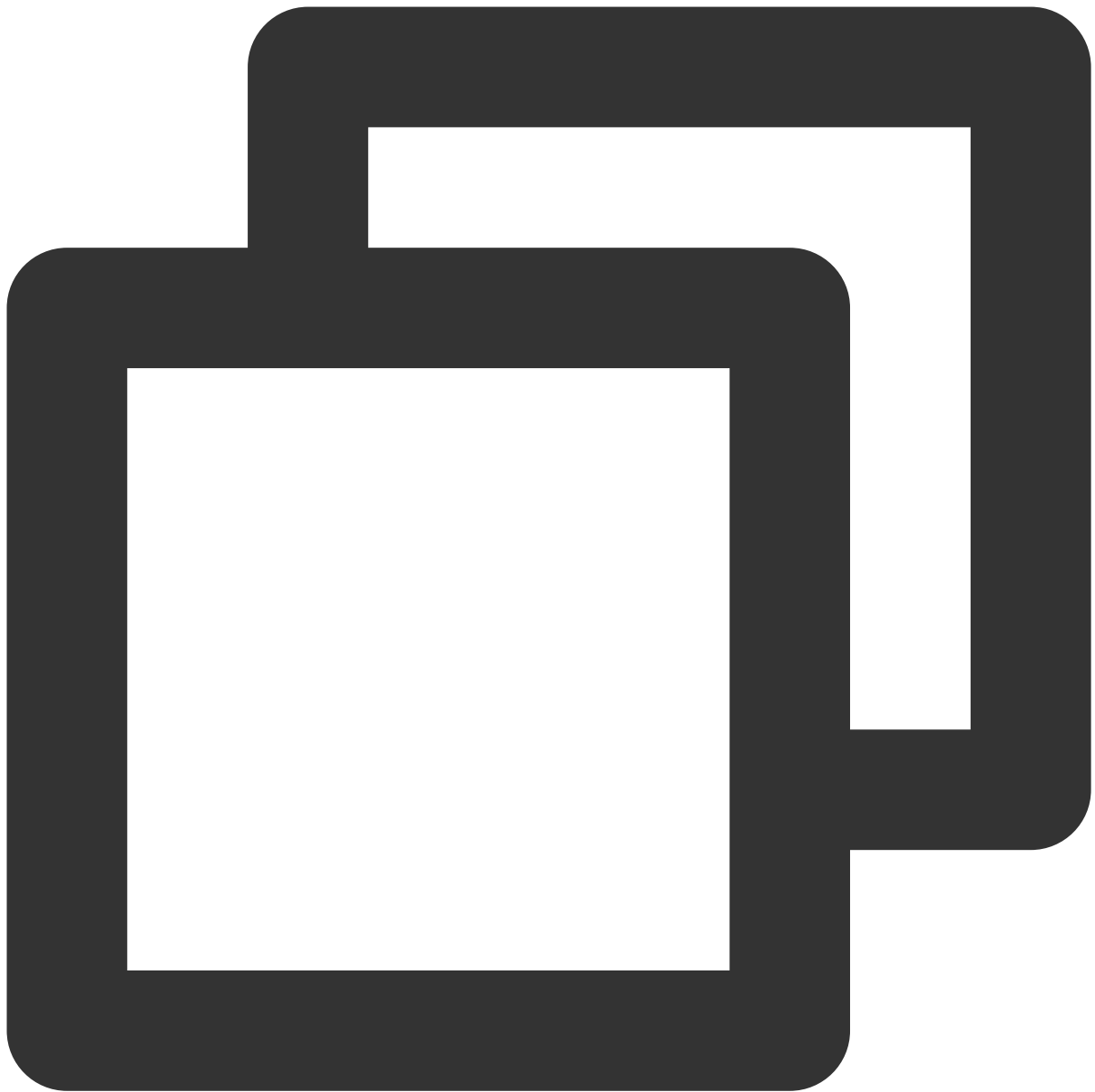
1. Check whether the locally calculated signature string (StringToSign) is the same as that returned by the server.
2. Check whether the `ApiAppSecret` used for signature calculation is correct.

Since line breaks cannot be expressed in HTTP headers, line breaks in `StringToSign` are replaced with `#`.



```
"message": "HMAC signature does not match, Server StringToSign:source: apigw test#x-
```

This means that the server signature is:



```
source: apigw test
x-date: Thu, 11 Mar 2021 08:29:58 GMT
POST
application/json
application/x-www-form-urlencoded

/?p=test
```

Authentication-Free Mode

Last updated : 2023-12-22 09:54:18

You can select "Authentication-Free" for your API when creating it.

If "Authentication-Free" is checked, authentication will succeed and the bound usage plan will take effect when API Gateway receives an anonymous request.

If signature authentication is performed with the key in the usage plan, the traffic limit in the usage plan will take effect. In case of access by an anonymous user, the maximum traffic limit imposed on each API by Tencent Cloud will take effect.

OAuth2.0

Last updated : 2023-12-22 09:54:29

Overview

This topic describes how to configure OAuth 2.0 authorization access for APIs in the API Gateway console to meet your personalized security setting needs.

OAuth 2.0 Overview

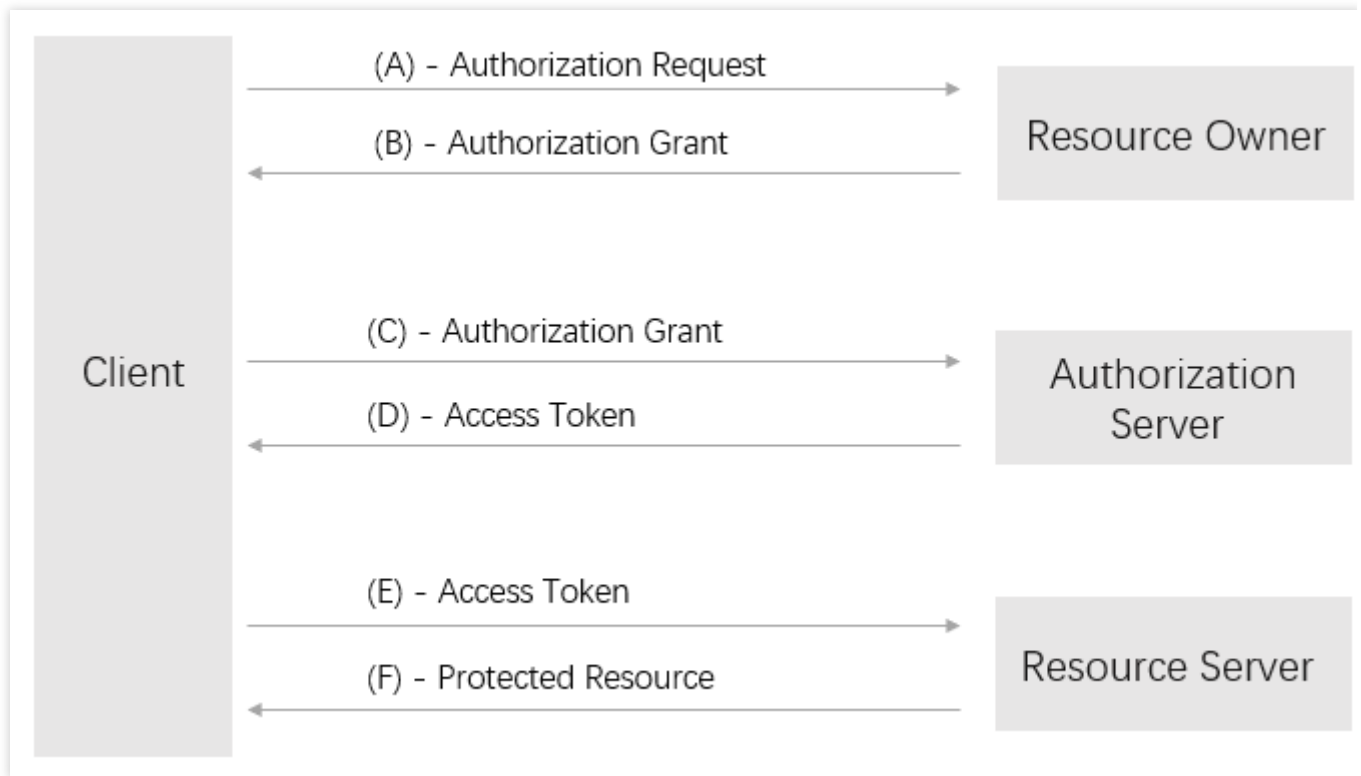
OAuth 2.0 is an open authorization standard that enables you to allow **third-party applications** to access your **specific private resources** in a service without **providing the account and password** to the applications. OAuth 2.0 is an authorization protocol rather than an authentication protocol.

OAuth 2.0 roles

OAuth 2.0 has the following 4 roles:

Role	Description
Resource owner	Owner of the resource
Resource server	Server where the resource is stored
Client	Third-party application client, which can be any third-party application that can consume the resource server
Authorization server	Intermediate layer that manages the above 3 roles

OAuth 2.0 authorization process



(A) The client initiates a request to the resource owner for authorization.

(B) The resource owner approves the authorization.

(C) The client applies to the authorization server for an authorization token after getting the resource owner's authorization.

(D) The authorization server grants the authorization token after authenticating the client.

(E) The client requests the resource server to send the user information after getting the authorization token.

(F) The resource server sends the user information to the client after verifying that the token is correct.

Prerequisites

An authorization server for distributing tokens is available. (You need to build an authorization server. The API Gateway provides the [Python3 Demo](#) and the [Golang Demo](#) for your reference.)

You have created an API Gateway service (for more information, see [Creating Services](#)).

Directions

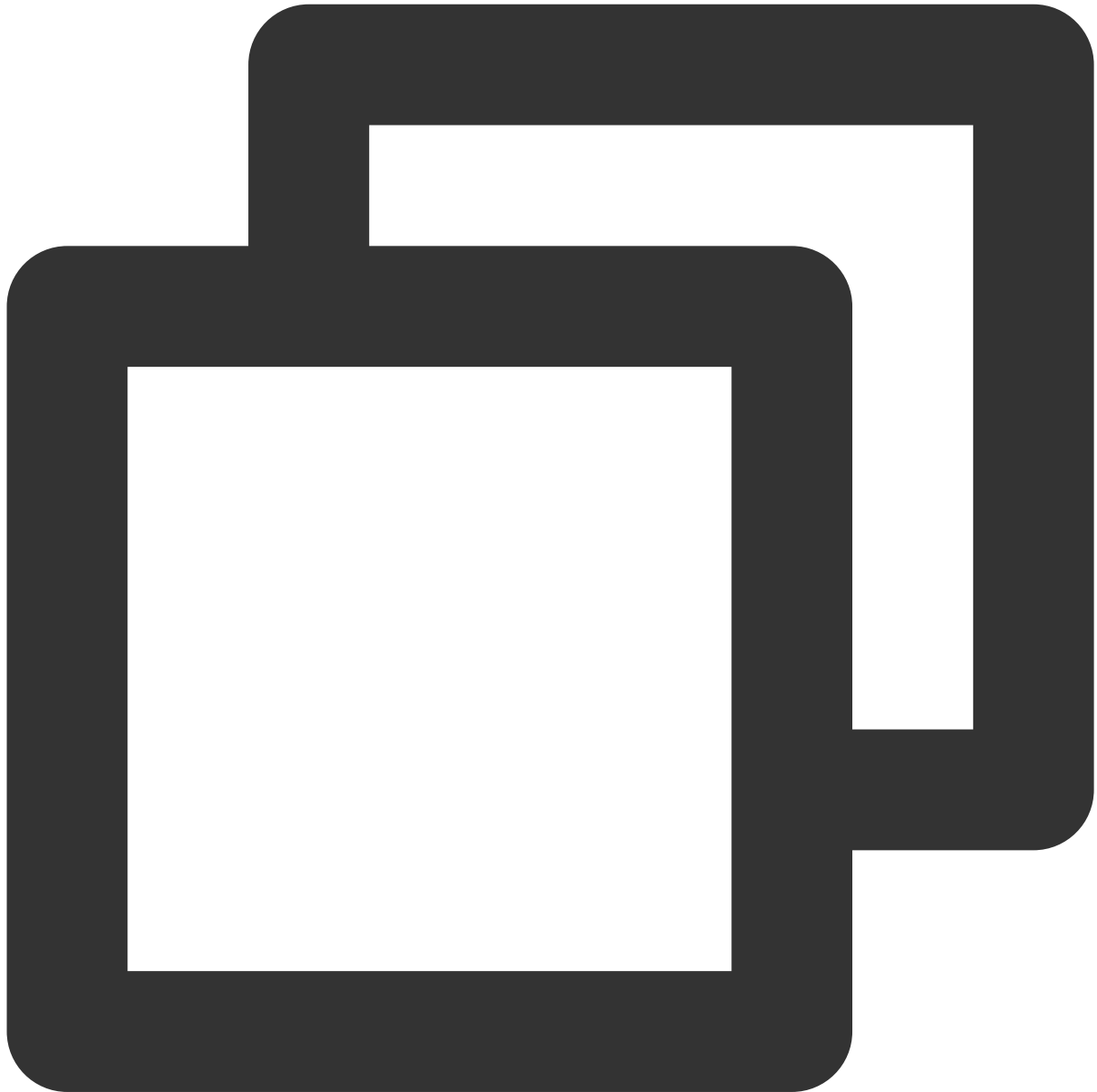
Step 1: build an authorization server (Python3 Demo is used as an example).

1. Download the [Python3 Demo](#) from the official repository of the API Gateway.
2. Generate the RSA public and private keys and run `produce_key.py` in Python 3 to generate 3 files:

public_pem: public key in PEM format.

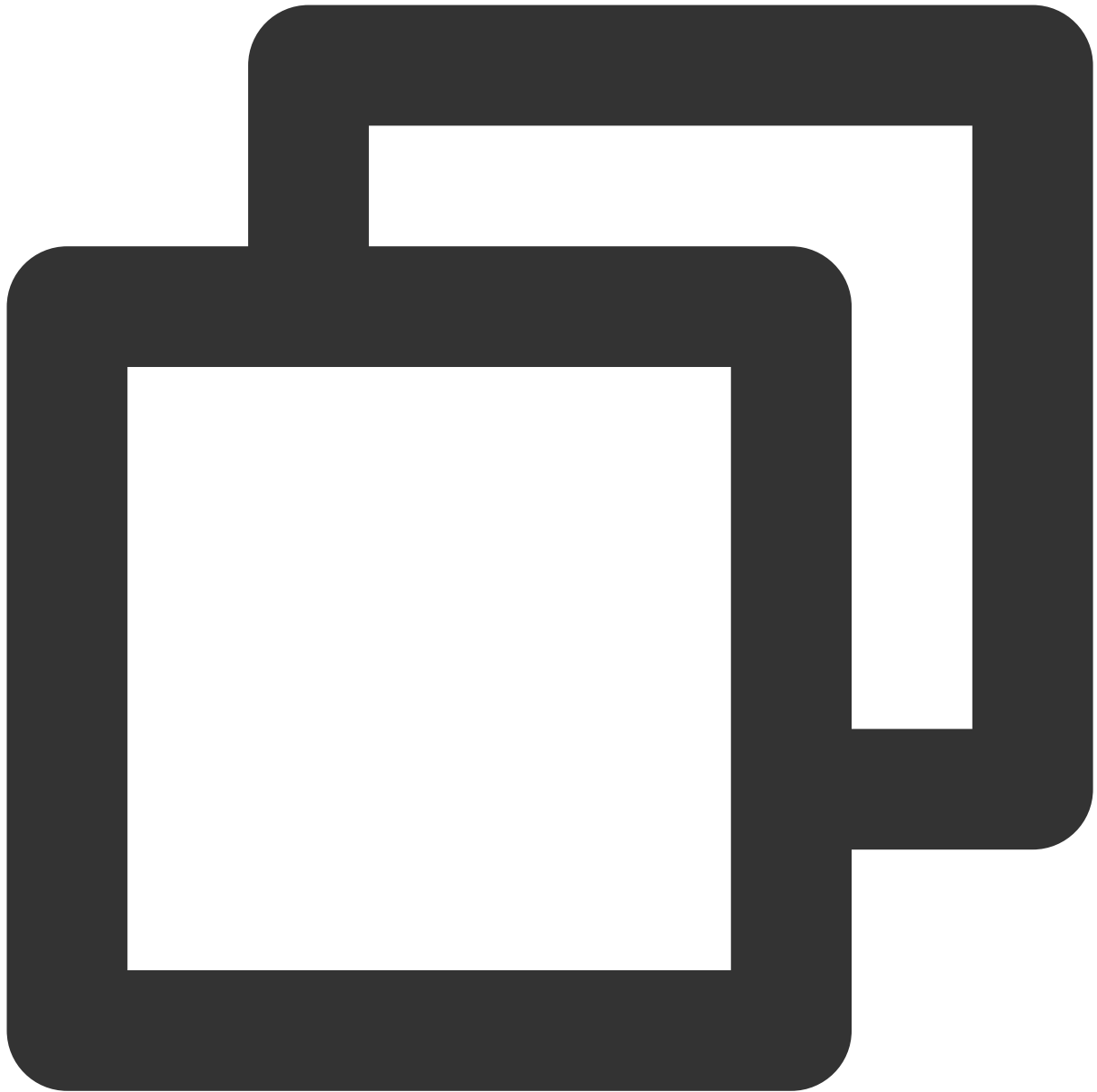
priv_pem: private key in PEM format.

public: public key in JSON format. The file content is used to configure the authorization API of API Gateway and is in the following format:



```
{"e": "AQAB", "kty": "RSA", "n": "43nSuC6lmGLogEPgFVwaaxAmPDzmZcocRB4Jed_dHc-sV7rcAcNB0i
```

3. Start the service. After installing the `bottle` library by running `pip3 install bottle`, run `server.py` in Python 3 to generate a token. Then, you can simply check whether the token is successfully generated.



```
curl localhost:8080/token
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1OTIyNzgwODksImZvbyI6ImJhcnVzImVudC
```

Step 2: configure a Tencent Cloud API Gateway authorization API.

1. In the created service, create an authorization API. When you are configuring the frontend, select **OAuth 2.0** as the authentication type and **Authorization API** as the OAuth mode.

1 Frontend Configuration

2 Backend Configuration

3 Response Result

Service

testapigateway

API Name

outhapi

Up to 60 chars

Frontend Type

http

Path

1、 It starts with "/", "=", or "^~/", and supports uppercase and lowercase letters, digits, and \$-_.+!*()/%.
2、 The request parameter of Path type must be enclosed with {}, and should be an independent component of the path (s

Request Method

GET

Authentication Type

OAuth 2.0

OAuth Mode

Authorization API

CORS is supported

☐

Remarks

Please enter remarks

Parameter Configuration

Parameter Name	Parameter Location ⓘ	Type	Default Value ⓘ
Newly added parameter configuration (0/30)			

2. When you are configuring the backend, select your own server address as the authentication server, select **Header** as the token location, and enter the content in the `public` file generated by running `produce_key.py` as the public key. After the API is created, click **Complete**.

✓ Frontend Configuration

>

2 Backend Configuration

>

3 Response Result

Backend Type

OAuth 2.0

VPC Info

Disable VPC

Backend Domain Name ⓘ

http://

8000

Backend Path

Request Method

GET

Token Location

Header

Redirect Address

Backend timeout ⓘ

15

Public Key

It starts with http or https and contains domain content, and "/" is not required at the end

1、 It starts with "/", and supports uppercase and lowercase letters, digits, and \$-.,+!*()/%.

2、 "=" and "^~" in the frontend parameters are used for exact match to the frontend path, which are not available to the backend.

3、 The request parameter of Path type must be enclosed with {}, and should be an independent component of the path (such as {id}).

Optional. You may enter the address to redirect to when a bound service API is called without authorization. But the redirect is in the same domain as the API.

Time range: 1-1,800s

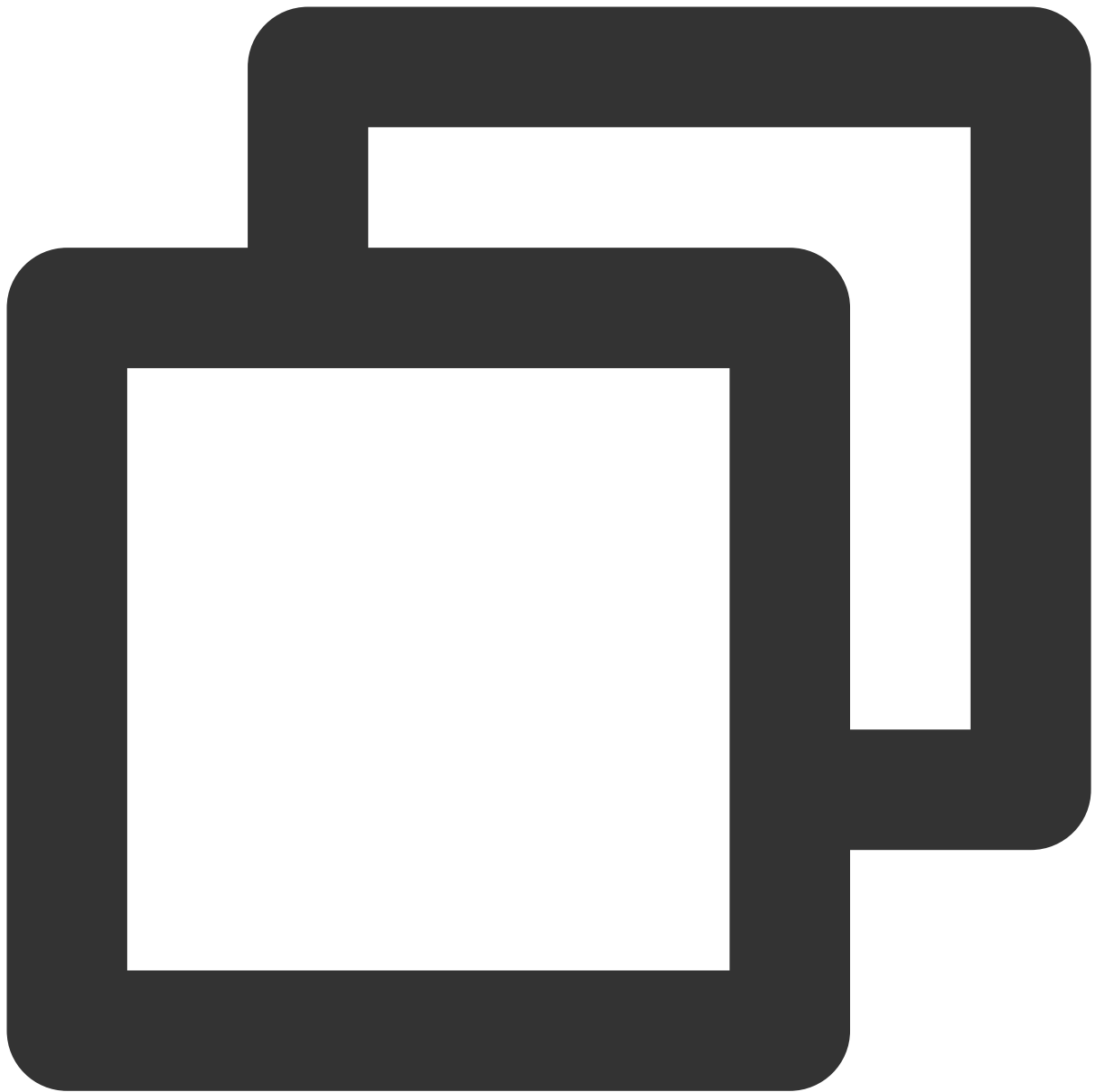
The public key required for verifying the Token carried on the client

Step 3: configure a Tencent Cloud API Gateway business API.

1. In the authorization API service, create a business API. When you are configuring the frontend, select **OAuth 2.0** as the authentication type, **Business API** as the OAuth mode, and the created authorization API as the associated authorization API.
2. When you are configuring the backend, select **mock** as the backend type and enter `hello world` as the returned data.

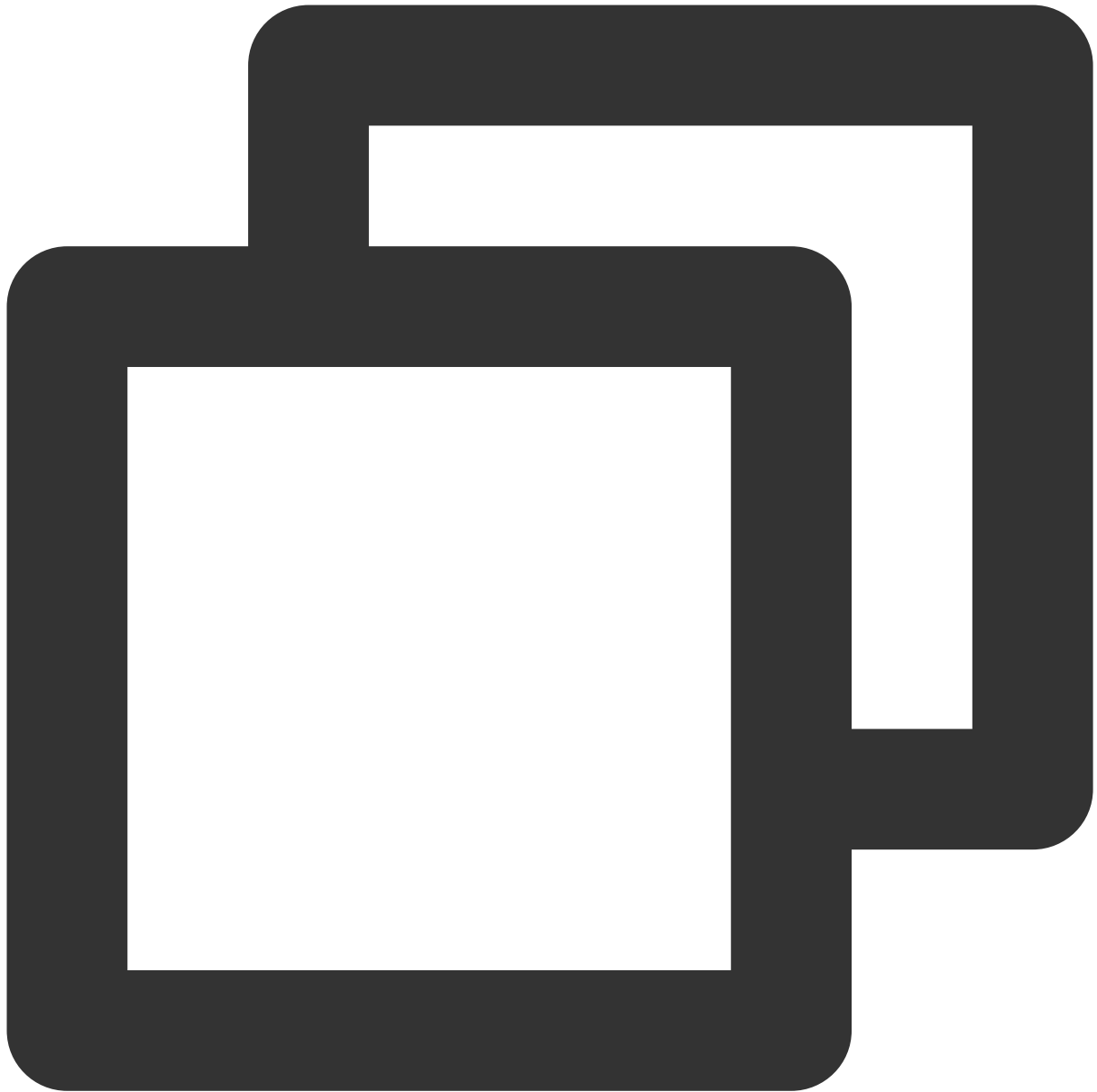
Step 4: perform verification.

1. Request the authorization API to get the token:



```
curl http://service-cmrrdq86-1251890925.gz.apigw.tencentcs.com:80/token
```

Returned result:

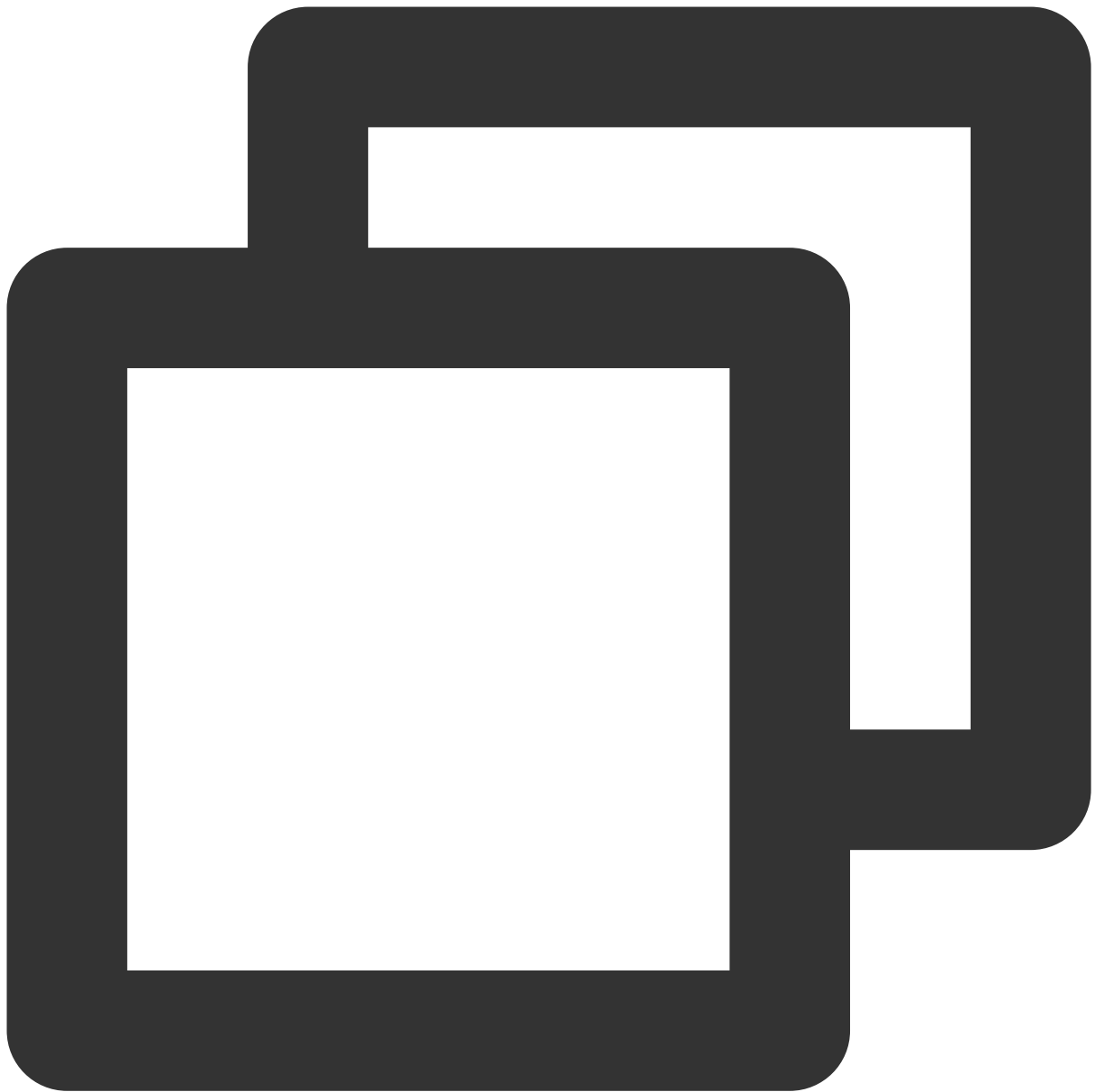


```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1OTIyNzk3MTAsImZvbyI6ImJhcnVzImlhdC
```

Note:

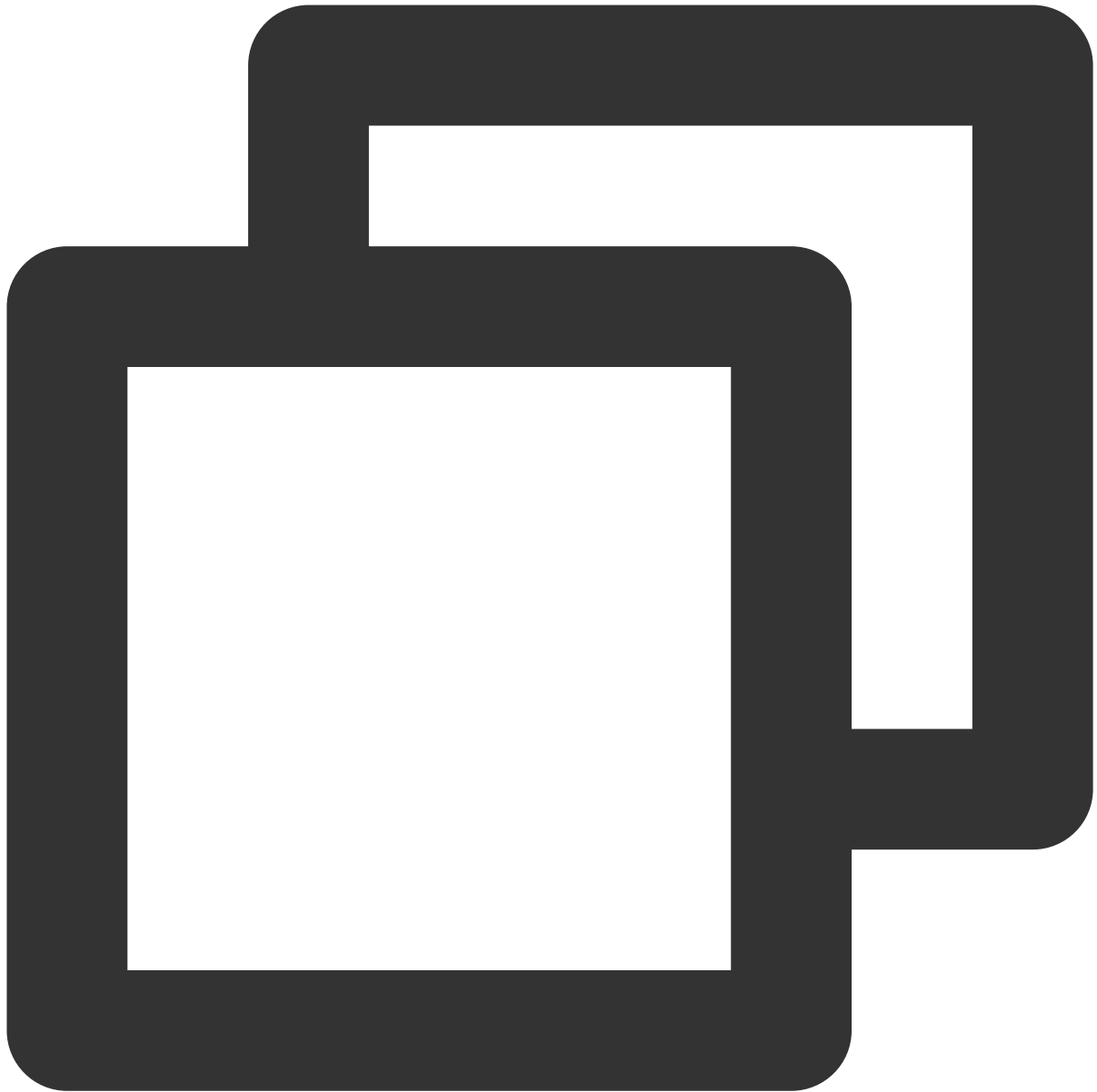
You can get the token using either of the following methods: 1. send a request to the API Gateway authorization API address to get the token; 2. quickly get the token directly from the authorization server. The first method is used in this document to protect the authorization server.

2. Use the token to request the business API. As you can see, the business API can be requested successfully.



```
curl http://service-cmrrdq86-1251890925.gz.apigw.tencentcs.com:80/work -H'Authoriza
```

Returned result:



```
hello world
```

Using the authorization code to get the token

In the sample above, no authorization code is used to get the token. To ensure that only specified users can get the token, the authorization code needs to be obtained from the resource owner according to the authorization process. As can be seen in the `server.py` file, you can first request the authorization code path to get the code and then register the distributed code to verify its validity when getting the token.

CAM Policy

Last updated : 2023-12-22 09:54:39

CAM Introduction

Basic concepts

The root account authorizes sub-accounts by binding policies. The policy setting can be accurate to **"API, Resource, User/User Group, Allow/Deny, Condition"**.

Account

Root account: As the fundamental owner of Tencent Cloud resources, root account acts as the basis for resource usage fee calculation and billing, and can be used to log in to Tencent Cloud services.

Sub-account: An account created by the root account, which has a specific ID and identity credential that can be used to log in to Tencent Cloud console. A root account can create multiple sub-accounts (users). **A sub-account does not own any resources by default, and must be authorized by its root**

account.Identity credential: Includes login credential and access certificate. **Login credential** refers to a user's login name and password, while **access certificate** refers to the Cloud API key (SecretID and SecretKey).

Resources and permission

Resource: Resources are objects that the cloud services operate on, such as the CVM instance, COS bucket and VPC instance.

Permission : Permission is an authorization to allow or forbid users to perform certain operations. By default, **root account has full access to all resources under the account**, while **sub-accounts do not have access to any resources under its root account**.

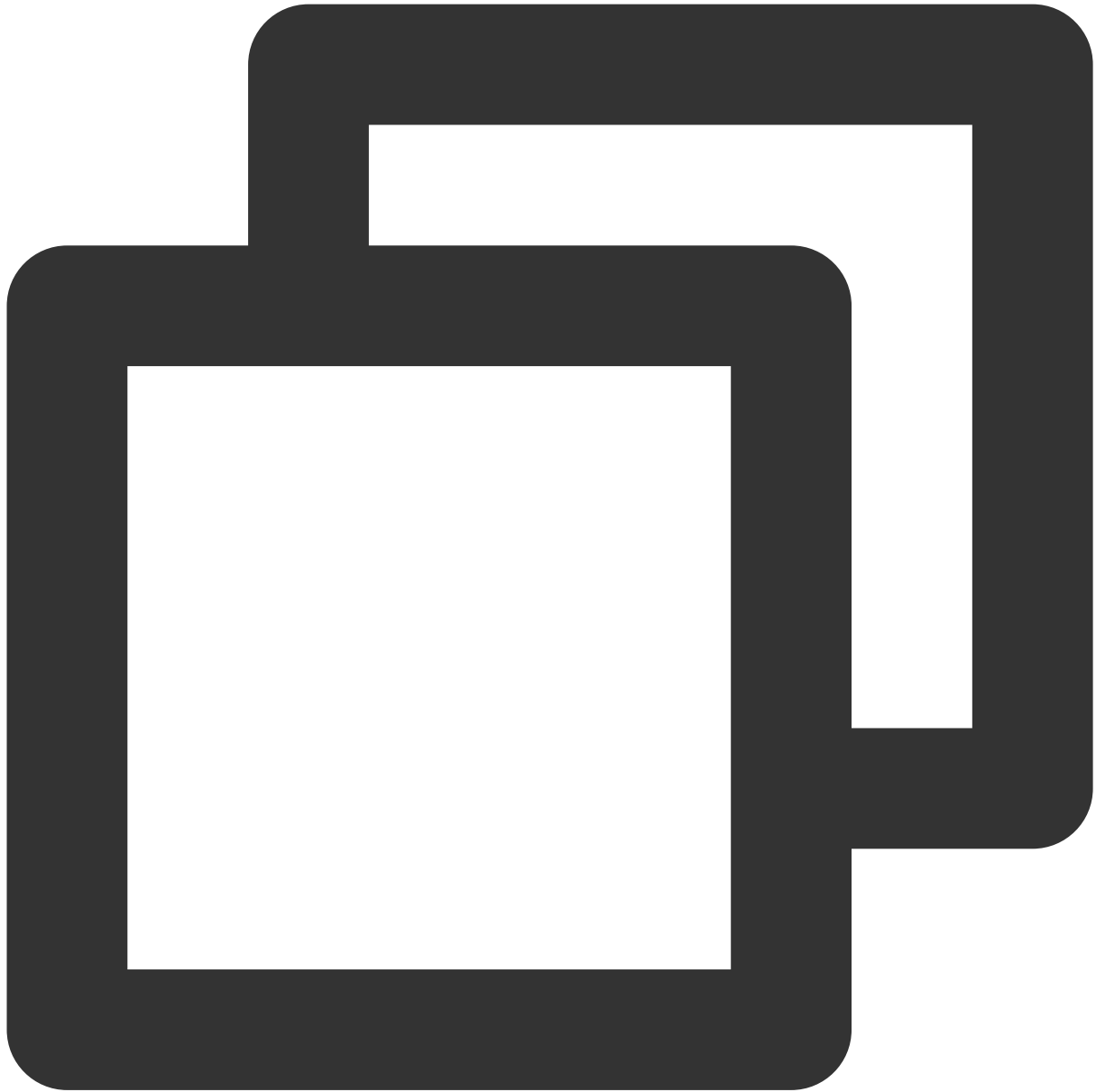
Policy : Policy is the syntax rule used to define and describe one or more permissions. **Root account** performs authorization by **associating policies** with users/user groups.

Related Documents

Content	Link
Relationship between policy and user	Policy Management
Basic structure of policy	Policy Syntax
More products that support CAM	CAM-Enabled Products

[Click to learn more about CAM](#)

API Gateway Resources



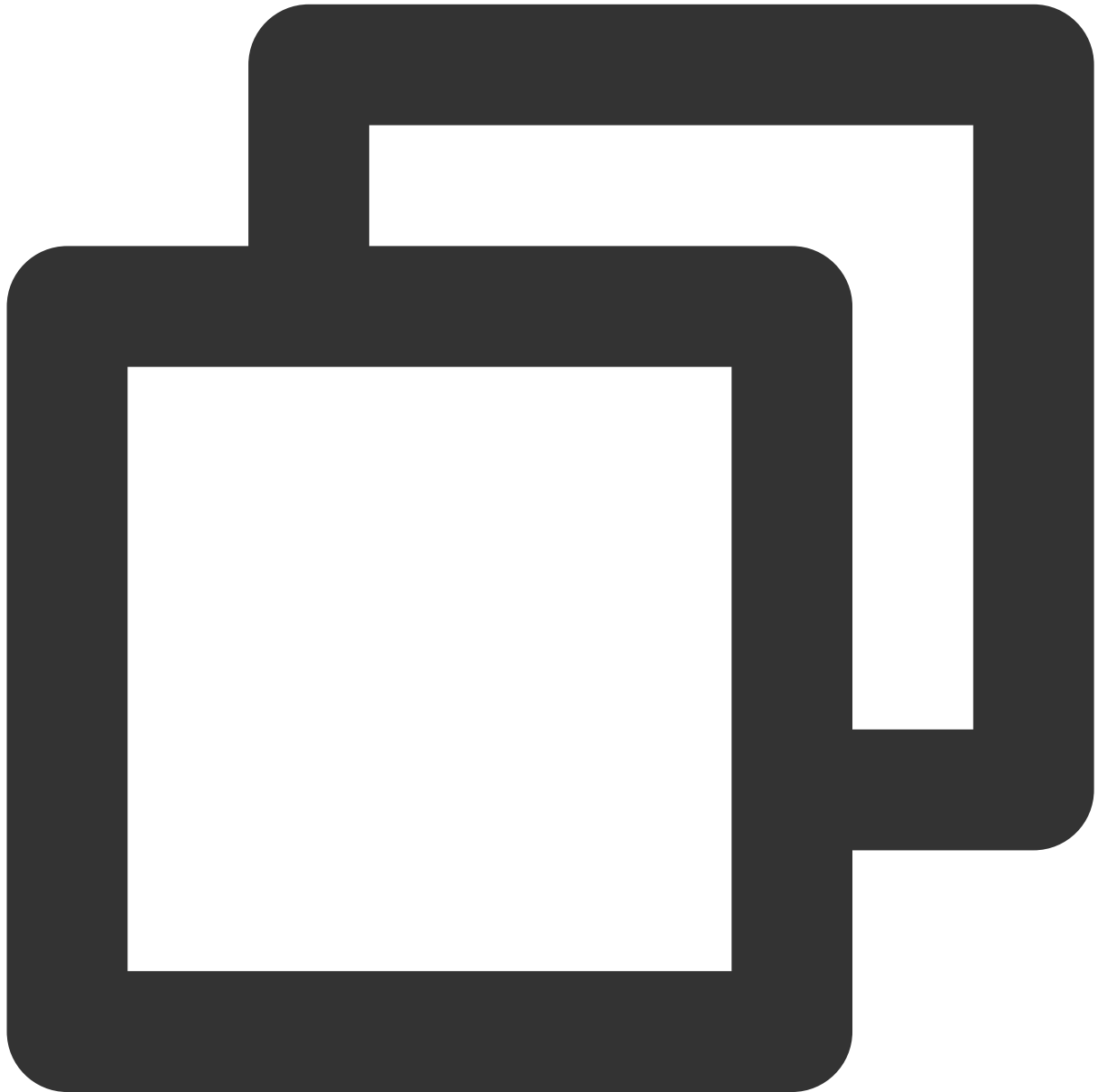
```
- qcs::APIgateway:_`region`_:uin/_`uin-id`_:service/_`serviceid`_  
- qcs::APIgateway:_`region`_:uin/_`uin-id`_:service/_`serviceid`_/API/_`apiid`_  
- qcs::APIgateway:_`region`_:uin/_`uin-id`_:usagePlan/_`usagePlanid`_  
- qcs::APIgateway:_`region`_:uin/_`uin-id`_:secret/_`secretid`_  
- qcs::APIgateway:_`region`_:uin/_`uin-id`_:IPStrategy/_`IPStrategyId`_  
- qcs::APIgateway:_`region`_:uin/_`uin-id`_:logRule/_`logRuleId`_
```

All creation APIs are at account level, while other APIs are at resource level.

CAM Policy Examples

Full read-write policy for any API Gateway resources

The following policy statement gives the sub-user permission to fully manage (creating, managing, etc.) any API services.



```
{  
  "version": "2.0",  
  "statement": [  
    {
```



```
"action": [
    "apigw:*"
],
"resource": "*",
"effect": "allow"
}
]
```

You can also configure the system's [full read-write policy](#) to support this permission.

Create by Policy Syntax

1 Select policy template > 2 Edit Policy

Template Type: System APIGW

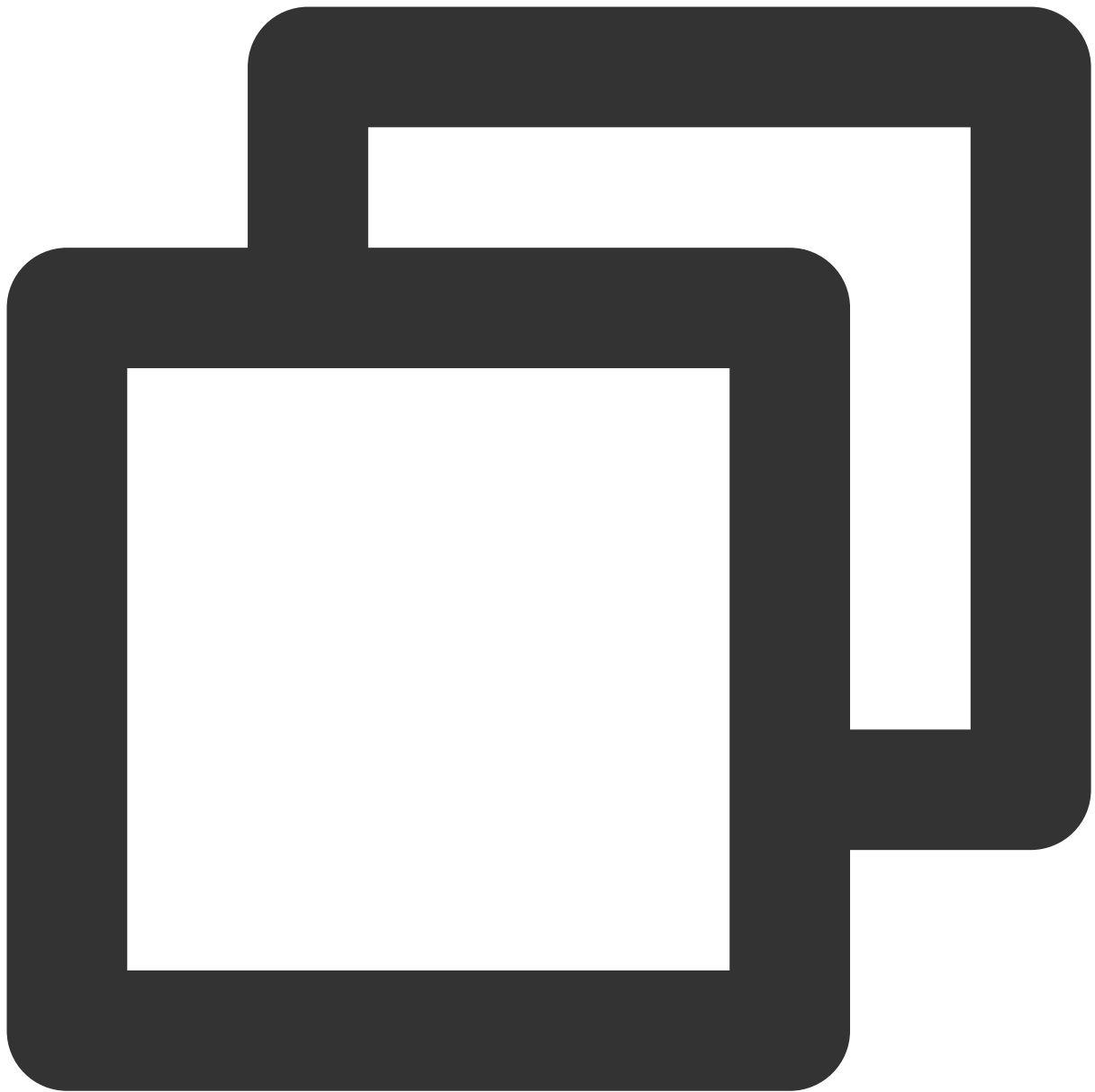
Select template type

System Template Search "APIGW", 2 result(s) found. [Back to the original list](#)

☐ QcloudAPIGWFullAccess System Full read-write access to API Gateway	☐ QcloudAPIGWReadOnlyAccess System Read-only access to API Gateway
---	---

Full management policy for single API Gateway service

The following policy statement gives the sub-user permission to fully manage (creating, managing, etc.) a specified API service:

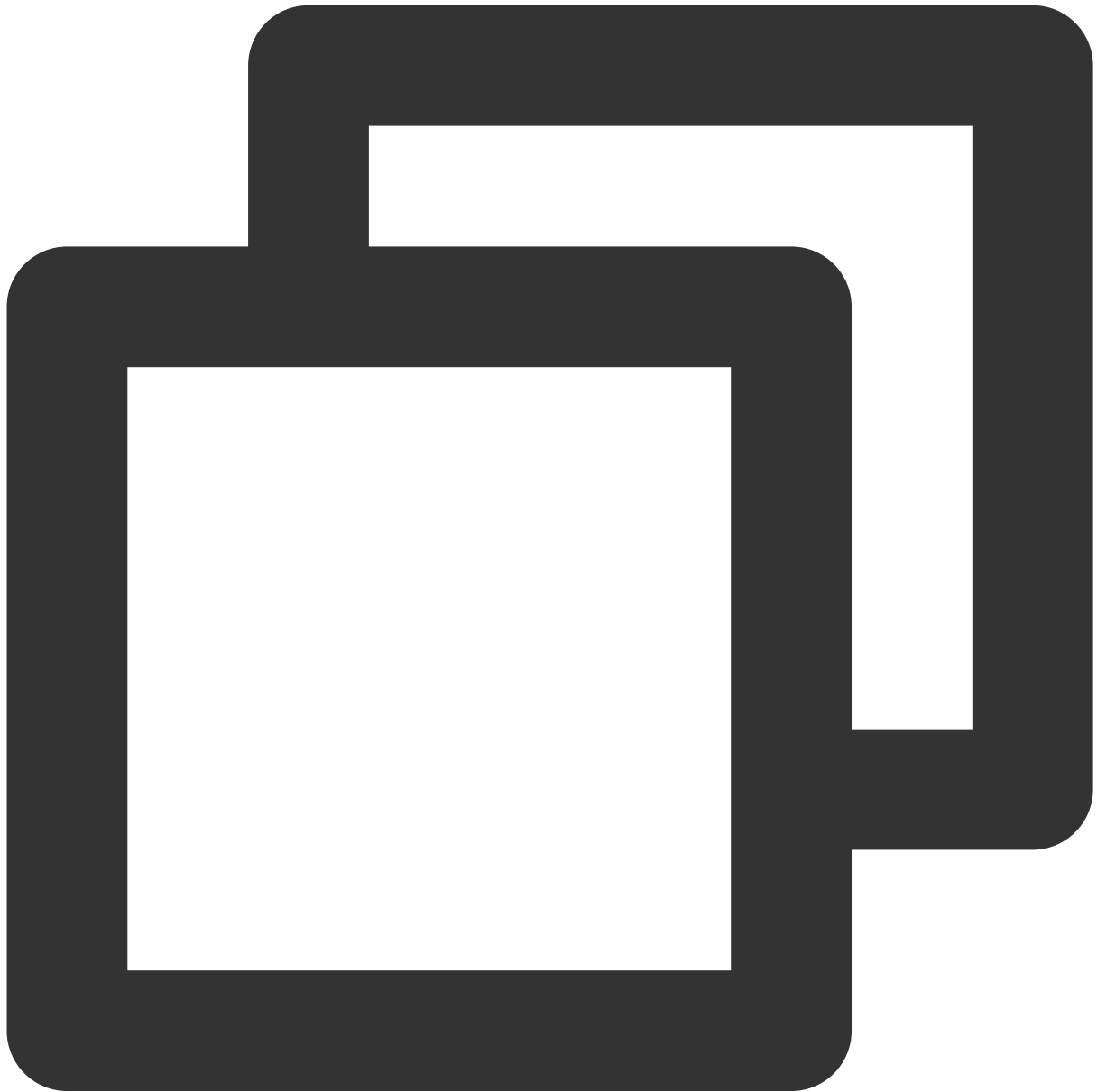


```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "apigw:*"
      ],
      "resource": "qcs::apigw:ap-guangzhou:uin/{ownerUin}:service/service-id/A",
      "effect": "allow"
    }
  ]
}
```

```
}
```

Read-only policy for single API Gateway service

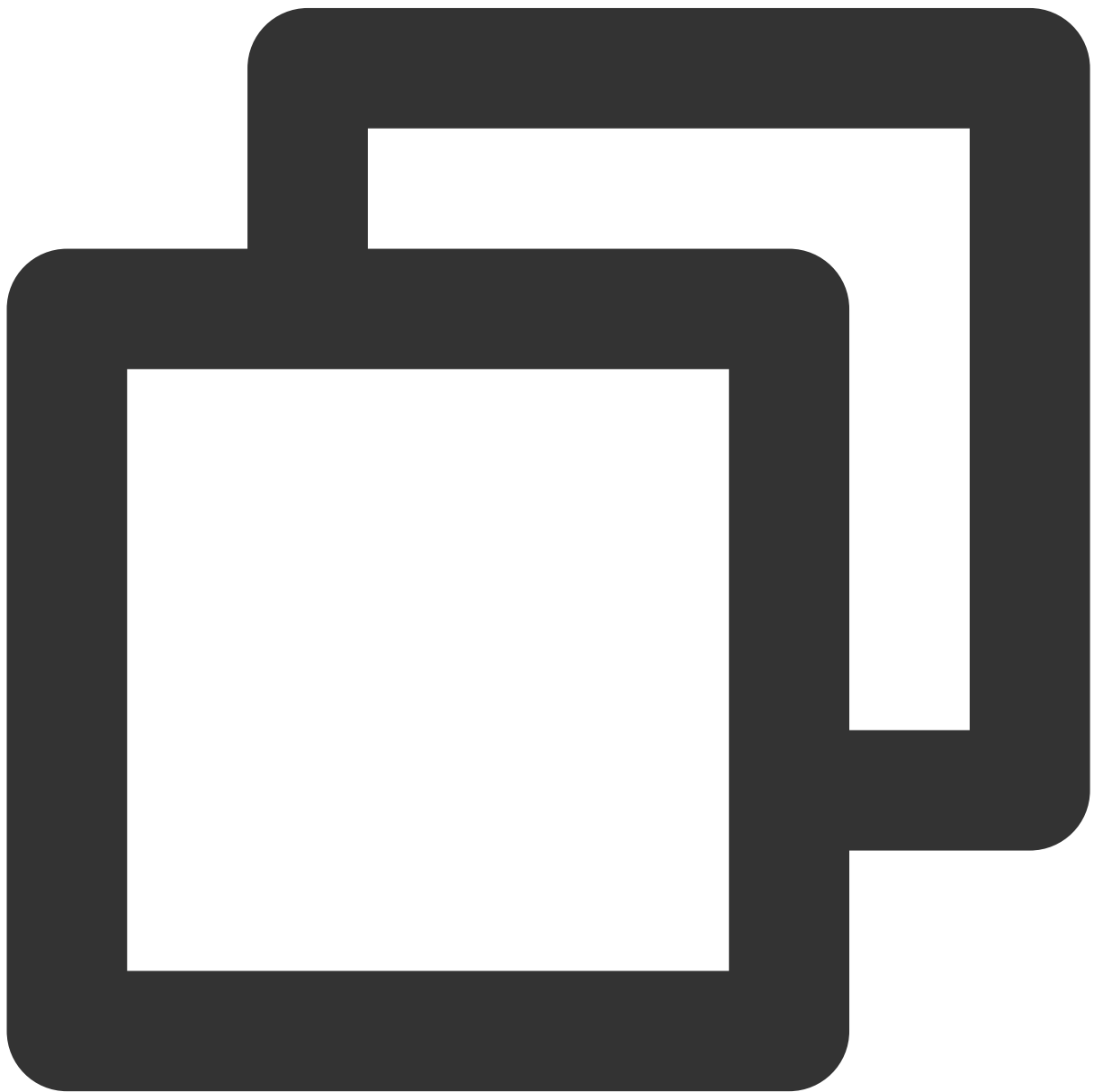
1. Create a policy with policy generator, and grant the permissions to list information for all resources and product monitoring. The following policy statement will grant read-only permission to all resources of the account.



```
{  
  "version": "2.0",  
  "statement": [  
    {
```

```
    "action": [  
        "apigw:Describe*",  
        "apigw:GenerateApiDocument"  
    ],  
    "resource": "*",  
    "effect": "allow"  
}  
  
]  
}
```

2. Grant read-only permission to a single API.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "ckafka:Get*",
        "ckafka:List*"
      ],
      "resource": "qcs::apigw:ap-guangzhou:uin/{ownerUin}:service/service-id/API",
      "effect": "allow"
    }
  ]
}
```