人脸支付

API 文档

产品文档

腾讯云

文档目录

API 文档
鉴权签名

最近更新时间：2019-08-07 15:34:25

拼接签名串

操作

签名与鉴权

人脸支付通过签名来验证请求的合法性。开发者将签名授权给客户端，使其具备上传下载及管理指定资源的能力。

签名分为两种：

- 多次有效签名：签名中绑定或者不绑定文件 fileid，需要设置大于当前时间的有效期，最长可设置三个月，在此期间内签名可多次使用。
- 单次有效签名：签名中绑定文件 fileid，有效期必须设置为 0，此签名只可使用一次，且只能应用于被绑定的文件。

具体应用参见 签名适用场景。

签名算法

获取签名所需信息

生成签名所需信息必须使用主账号的，包括 App ID、Secret ID 和 Secret Key。

注意：

- 如果您已使用过 API 密钥，或在2018年4月1日后接入人脸支付，请使用 API 密钥 。
- 如果您已使用过100、101等开头的项目ID，可以继续使用 项目密钥，但建议使用 API 密钥，2018年4月1日后创建的项目ID，不再支持使用 项目密钥。
- 目前仅支持使用主账号的 Secret ID 和 Secret Key，暂不支持子账号的使用，计划后续实现。

拼接签名串

拼接多次有效签名串：
a=[appid]&b=[bucket]&k=[SecretID]&e=[expiredTime]&t=[currentTime]&r=[rand]&f=[fileid]

拼接单次有效签名串：
a=[appid]&b=[bucket]&k=[SecretID]&e=[expiredTime]&t=[currentTime]&r=[rand]&f=[fileid]

注意：

- 多次有效签名串中 fileid 为可选参数。
- fileid 为空，表示不绑定资源，例如上传签名和下载签名。
- fileid 不为空，表示绑定资源，例如绑定资源的下载。

签名串中各字段含义如下：

字段	解释
a	开发者的 AppID，接入人脸支付时由系统生成。
b	Bucket，图片资源的组织管理单元，历史遗留字段，可不填。
k	Secret ID。
e	签名的有效期，是一个符合 UNIX Epoch 时间戳规范的数值，单位为秒；单次签名时，e 必须设置为 0。
t	当前时间戳，是一个符合 UNIX Epoch 时间戳规范的数值，单位为秒，多次签名时，e 应大于 t。
r	随机串，无符号 10 进制整数，用户需自行生成，最长 10 位。
f	资源存储的唯一标识，单次签名必填；多次签名选填，如填写则会验证与当前操作的文件路径是否一致。

注意：

- 拼接单次有效签名串时，有效期e必须设置为 0，以保证此签名只能针对固定资源使用一次。
- 删除和复制文件必须使用单次有效签名，上传必须使用多次有效签名。
- 具体应用参见 签名适用场景。

生成签名

1. 使用 HMAC-SHA1 算法对请求进行加密（SHA1算法加密后的输出必须是原始的二进制数据，否则签名失败）。
2. 对 orignal 使用 HMAC-SHA1 算法进行签名，然后将 orignal 附加到签名结果的末尾，再进行 Base64 编码，得到最终的 sign。
3. 生成签名的公式如下：
   SignTmp = HMAC-SHA1(SecretKey, orignal)
   Sign = Base64(SignTmp.orignal)

注意：

- 此处使用的是标准的 Base64 编码，不是 urlsafe 的 Base64 编码。
- SecretKey 为 API 密钥，orignal 为 拼接签名串 节中拼接好的签名串。

PHP 签名示例

本节介绍生成签名的算法实例，实例中使用 PHP 语言，如果开发者使用其他与开发，请使用对应的算法。

获取签名所需信息

获取得到的签名所需信息如下：

AppID：YOUR AppID_ID
Bucket：tencentyun（可不填）
Secret ID： YOUR SECRET_ID
Secret Key： YOUR SECRET_KEY

拼接签名串

```
$appid = "YOUR APPID_ID";
$bucket = "tencentyun";
$secret_id = "YOUR SECRET_ID";
$secret_key = "YOUR SECRET_KEY";
$expired = time() + 2592000;
$onceExpired = 0;
$current = time();
$rdm = rand();
$userid = "0";
$fileid = "tencentyunSignTest";

$srcStr = 'a='.$appid.'&b='.$bucket.'&k='.$secret_id.'&e='.$expired.'&t='.$current.'&r='.$rdm.'&f=';
```

```
$srcWithFile = 'a='.$appid.'&b='.$bucket.'&k='.$secret_id.'&e='.$expired.'&t='.$current.'&r='.$rdm.'&f='.$fileid;

$srcStrOnce= 'a='.$appid.'&b='.$bucket.'&k='.$secret_id.'&e='.$onceExpired .'&t='.$current.'&r='.$rdm
.'&f='.$fileid;
```

生成签名

SHA1 算法加密后的输出必须是原始的二进制数据，否则签名失败：

```
$signStr = base64_encode(hash_hmac('SHA1', $srcStr, $secret_key, true).$srcStr);

$srcWithFile = base64_encode(hash_hmac('SHA1', $srcWithFile , $secret_key, true).$srcWithFile );

$signStrOnce = base64_encode(hash_hmac('SHA1',$srcStrOnce,$secret_key, true).$srcStrOnce);

echo $signStr."\n";

echo $srcWithFile ."\n";

echo $signStrOnce."\n";
```

Java 签名示例

```
/*
* Copyright 2017, Tencent Inc
* All rights reserved.
*
* Created on 2017年9月12日
*/
package sign;

import java.util.Base64;
import java.util.Random;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class Sign {

/**
```

```
 * 生成 Authorization 签名字段
 *
 * @param appId
 * @param secretId
 * @param secretKey
 * @param bucketName
 * @param expired
 * @return
 * @throws Exception
 */
public static String appSign(long appId, String secretId, String secretKey, String bucketName,
long expired) throws Exception {
long now = System.currentTimeMillis() / 1000;
int rdm = Math.abs(new Random().nextInt());
String plainText = String.format("a=%d&b=%s&k=%s&t=%d&e=%d&r=%d", appId, bucketName,
secretId, now, now + expired, rdm);
byte[] hmacDigest = HmacSha1(plainText, secretKey);
byte[] signContent = new byte[hmacDigest.length + plainText.getBytes().length];
System.arraycopy(hmacDigest, 0, signContent, 0, hmacDigest.length);
System.arraycopy(plainText.getBytes(), 0, signContent, hmacDigest.length,
plainText.getBytes().length);
return Base64Encode(signContent);
}

/**
 * 生成 base64 编码
 *
 * @param binaryData
 * @return
 */
public static String Base64Encode(byte[] binaryData) {
String encodedstr = Base64.getEncoder().encodeToString(binaryData);
return encodedstr;
}

/**
 * 生成 hmacsha1 签名
 *
 * @param binaryData
 * @param key
 * @return
 * @throws Exception
 */
public static byte[] HmacSha1(byte[] binaryData, String key) throws Exception {
Mac mac = Mac.getInstance("HmacSHA1");
```

```
SecretKeySpec secretKey = new SecretKeySpec(key.getBytes(), "HmacSHA1");
mac.init(secretKey);
byte[] HmacSha1Digest = mac.doFinal(binaryData);
return HmacSha1Digest;
}

/**
* 生成 hmacsha1 签名
*
* @param plainText
* @param key
* @return
* @throws Exception
*/
public static byte[] HmacSha1(String plainText, String key) throws Exception {
return HmacSha1(plainText.getBytes(), key);
}

}
```

Node JS 签名示例

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

var secretId = 'YOUR SECRET_ID',
secretKey = 'YOUR SECRET_KEY',
appid = 'APPID',
pexpired = 86400,
userid = 0;

var now = parseInt(Date.now() / 1000),
rdm = parseInt(Math.random() * Math.pow(2, 32)),
plainText = 'a=' + appid + '&k=' + secretId + '&e=' + (now+pexpired) + '&t=' + now + '&r=' + rdm
+ userid + '&f=',
data = new Buffer(plainText,'utf8'),
res = crypto.createHmac('sha1',secretKey).update(data).digest(),
bin = Buffer.concat([res,data]);

var sign = bin.toString('base64');
```

C++ 签名示例

```
//g++ -g sign_sample.cpp -o sign -lcrypto

#include <stdio.h>
#include <stdlib.h> /* srand, rand */
#include <time.h> /* time */
#include <openssl/hmac.h>
#include <openssl/pem.h>
#include <openssl/bio.h>
#include <openssl/evp.h>
#include <string>
#include <vector>
#include <sstream>

#define HMAC_LENGTH 20

std::vector<unsigned char> hmac_sha1(std::string& data, std::string& key)
{
unsigned char* result;
unsigned int len = HMAC_LENGTH;

result = (unsigned char*)malloc(sizeof(char) * len);

HMAC_CTX ctx;
HMAC_CTX_init(&ctx);

HMAC_Init_ex(&ctx, key.c_str(), key.length(), EVP_sha1(), NULL);
HMAC_Update(&ctx, (unsigned char*)data.c_str(), data.length());
HMAC_Final(&ctx, result, &len);
HMAC_CTX_cleanup(&ctx);

std::vector<unsigned char> sha1;
for (int i = 0; i < len; i++){
sha1.push_back(result[i]);
}

free(result);
return sha1;
}

std::string base64_encode(const std::string& src){
BUF_MEM * bptr = NULL;
BIO* b64 = BIO_new(BIO_f_base64());
```

```
BIO_set_flags(b64, BIO_FLAGS_BASE64_NO_NL);
BIO* bmem = BIO_new(BIO_s_mem());
if(NULL == b64 || NULL == bmem){
return "";
}

bmem = BIO_push(b64, bmem);
int ret = BIO_write(bmem, src.data(), src.length());
if(ret <= 0){
return "";
}

ret = BIO_flush(bmem);
BIO_get_mem_ptr(bmem, &bptr);
std::string res(bptr->data, bptr->length);
BIO_free_all(bmem);
return res;

}


int main() {
std::string appid = "1000001";
std::string secret_id = "YOUR SECRETID";
std::string secret_key = "YOUR SECRETKEY";
time_t now = time(NULL);
long expired = (long)now + 2592000;
long onceExpired = 0;
long current = (long)now;
int rdm = rand();
std::string userid = "0";

std::stringstream raw_stream;
raw_stream << "a=" << appid << "&k=" << secret_id << "&e=" << expired << "&t=" << current
<< "&r=" << rdm;
std::string raw = raw_stream.str();

std::vector<unsigned char> sha1 = hmac_sha1(raw, secret_key);

std::stringstream data_stream;
for (int i = 0; i != sha1.size(); i++)
data_stream << sha1[i];
data_stream << raw;
std::string data = data_stream.str();
```

```
std::string sign = base64_encode(data);

printf("%s\n", sign.c_str());

return 0;
}
```

签名适用场景

签名的适用场景有如下限制：

场景	适用签名
智能鉴黄	多次有效签名
图片标签	多次有效签名
OCR识别	多次有效签名
人脸识别	多次有效签名
人脸核身	多次有效签名
人脸融合	多次有效签名

错误码说明

最近更新时间：2019-08-07 15:32:41

人脸错误码说明

错误码	含义
3	错误的请求；其中 message:account abnormal,errorno is:2 为账号欠费停服
4	签名为空
5	签名串错误
6	签名中的 appid/bucket 与操作目标不匹配
9	签名过期
10	appid 不存在
11	secretid 不存在
12	appid 和 secretid 不匹配
13	重放攻击
14	签名校验失败
15	操作太频繁，触发频控
16	Bucket 不存在
21	无效参数
23	请求包体过大
107	鉴权服务内部错误
108	鉴权服务不可用
213	内部错误
-1101	人脸检测失败
-1102	图片解码失败

错误码	含义
-1103	特征处理失败
-1104	提取轮廓错误
-1105	提取性别错误
-1106	提取表情错误
-1107	提取年龄错误
-1108	提取姿态错误
-1109	提取眼镜错误
-1200	特征存储错误
-1300	图片为空
-1301	参数为空
-1302	个体已存在
-1303	个体不存在
-1304	参数过长
-1305	人脸不存在
-1306	组不存在
-1307	组列表不存在
-1308	url 图片下载失败
-1309	人脸个数超过限制
-1310	个体个数超过限制
-1311	组个数超过限制
-1312	对个体添加了几乎相同的人脸
-1400	非法的图片格式
-1403	图片下载失败