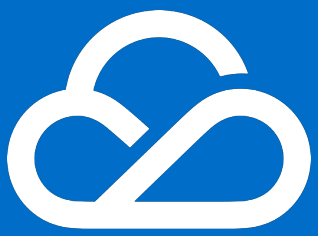
云通信

账号登录集成

产品文档



腾讯云

【版权声明】

【商标声明】

【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模式的承诺或保证。

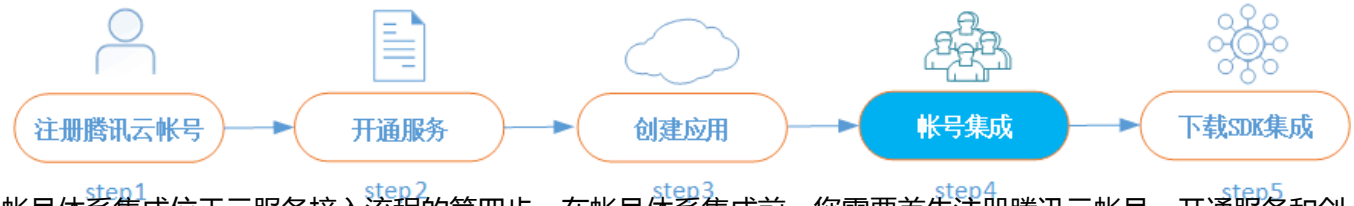
文档目录

账号登录集成说明

1 帐号集成简介

腾讯登录服务（Tencent Login Service，TLS）是腾讯为开发者快速完成帐号集成接入音视频或即时通信云服务（后面简称云服务）而提供的一套通用帐号登录组件。

1.1 简要说明



帐号体系集成位于云服务接入流程的第四步。在帐号体系集成前，您需要首先注册腾讯云帐号，开通服务和创建应用，具体方法请点击“应用接入指引”。 本文后续将重点讲解如何进行帐号集成。

2 两种模式介绍

1. 如果您的应用已经发布，用户较多，独立模式可以帮您快速集成云服务（
   温馨提醒：帐号的独立模式并不影响资料、关系链和群组的托管）；
2. 如果您不想维护复杂的用户帐号系统，托管模式可以帮您方便快速地搭建安全的应用自有帐号体系。

3 附录

3.1 集成模式

根据开发者使用场景，我们为开发者提供了两种不同的帐号集成方式，开发者可根据自身情况选择：

1. 独立模式：用户帐号信息由开发者保存，用户身份验证（比如注册与验密）也由开发者负责；
2. 托管模式
   ：由腾讯为开发者提供帐号的密码注册、存储和密码验证，以及第三方openid和token的托管验证服务。

注意：帐号集成模式选择后，就不能变更了，所以选择时请慎重。如果想改变集成模式，请创建新的应

用重新走帐号集成流程。

3.2 公私钥

公私钥是密码学中非对称加密算法中的概念。非对称加密算法中，公钥与私钥是成对出现的，私钥需要保密，公钥是公开的。用公钥加密的数据，只有私钥才能解开；用私钥加密的数据，也只有公钥才能解开。在帐号集成中，公私钥用于鉴别开发者用户的身份合法性，使用场景如下：

1. 独立模式
   ：私钥由开发者保存，公钥由腾讯保存。开发者使用私钥生成用户签名UserSig，腾讯使用公钥对签名UserSig进行校验；
2. 托管模式
   ：私钥由腾讯保存，公钥由开发者保存。腾讯使用私钥生成用户签名UserSig，开发者可以使用公钥对签名UserSig进行校验，注意，对于第三方开放帐号，此时不需要公私钥。

对于独立模
式，为了方便开发者快
速开发，开发者进行帐号集成后，可以在应用
列表的应用配置中下载公私钥。同时我们也提供了工具给开发者（详见TLS后台API开发指引）供调试使用。
关于非对称算法，可通过以这篇文章进一步了解。

3.3 如何生成公私钥

详见“TLS后台API开发指引”中的“工具使用”章节。

3.4 APP管理员

对外的开放接口中，有些操作是管理员身份才能调用的，例如解散群，给用户发C2C消息等。因此在开发者的帐号体系中，可以标识一些帐号为管理员，在腾讯验证管理员身份后，管理员可以对自己的业务进行一些特殊操作。
APP管理员是对APP具有最高管理权限的角色，与普通帐号相比，其：

1. 读取权限更高，例如获取APP内部的所有群组、获取任意群组的任意资料；
2. 操作权限更高：例如给任意用户发消息、在任意群组中增删成员。

APP在调用REST API进行服务端集成时，应当使用APP管理员帐号作为identifier进行调用。APP可以在帐号体系中将数个帐号设置为APP管理员，设置方法参见这里。

3.5 短信验证码内容

开发者选择托管模式集成自有帐号后，由腾讯为开发者提供帐号的注册和登录服务。在开发注册和功能时，开发者可以选择让用户通过短信验证的方式进行注册和登录。验证码短信格式如下：
1234（XXXX短信验证码），请勿转发，否则会导致帐号被盗。【腾讯云】
【说明】

1. 以上内容开发者只能更改“XXXX”部分，可填写公司名称或者app名称，长度30个字符以内；
2. “1234”为随机的四位数字，实际使用时会随机生成；
3. 【腾讯云】为默认短信签名（应运营商要求，下发短信必须要有短信签名）；如果您的月短信量超过 1 万条，可以联系我们给您定制短信签名。
4. 如果开发者没有填写“XXXX”部分，则使用默认的短信验证码，格式如下：
   1234（短信验证码），请勿转发，否则会导致帐号被盗。【腾讯云】

3.6 identifier 格式说明

identifier 即用户帐号标识，下面是对此参数的格式说明：
独立模式下，identifier 长度建议不超过 32 字节 。托管模式下，字符串类型的 identifier 长度为4~24个字节，请使用英文字符和下划线，不能全为数字，大小写不敏感。

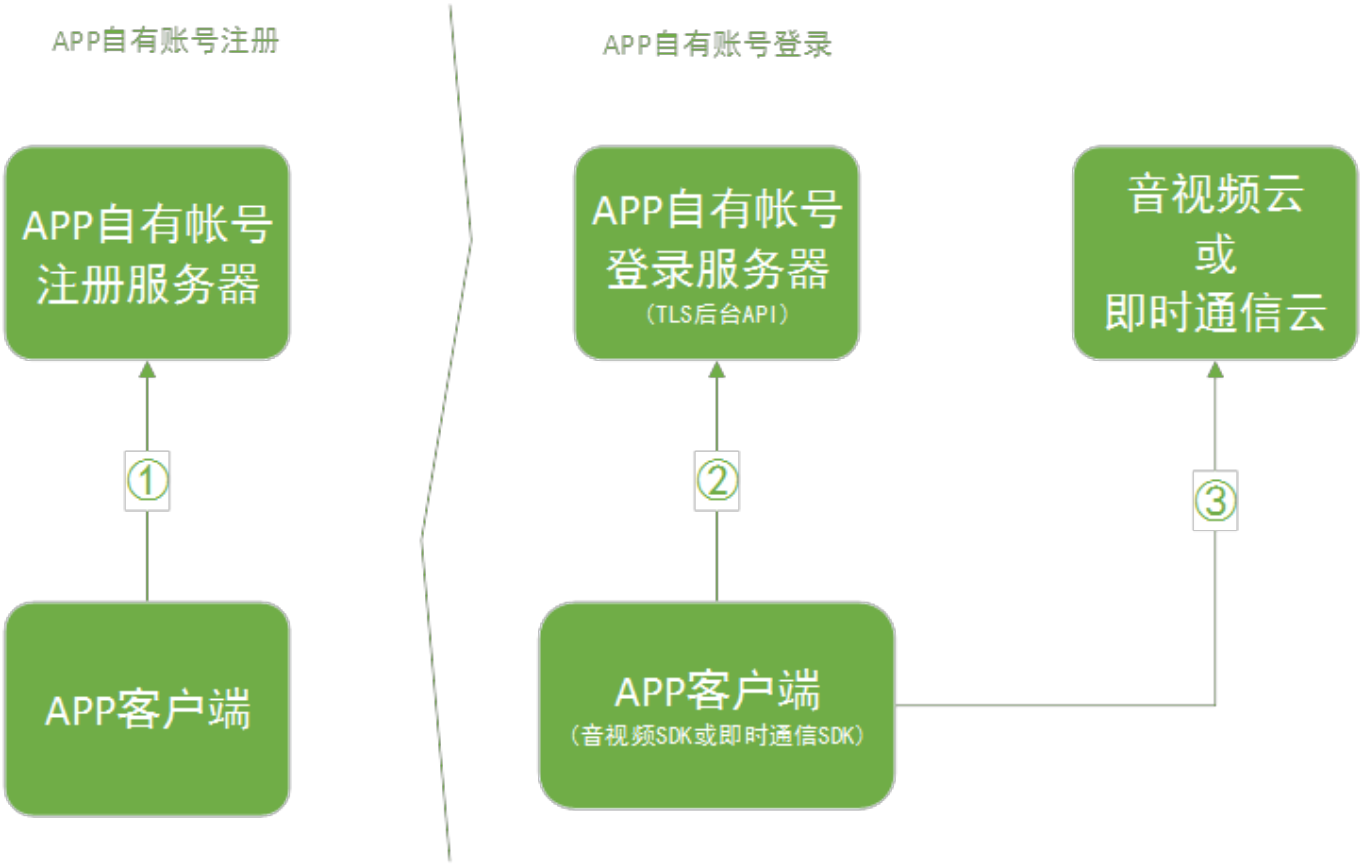
4 联系我们

如果遇到问题建议先到这里查找解决方法，如仍需支持，请@TLS帐号支持，QQ 3268519604，电子邮箱 tls_assistant@tencent.com。

独立模式

独立模式是指，用户注册和身份验证由开发者负责，开发者和腾讯之间通过签名验证建立信任关系。开发者在申请接入时，直接下载公私钥用于开发，而后用私钥加密指定数据生成签名交由腾讯服务器验证合法性。

1 APP自有帐号



【注册说明】

帐号注册在APP自有注册服务器完成，帐号密码无需同步到腾讯。

【登录说明】

用户在APP客户端输入帐号密码后到APP自有帐号登录服务器验证，验证成功后，APP自有帐号登录服务器使用私钥派发签名（UserSig）给客户端；

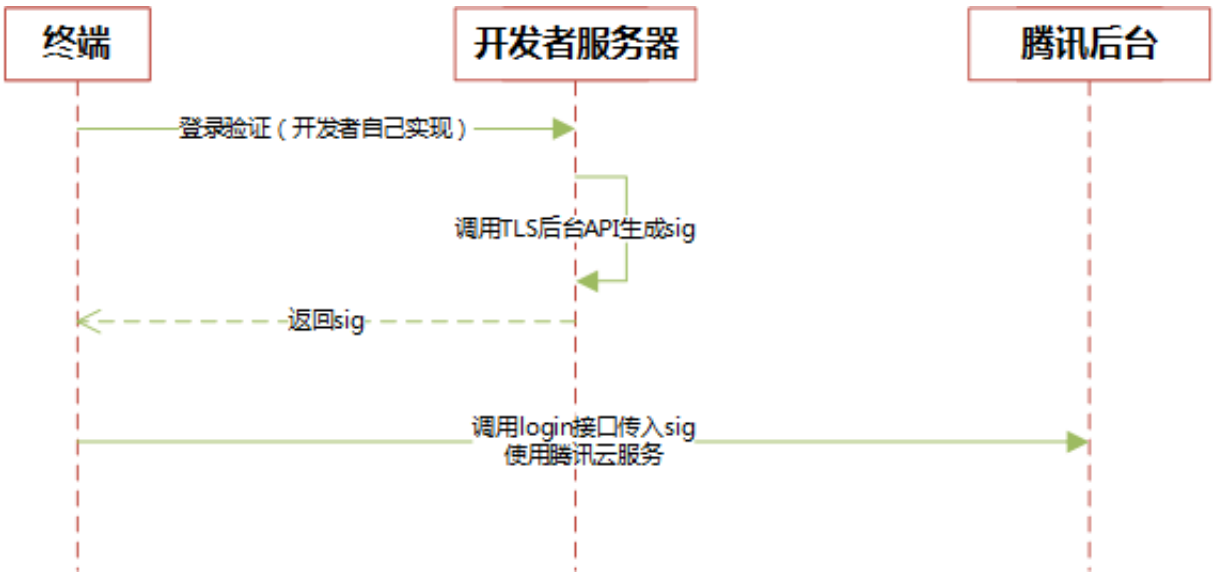
APP客户端使用用户帐号和私钥签名，调用音视频SDK或即时通信SDK的接口进行验证，验证成功后即可使用音视频云或即时通信云服务。

【使用说明】

开发者需要保证私钥安全，腾讯完全信赖私钥签名；

TLS后台API默认接口生成的签名有效期为180天，开发者可以使用含有有效期参数的接口自行设定有效期，开发者需要在签名过期前到开发者后台获取新的签名，TLS后台API详见这里。

【典型流程】



2 第三方开放帐号

开发者集成第三方开放帐号，流程上与自有帐号的集成方式一致，在腾讯这一侧使用的都是identifier（用户id）和 UserSig。

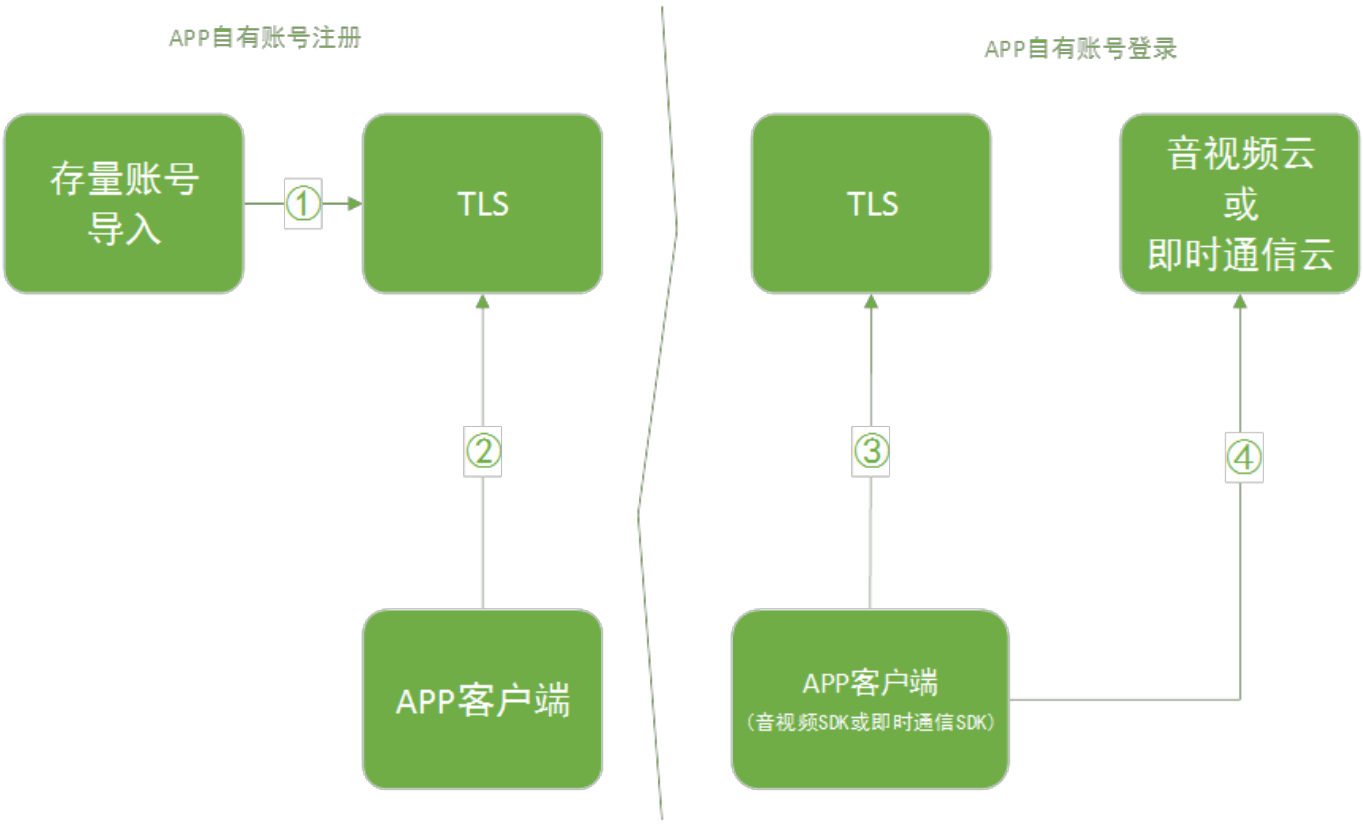
3 联系我们

如果遇到问题建议先到这里查找解决方法，如仍需支持，请@TLS帐号支持，QQ 3268519604，电子邮箱 tls_assistant@tencent.com。

托管模式

托管模式是指，由TLS为开发者提供APP帐号的密码注册、存储和密码验证。帐号验证成功后，派发私钥加密生成的签名到客户端，APP业务服务器可以通过下载的公钥解密签名进行验证。

1 APP自有帐号



【注册说明】

1. APP自有帐号注册有2个接口：
2. （可选）提供接口支持存量帐号导入；
3. 新帐号注册使用TLS SDK完成。

【登录说明】

1. 用户在客户端输入帐号密码后，到TLS验证；
2. 登录后可以直接使用音视频或者即时通信云服务；
3. （可选）APP应用服务器可以使用TLS后台API对签名进行验证，用以确定用户的合法性。

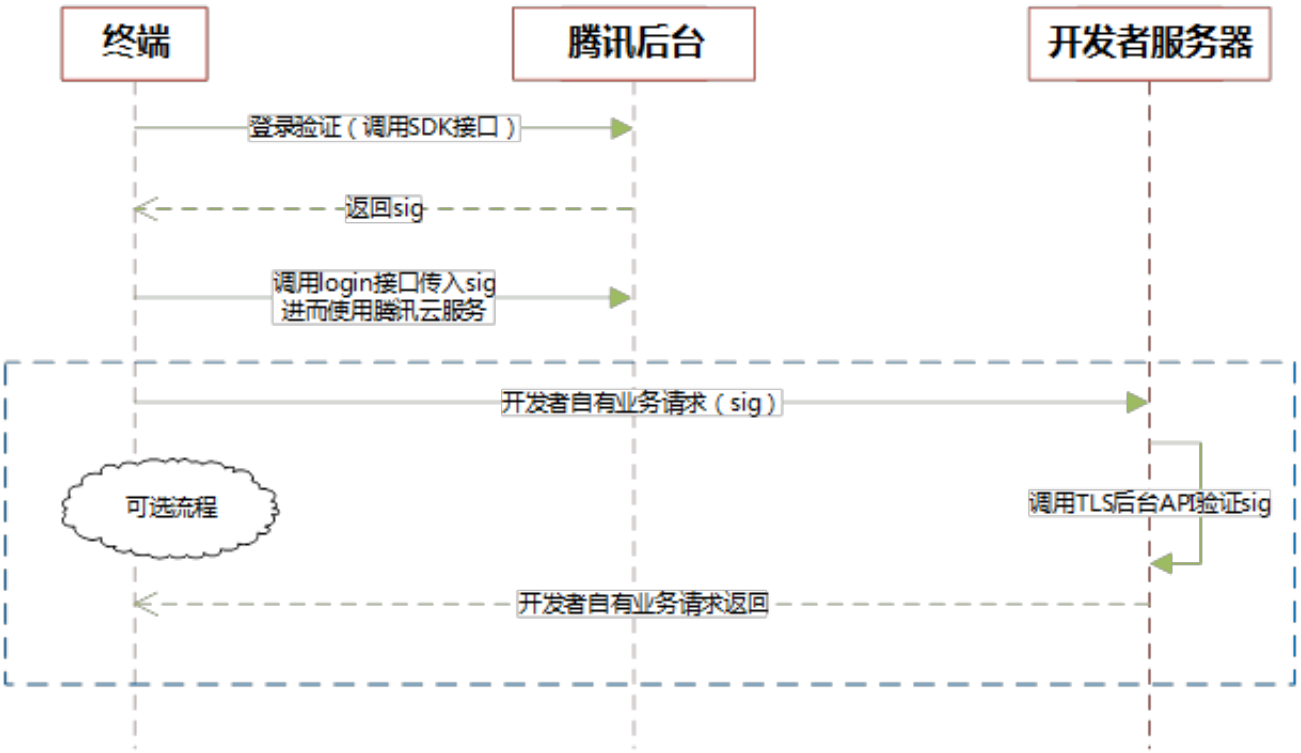
【使用说明】

1. 直接使用TLS SDK即可快速完成注册和登录能力集成；
2. 已上线的APP也可以通过存量帐号导入使用托管模式。

【安全性说明】

帐号密码存储安全级别与QQ帐号相同，能够保证拖库后帐号安全。

【典型流程】



2 开发文档下载

TLS SDK 开发手册（android）

TLS SDK 开发手册（ios）

3 联系我们

如果遇到问题建议先到这里查找解决方法，如仍需支持，请@TLS帐号支持，QQ 3268519604，电子邮箱 tls_assistant@tencent.com。

TLS后台API使用手册

1 概述

开发者可以使用TLS后台API及相关工具，生成公私钥、生成UserSig和校验UserSig。TLS后台API我们提供了6个包供开发者下载
，内容分别是windows下64位预编译文件包、windows下32位预编译文件包、linux下64位预编译文件包、linux下32位预编译文件包、zip格式的源代码文件和tar.gz格式的源代码文件。

2 linux平台

2.1 工具使用

注：这里讲解的是工具的使用说明，实际应用中需要开发者后台调用tls的后台api接口生成sig。

linux下生成sig和校验sig

首先不带参数执行 tls_licence_tools，即执行下面的命令：

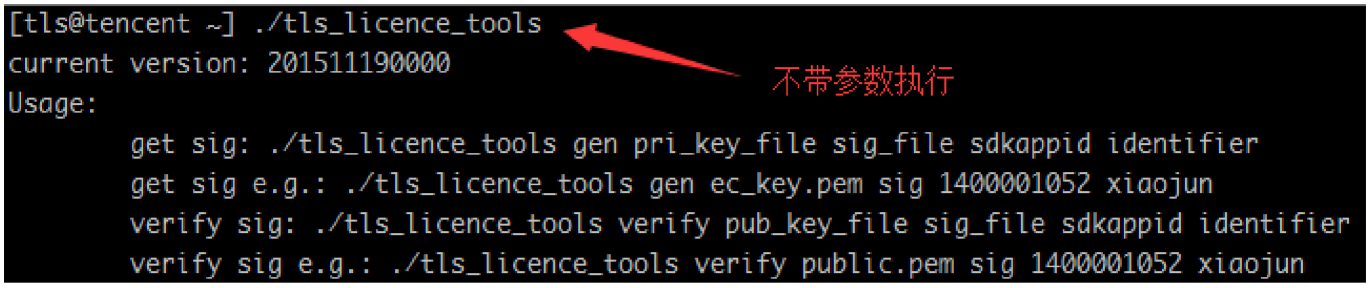
```
$ ./tls_licence_tools
```

输出：

```
current version: 201511190000
Usage:
    get sig: ./tls_licence_tools gen pri_key_file sig_file sdkappid identifier
    get sig e.g.:./tls_licence_tools gen ec_key.pem sig 1400001052 xiaojun
    verify sig:./tls_licence_tools verify pub_key_file sig_file sdkappid identif
ier
    verify sig e.g.: ./tls_licen
```

```
ce_tools verify public.pem sig 1400001052 xiaojun
```

下面是演示截图：



输出实际上是参数模板和示例。

执行类似于下面的命令可以生成 sig：

```
./tls_licence_tools gen ec_key.pem sig 1400001052 xiaojun
```

对应的参数解释是：

```
./tls_licence_tools gen 私钥文件路径 sig将要存放的路径 sdkappid 用户id
```

执行类似于下面的命令可以校验 sig：

```
./tls_licence_tools verify public.pem sig 1400001052 xiaojun
```

对应参数的解释是：

```
./tls_licence_tools verify 公钥文件路径 sig的存放路径 sdkappid 用户id
```

下面是生成sig演示截图：

```
[tls@tencent ~] ./tls_licence_tools gen ec_key.pem sig 1400001052 xiaojun
generate sig ok
[tls@tencent ~] cat sig
eJxlj0tvgkAYRff8CjLrpp3hkUJ3SjRaaWSkVboiPAb50A7DMIqP9L*3pSYl6fqcm3vvVdN1Hb364X2SZfWBq1idBUP6k44wuvuDQkAeJyo2Zf4Psp
MAyeKkUEz20LBdA*OhAjnjCgq4CSdI6urAB0Kb7*K*pOfE*o5jgm1jqMC2hy*Td29OPT8KqXwG7G9hzZebcHNRTlvRYlbQLIjGwdF72Her5WKa0nk5
Ch6nC7E7O6Ou4Zey6mZvQbRKw0b5VtjJCd235lqY5bgh6aBSwQe7DbIc23QNwxrQI5Mt1Pz3MiY2IcT92Y20T*0Ld8VfPg__[tls@tencent ~]
```

上面是生成sig，下面是校验sig：

```
[tls@tencent ~] ./tls_licence_tools verify public.pem sig 1400001052 xiaojun
verify sig ok
[tls@tencent ~]
```

下面解释下参数模板中参数的意义：

`gen`和`verify`分别表示生成`sig`和校验`sig`的命令

`pri_key_file`: 私钥文件的路径

`pub_key_file`: 公钥文件的路径

`sig_file`: `sig` 文件的路径，如果是生成 `sig`，那么会将 `sig` 写入这个文件，如果是校验 `sig`，那么会从这个文件读取 `sig` 的内容

`sdkappid`: 创建应用时页面上分配的 `sdkappid`

`identifier`: 用户标识，即用户`id`

注意：生成的sig有效期为180天，开发者需要在sig过期前，重新生成sig。

2.2 C++接口

首先包含include/tls_sig_api目录下的tls_signature.h。头文件中包含的接口，tls_gen_signature_ex2和tls_check_signature_ex2，前者是生成sig的接口，后者是校验sig的接口，详细的参数和返回值说明请参考头文件tls_signature.h。
然后是链接静态库，在lib目录下有下列目录：

├── jni
├── jsoncpp
├── openssl

└── tls_sig_api

需要链接的静态库是libjsoncpp.a、openssl目录下的libcrypto.a和libtlsignature.a。另外还需要链接系统的-ldl和-lz，详细可以查看example/cpp/Makefile，由于libtlsignature.a引用了openssl和json的开发库，所以链接时libtlsignature.a出现命令的最前面。

下面的截图是我们开发时编译tls_licence_tools的命令行，由于是我们这边的开发环境，链接库的路径可以按照开发者自己的实际情况给出。

```
[tls@tencent ~] make
g++ -g -I../../include -I../../include/tls_sig_api -Wall -fPIC -c tls_licence_tools.cpp
g++ tls_licence_tools.o ../../src/libtlsignature.a -o -Wall tls_licence_tools -g -I../../include -I../../include/t
ls_sig_api -Wall -fPIC -L../../linux/64/lib/jsoncpp ../../openssl-dynamic/lib/libcrypto.a -ljsoncpp -ldl -lz
```

【特别注意】
如果程序有多线程调用TLS后台API的用法，请在程序初始化时和结束时分别调用下面的接口：

```
int multi_thread_setup(void);
void multi_thread_cleanup(void);
```

2.3 Java接口

目前java接口使用jni的方式实现。Java目录下tls_sigcheck.class，是由tls_sigcheck.java编译得到，如果有jdk兼容性问题，开发者可自行重新编译此文件，编译命令为：

```
javac -encoding utf-8 tls_sigcheck.java
```

请注意接口的包路径为com.tls.sigcheck，典型的使用方法是example目录下java版本demo的组织方式：

├── com
| └── tls
| └── sigcheck
| └── tls_sigcheck.class
├── Demo.class

├── Demo.java
├── ec_key.pem
├── public.pem
└── README

之前提到java接口目前使用的jni的方式，所以Demo.java调用了载入so的语句，开发者根据自己的存放jnisigcheck.so实际路径进行修改，在二进制包中预编译的jnisigcheck.so存放在lib/jni目录下。demo的使用方式请参考example/java/README。
下面是演示截图：

2.4 Java原生接口

Java原生接口依赖于5个jar包中。在tls_sig_api/java_native/lib目录下：

├── bcpkix-jdk15on-152.jar
├── bcprov-jdk15on-152.jar
├── commons-codec-1.10.jar
├── gson-2.3.1.jar
├── json.jar

└── tls_signature.jar

【特别注意】

从控制台界面下载
的公私钥，将公钥内容赋值给接口中的publicBase64Key参数，私钥内容赋值给接口中的privateBase64Key参数。

2.5 PHP接口

php实现的方式较为简单，就是调用命令行工具生成sig，工具是bin/signature.exe，php的调用方式如下：

```
function signature($identifier, $sdkappid, $private_key_path)
{
    #  这里需要写绝对路径，开发者根据自己的路径进行调整
    $command = '/home/signature'
    . ' ' . escapeshellarg($private_key_path)
    . ' ' . escapeshellarg($sdk_appid)
    . ' ' . escapeshellarg($identifier);
    $ret = exec($command, $out, $status);
    if ($status == -1)
    {
        return null;
    }
    return $out;
}
```

开发者请注意命令执行的路径和可执行权限，如果出现问题请尝试打印出 command 变量的内容进行定位。

2.6 PHP原生接口

在源码包和二进制包中都带有php/TLSSig.php文件，生成sig接口genSig和校验sig接口verifySig均在其中，注意PHP环境需要带openssl扩展，否则接口使用会报错，另外只支持PHP 5.3及以上的版本。

如果上述实现PHP环境无法满足要求，比如使用了红帽系（fedora、centos 和 rel
等）的操作系统，可以参考此处另一种与openssl和系统无关的实现。

3 windows平台

3.1 工具使用

注：这里讲解的是工具的使用说明，实际应用中需要开发者后台调用tls的后台api接口生成sig。

windows下生成sig和校验sig

首先不带参数执行tls_licence_tools.exe，即执行下面的命令：

```
tls_licence_tools.exe
```

输出：

```
current version: 201511190000
Usage:
    get sig: tls_licence_tools.exe gen pri_key_file sig_file sdkappid identifier
    get sig e.g.: tls_licence_tools.exe gen ec_key.pem sig 1400001052 xiaojun
    verify sig: tls_licence_tools.exe verify pub_key_file sig_file
sdkappid identifier
    verify sig e.g.: tls_licence_tools.exe verify public.pem sig
 1400001052 xiaojun
```

下面是演示截图：

```
D:\src\oicq64\tinyid\tls_sig_api\windows\64\tools>tls_licence_tools.exe
current version: 201511190000
Usage:
        get sig: tls_licence_tools.exe gen pri_key_file sig_file sdkappid identifier
        get sig e.g.: tls_licence_tools.exe gen ec_key.pem sig 1400001052 xiaojun
        verify sig: tls_licence_tools.exe verify pub_key_file sig_file sdkappid identifier
        verify sig e.g.: tls_licence_tools.exe verify public.pem sig 1400001052 xiaojun

D:\src\oicq64\tinyid\tls_sig_api\windows\64\tools>
```

输出实际上是参数模板和示例。

执行类似于下面的命令可以生成 sig：

```
tls_licence_tools.exe gen ec_key.pem sig 1400001052 xiaojun
```

对应的参数解释是：

```
tls_licence_tools gen 私钥文件路径 sig将要存放的路径 sdkappid 用户id
```

执行类似于下面的命令可以校验 sig：

```
tls_licence_tools.exe verify public.pem sig 1400001052 xiaojun
```

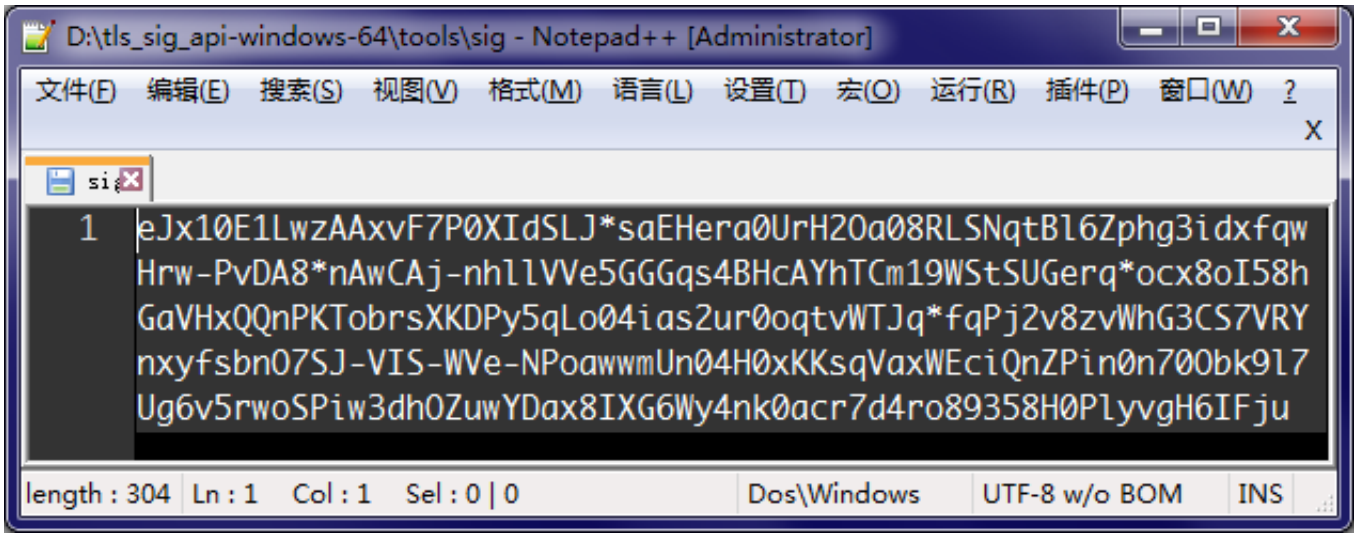
对应参数的解释是：

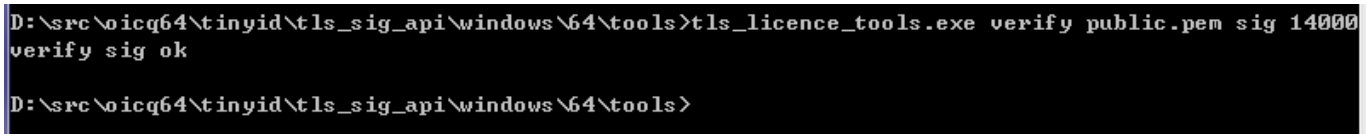
```
tls_licence_tools verify 公钥文件路径 sig的存放路径 sdkappid 用户id
```

下面演示截图：

```
D:\src\oicq64\tinyid\tls_sig_api\windows\64\tools>tls_licence_tools.exe gen ec_key.pem sig 14000010
generate sig ok

D:\src\oicq64\tinyid\tls_sig_api\windows\64\tools>
```

sig文件的内容如下图：





校验sig演示截图：

```
D:\src\oicq64\tinyid\tls_sig_api\windows\64\tools>tls_licence_tools.exe verify public.pem sig 14000
verify sig ok

D:\src\oicq64\tinyid\tls_sig_api\windows\64\tools>
```

下面解释下参数模板中参数的意义：

`gen`和`verify`分别表示生成`sig`和校验`sig`的命令

`pri_key_file`: 私钥文件的路径

`pub_key_file`: 公钥文件的路径

`sig_file`: `sig` 文件的路径，如果是生成 `sig`，那么会将 `sig` 写入这个文件，如果是校验 `sig`，那么会从这个文件读取 `sig` 的内容

`sdkappid`: 创建应用时页面上分配的 `sdkappid`

`identifier`: 用户标识，即用户 `id`

注意：生成的sig有效期为180天，开发者需要在sig过期前，重新生成sig。

3.2 C++接口

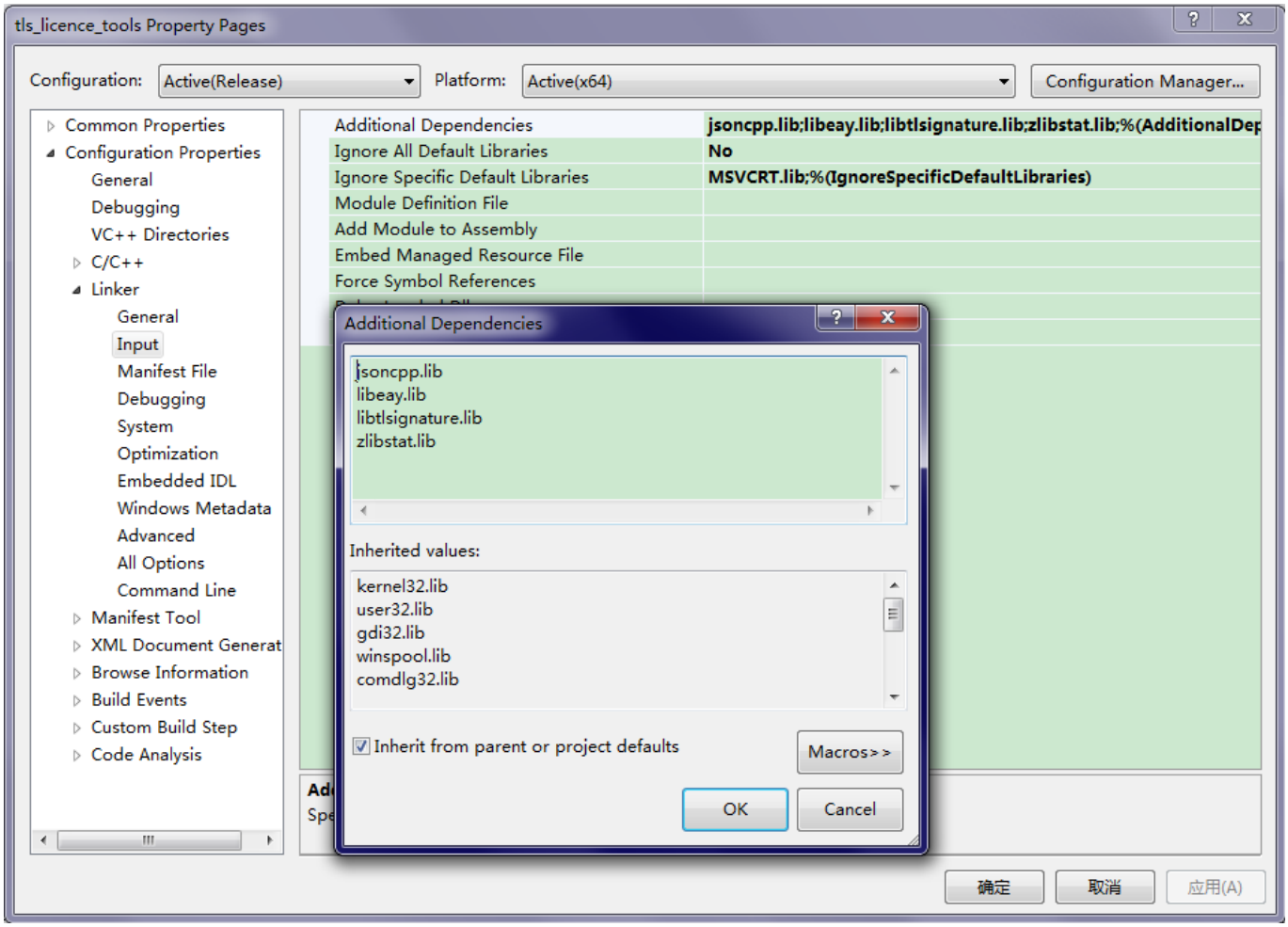
windows下C++接口的使用方式我们采用vs2012来举例。

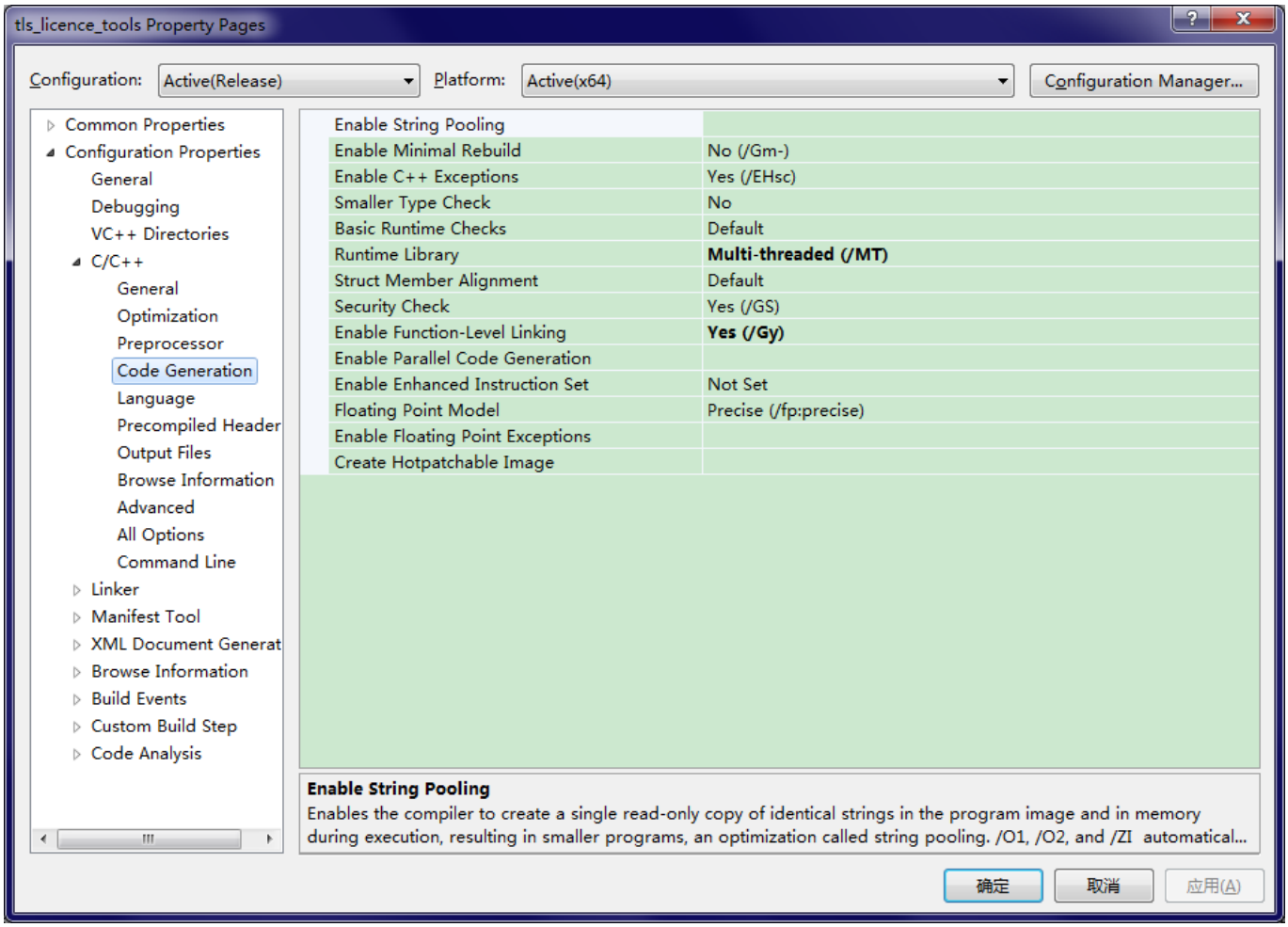
首先包含include\tls_sig_api目录下的tls_signature.h。头文件中包含的接口，tls_gen_signature_ex2和tls_check_signature_ex2，前者是生成sig的接口，后者是校验sig的接口，详细的参数和返回值说明请参考头文件tls_signature.h。

然后是链接静态库，在lib目录下有下列目录：

```
├── jni
├── jsoncpp
├── libsigcheck
├── openssl
├── tls_sig_api
└── zlib
```

需要链接的静态库是jsoncpp.lib、openssl目录下的libeay.lib、libtlsignature.lib和zlib目录下的zlibstat.lib，典型的编译配置如下：





【特别注意】

如果程序是多线程调用TLS后台API，请在程序初始化时和结束时分别调用下面的接口：

```
int multi_thread_setup(void);
void multi_thread_cleanup(void);
```

3.3 Java接口

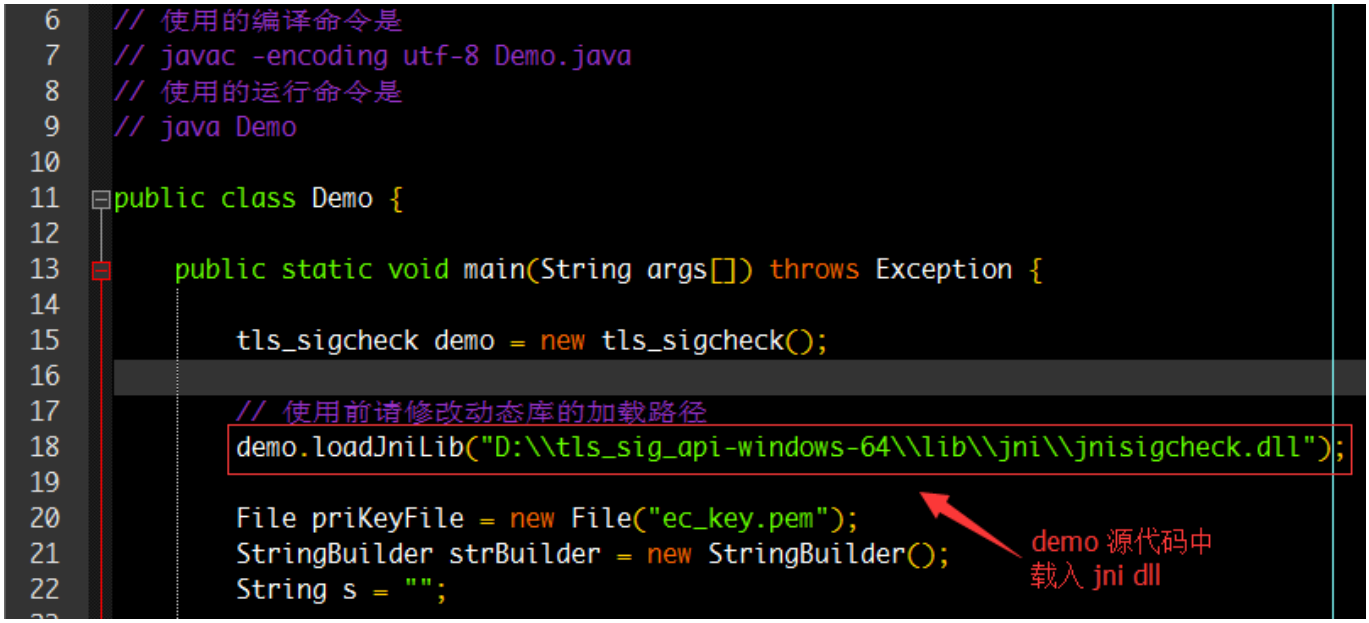
目前java接口使用jni的方式实现。Java目录下tls_sigcheck.class，是由tls_sigcheck.java编译得到，如果有jdk兼容性问题，开发者可自行重新编译此文件，编译命令为：

```
javac -encoding utf-8 tls_sigcheck.java
```

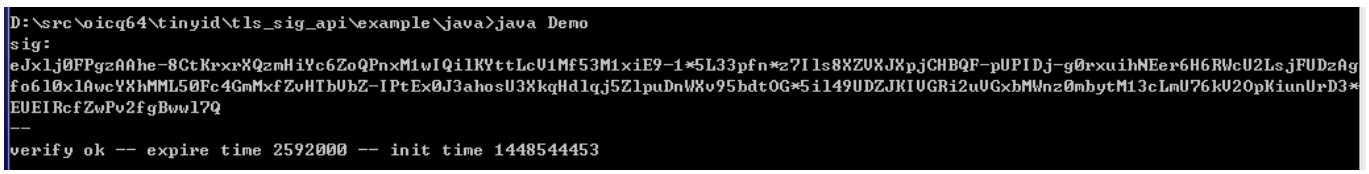
请注意接口的包路径为com.tls.sigcheck，典型的使用方法是example目录下java版本demo的组织方式：

```
├── com
│   └── tls
│       └── sigcheck
│           └── tls_sigcheck.class
├── Demo.class
├── Demo.java
├── ec_key.pem
├── public.pem
└── README
```

之前提到java接口使用的jni的方式，所以Demo.java调用了载入dll的语句，开发者根据自己的存放jnisigcheck.dll实际路径进行修改，预编译的jnisigcheck.dll存放在lib\jni目录下。demo的使用方式请参考example\java\README。下面是演示截图：



下面是运行结果，



3.4 Java原生接口

Java原生接口依赖于5个jar包。在tls_sig_api/java_native/lib目录下：

├── bcpkix-jdk15on-152.jar
├── bcprov-jdk15on-152.jar
├── commons-codec-1.10.jar
├── gson-2.3.1.jar
├── json.jar
└── tls_signature.jar

【特别注意】

从控制台界面下载
的公私钥，将公钥内容赋值给接口中的publicBase64Key参数，私钥内容赋值给接口中的privateBase64Key参数。

3.5 C#接口

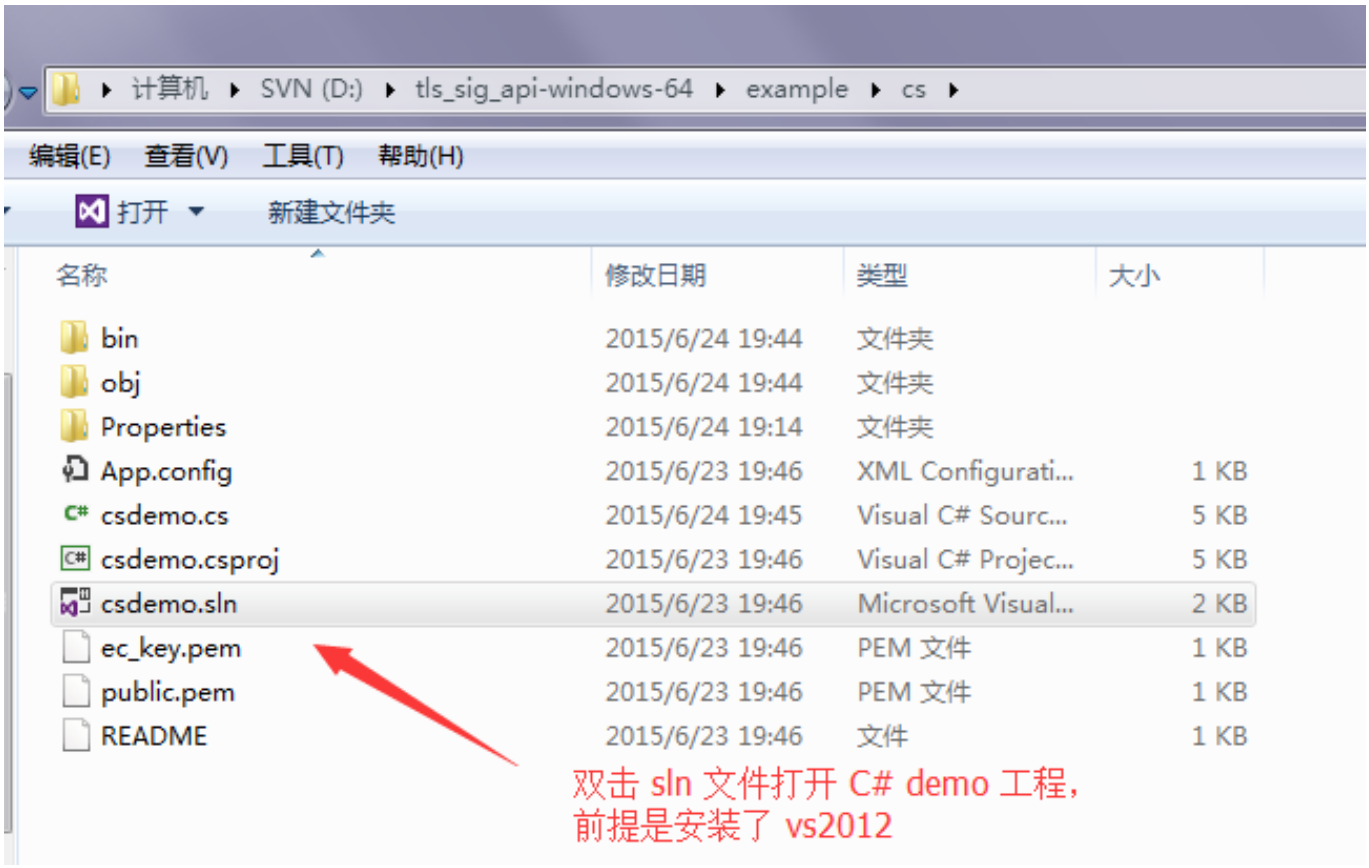
以非托管的方式调用dll实现，调用的dll为lib\libsigcheck\sigcheck.dll，C样式接口的参数与返回值说明参见include\sigcheck.h头文件，接口的转换方式如下：

```
class sigcheck
{
    [DllImport(dllpath.DllPath, Ent
ryPoint = "tls_gen_sig_ex"
, CharSet = CharSet.Ansi, CallingConvention = CallingConvention.Cdecl)]
    public extern static int tls_gen_sig_ex(
        UInt32 sdkappid,
        string identifier,
        StringBuilder sig,
        UInt32 sig_buff_len,
        string pri_key,
        UInt32 pri_key_len,
```

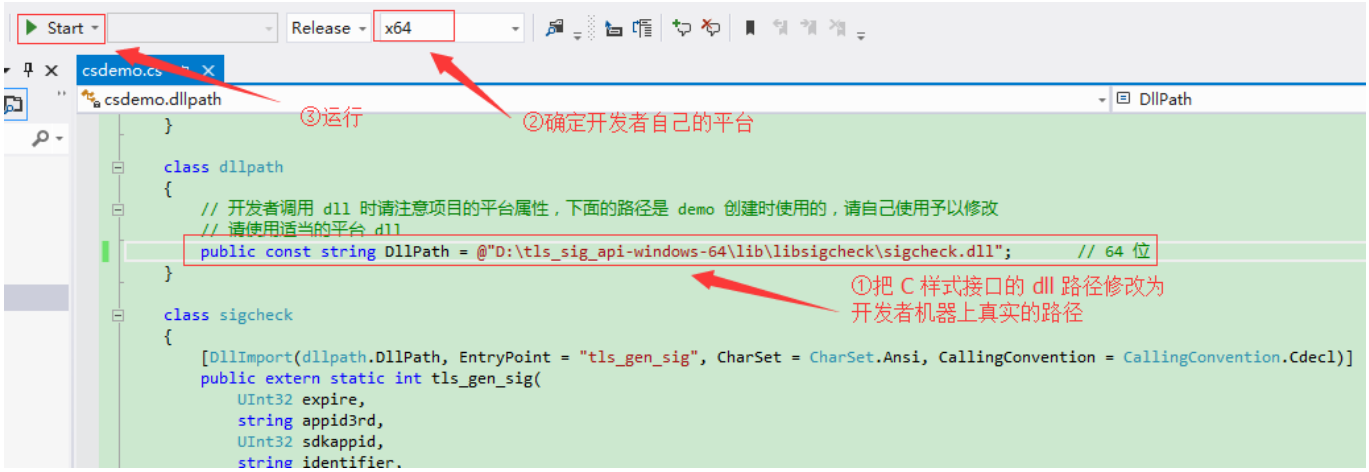
```
        StringBuilder err_msg,
        UInt32 err_msg_buff_len
    );

    [DllImport(dllpath.DllPath, Ent
ryPoint = "tls_vri_sig_ex"
, CharSet = CharSet.Ansi, CallingConvention = CallingConvention.Cdecl)]
    public extern static int tls_vri_sig_ex(
        string sig,
        string pub_key,
        UInt32 pub_key_len,
        UInt32 sdkappid,
        string identifier,
        ref UInt32 expire_time,
        ref UInt32 init_time,
        StringBuilder err_msg,
        UInt32 err_msg_buff_len
    );
}
```

其中dllpath.DllPath指明了dll的路径，详细请参见example\cs\csdemo.cs。

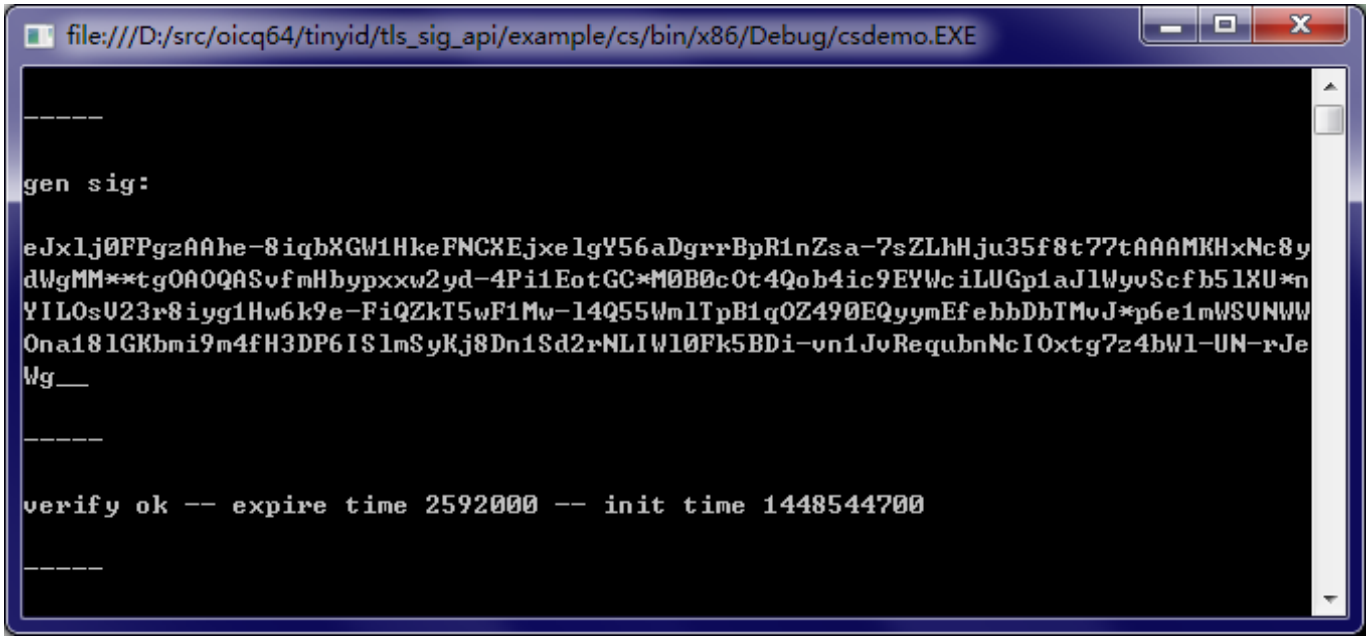
关于demo的使用方法参见example\cs\README。下面是演示截图：


计算机 ▸ SVN (D:) ▸ tls_sig_api-windows-64 ▸ example ▸ cs ▸
编辑(E) 查看(V) 工具(T) 帮助(H)
打开 新建文件夹
名称 修改日期 类型 大小
bin 2015/6/24 19:44 文件夹
obj 2015/6/24 19:44 文件夹
Properties 2015/6/24 19:14 文件夹
App.config 2015/6/23 19:46 XML Configurati... 1 KB
csdemo.cs 2015/6/24 19:45 Visual C# Sourc... 5 KB
csdemo.csproj 2015/6/23 19:46 Visual C# Projec... 5 KB
csdemo.sln 2015/6/23 19:46 Microsoft Visual... 2 KB
ec_key.pem 2015/6/23 19:46 PEM 文件 1 KB
public.pem 2015/6/23 19:46 PEM 文件 1 KB
README 2015/6/23 19:46 文件 1 KB
双击 sln 文件打开 C# demo 工程，
前提是安装了 vs2012


using System.Runtime.InteropServices;
// 此文件为 C# demo，使用的非托管模式的 C++ 接口
// 项目中使用 C# 调用 dllsigcheck.dll 时请注意平台，64 位和 32 位平台的 dll 我们均有预编译
namespace csdemo
{
class Program
{
// 下面的公私钥路径都是绝对路径，请开发者自行修改
const string pri_key_path = @"D:\tls_sig_api-windows-64\example\cs\ec_key.pem";
const string pub_key_path = @"D:\tls_sig_api-windows-64\example\cs\public.pem";
static void Main(string[] args)
{
// 生成 sig 文件
FileStream f = new FileStream(pri_key_path, FileMode.Open, FileAccess.Read);
BinaryReader reader = new BinaryReader(f);
byte[] b = new byte[f.Length];
reader.Read(b, 0, b.Length);
string pri_key = Encoding.Default.GetString(b);
修改公私钥路径



下面是运行结果：



注意：如果选择Any CPU平台，请默认加载32位dll。

3.6 PHP接口

php实现的方式较为简单，就是调用命令行工具生成sig，工具是bin\signature.exe，php的调用方式如下：

```
function signature($identifier, $sdkappid, $private_key_path)
{
```

```
    # 这里需要写绝对路径，开发者根据自己的路径进行调整
    $command = 'D:\\signature.exe'
    . ' ' . escapeshellarg($private_key_path)
    . ' ' . escapeshellarg($sdk_appid)
    . ' ' . escapeshellarg($identifier);
    $ret = exec($command, $out, $status);
    if ($status == -1)
    {
        return null;
    }
    return $out;
}
```

开发者请注意命令执行的路径，如果出现问题请尝试打印出 command 变量的内容进行定位。

3.7 PHP原生接口

在源码包和二进制包中都带有php/TLSSig.php文件，生成sig接口genSig和校验sig接口verifySig均在其中，注意PHP环境需要带openssl扩展，否则接口使用会报错，另外只支持PHP 5.3及以上的版本。

如果上述实现PHP环境无法满足要求，比如使用了红帽系（fedora、centos 和 rel
等）的操作系统，可以参考此处另一种与openssl和系统无关的实现。

4 其他平台接口

- javascript http://bbs.qcloud.com/thread-17311-1-1.html
- python http://bbs.qcloud.com/thread-14366-1-1.html
- golang http://bbs.qcloud.com/thread-21826-1-1.html

5 TLS 后台 API 下载

点击这里下载。

6 联系我们

这里的一些信息可能对您有帮助，如需支持，请@TLS帐号支持，QQ 3268519604，电子邮箱 tls_assistant@tencent.com。

更多