云通信

帐号登录集成

产品文档

腾讯云

文档目录

帐号登录集成
帐号登录集成说明

最近更新时间：2018-11-16 11:47:34

帐号集成用于开发者和云通信之间通过签名验证建立信任关系， 本文后续将重点讲解如何进行帐号集成。另外我们针对这里提到的内容录制了视频，可以打开腾讯云学院 云通信帐号集成示例 进行观看。

独立模式是指用户注册和身份验证由开发者负责，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，下面详细讲解签名验证流程。

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。

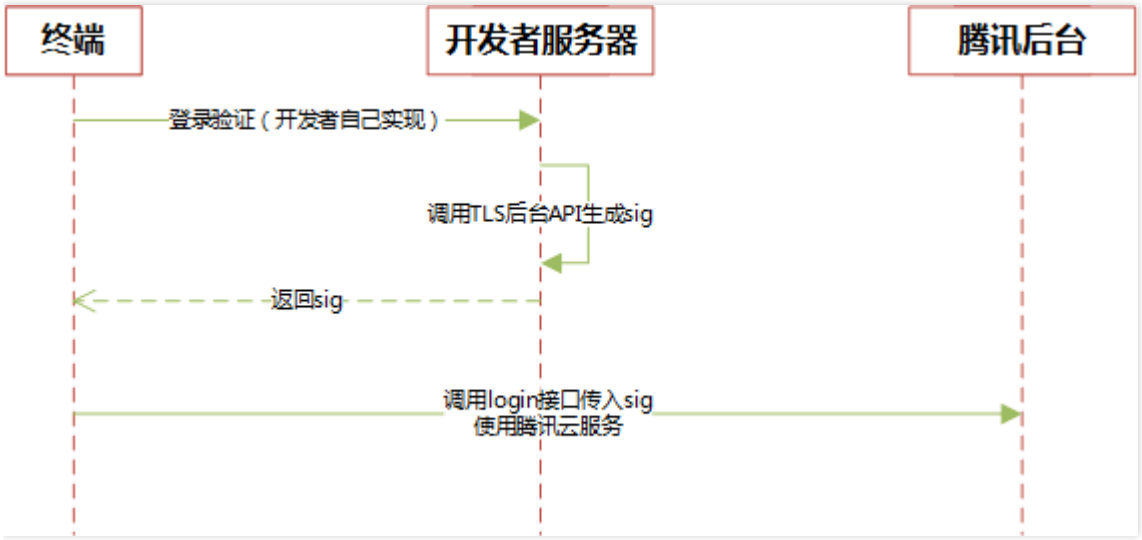
登录说明

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签名 UserSig ，调用云通信 SDK 的接口进行验证，验证成功后即可使用云通信服务。

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 TLS后台API使用手册。

典型流程



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用户ID `Identifier` 和 签名 `UserSig` 。

附录

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。

- 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。

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供开发者调试使用，详情请参见 TLS 后台 API 开发指引。

> 注：关于非对称算法，可通过这篇 公开密钥加密 进一步了解。

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，其区别如下。

- 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。
- 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 设置 App 管理员 文档。

TLS后台API使用手册

最近更新时间：2018-11-07 16:21:29

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。

> **注意：**
> 在控制台上下载的公私钥文件名分别为 private_key 和 public_key ，分别对应下面的 ec_key.pem 和 public.pem 。请在使用公私钥时注意区分。

Linux 平台

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

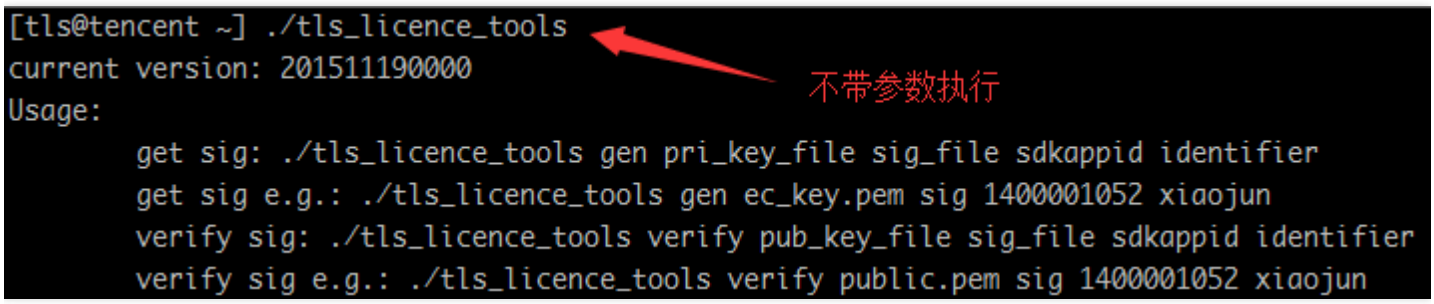
首先不带参数执行 tls_licence_tools ，即执行下面的命令：

```
$ ./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 identifier
verify sig e.g.: ./tls_licence_tools verify public.pem sig 1400001052 xiaojun
```

输出实际上是参数模板和示例。下面是演示截图：



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**

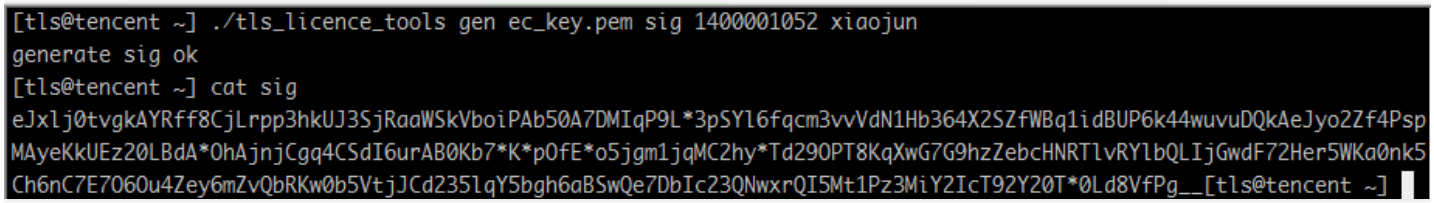
执行类似于下面的命令可以校验 sig：

> ./tls_licence_tools verify **public**.pem sig 1400001052 xiaojun

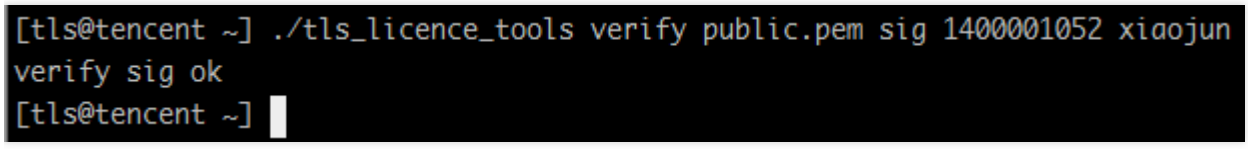
对应参数的解释是：

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

以下为生成 sig 演示：



以下为校验 sig演示：



以下为参数模板中各个参数的意义：

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。

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/tls_sig_api -Wall -fPIC -L../../linux/64/lib/jsoncpp ../../openssl-dynamic/lib/libcrypto.a -ljsoncpp -ldl -lz
```

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);
```

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：

> 注：如果在 Java 代码中使用了**多线程**的方式生成 usersig ，可以参阅 腾讯云论坛 相关介绍。

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
```

> **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。

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` ，PHP 的调用方式如下：

> 注：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。

```
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($sdkappid)
. ' ' . escapeshellarg($identifier);
$ret = exec($command, $out, $status);
if ($status == -1)
{
return null;
}
return $out;
}
```

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。

> 注：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，可以参考腾讯论坛 中另一种与 openssl 和系统无关的实现。

Windows 平台

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

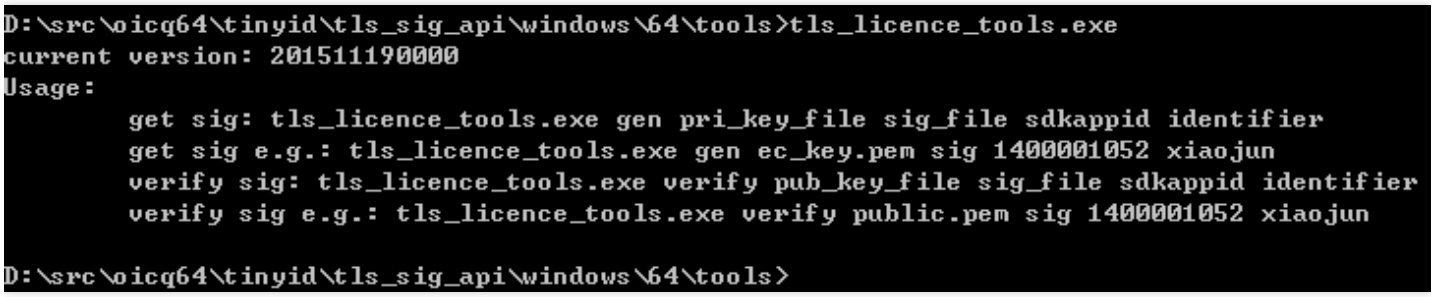
首先不带参数执行 `tls_licence_tools.exe` ，即执行下面的命令：

```
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
```

输出实际上是参数模板和示例。下面是演示截图：



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
```

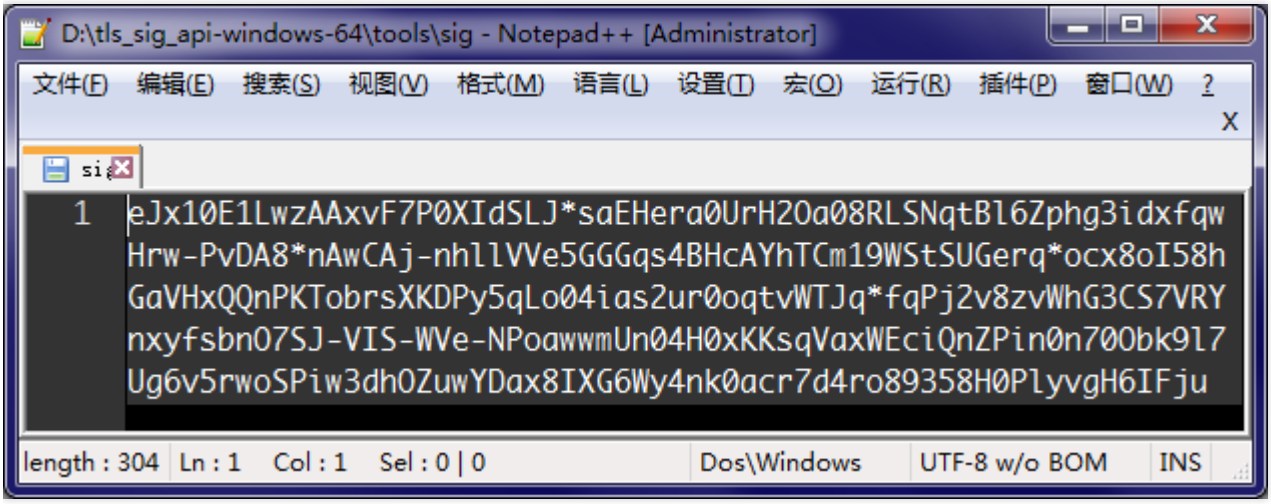
对应参数的解释是：

```
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：

> **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。

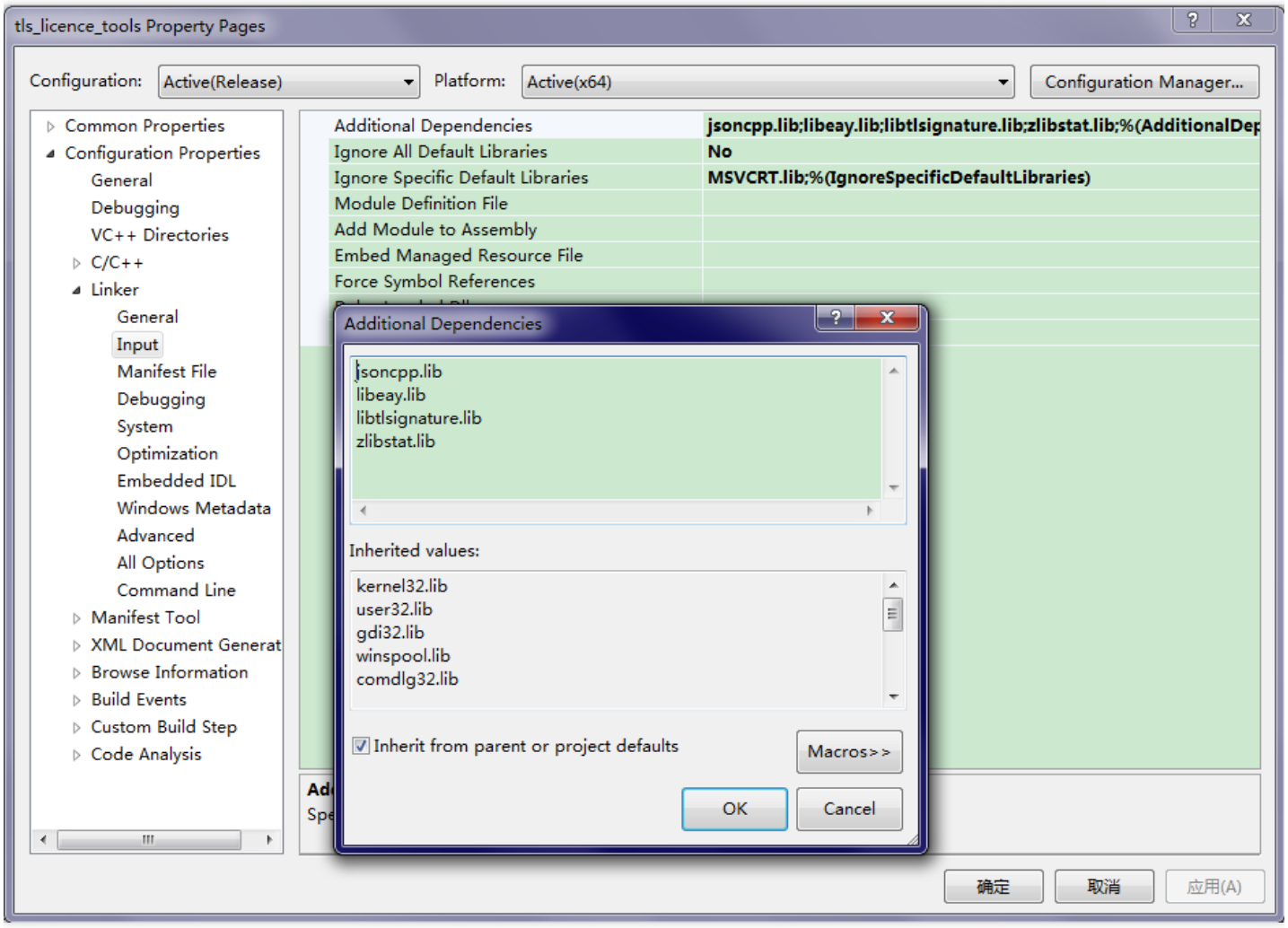
gen 和 verify:分别表示生成 **sig** 和校验 **sig** 的命令
pri_key_file：私钥文件的路径
pub_key_file：公钥文件的路径
sig_file：**sig** 文件的路径，如果是生成 **sig**，那么会将 **sig** 写入这个文件，如果是校验 **sig**，那么会从这个文件读取 **sig** 的内容
sdkappid：创建应用时页面上分配的 sdkappid
identifier：用户标识，即用户 id

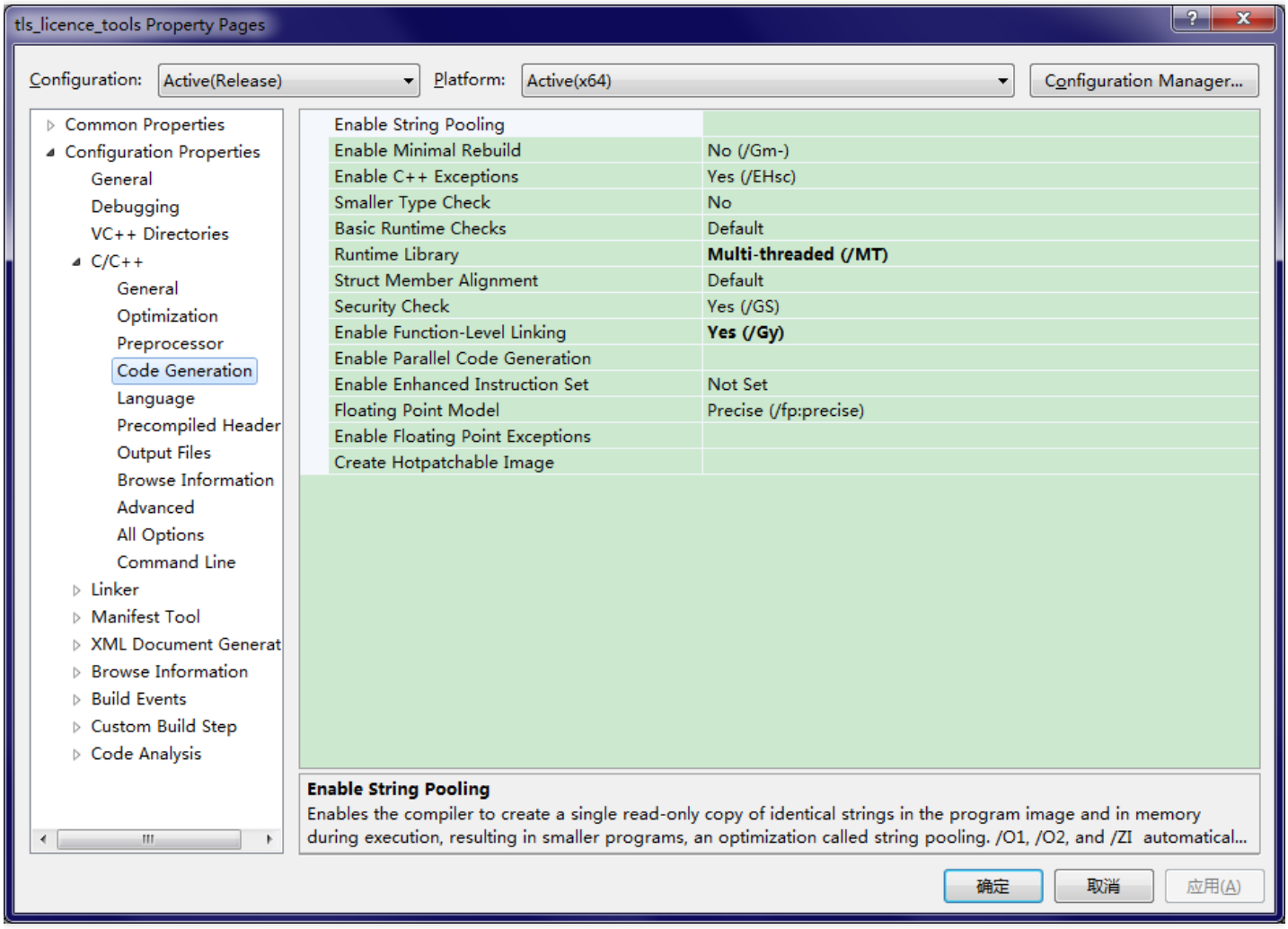
C++ 接口

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：


tls_licence_tools Property Pages
Configuration: Active(Release)
Platform: Active(x64)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
Advanced
All Options
Command Line
Manifest Tool
XML Document Generat
Browse Information
Build Events
Custom Build Step
Code Analysis
Additional Dependencies
jsoncpp.lib;libeay.lib;libtlsignature.lib;zlibstat.lib;%(AdditionalDep
Ignore All Default Libraries
No
Ignore Specific Default Libraries
MSVCRT.lib;%(IgnoreSpecificDefaultLibraries)
Module Definition File
Add Module to Assembly
Embed Managed Resource File
Force Symbol References
Additional Dependencies
jsoncpp.lib
libeay.lib
libtlsignature.lib
zlibstat.lib
Inherited values:
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
comdlg32.lib
Inherit from parent or project defaults
Macros>>
OK
Cancel
确定
取消
应用(A)



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);
```

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：

```
6  // 使用的编译命令是
7  // javac -encoding utf-8 Demo.java
8  // 使用的运行命令是
9  // java Demo
10
11 public class Demo {
12
13     public static void main(String args[]) throws Exception {
14
15         tls_sigcheck demo = new tls_sigcheck();
16
17         // 使用前请修改动态库的加载路径
18         demo.loadJniLib("D:\\tls_sig_api-windows-64\\lib\\jni\\jnisigcheck.dll");
19
20         File priKeyFile = new File("ec_key.pem");
21         StringBuilder strBuilder = new StringBuilder();
22         String s = "";
```

demo 源代码中载入 jni dll

下面是运行结果：

```
D:\src\oicq64\tinyid\tls_sig_api\example\java>java Demo
sig:
eJxlj0FPgzAAhe-8CtKrxrXQzmHiYc6ZoQPnxM1wIQilKYttLcV1Mf53M1xiE9-1*5L33pfn*z7Ils8XZVXJXpjCHBQF-pUPIDj-g0rxuihNEer6H6RWcU2LsjFUDzAg
fo610xlAwcYXhMML50Fc4GmMxfZvHTbVbZ-IPtEx0J3ahosU3XkqHdlqj5Z1puDnWXv95bdtOG*5il49UDZJKIVGRi2uVGxbMWnz0mbytM13cLmU76kV2OpKiunUrD3*
EUEIRcfZwPv2fgBww17Q
--
verify ok -- expire time 2592000 -- init time 1448544453
```

注意：

如果在 Java 代码中使用了多线程的方式生成 `usersig` ，可以参考腾讯云论坛 中的相关介绍。

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
```

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。

C# 接口

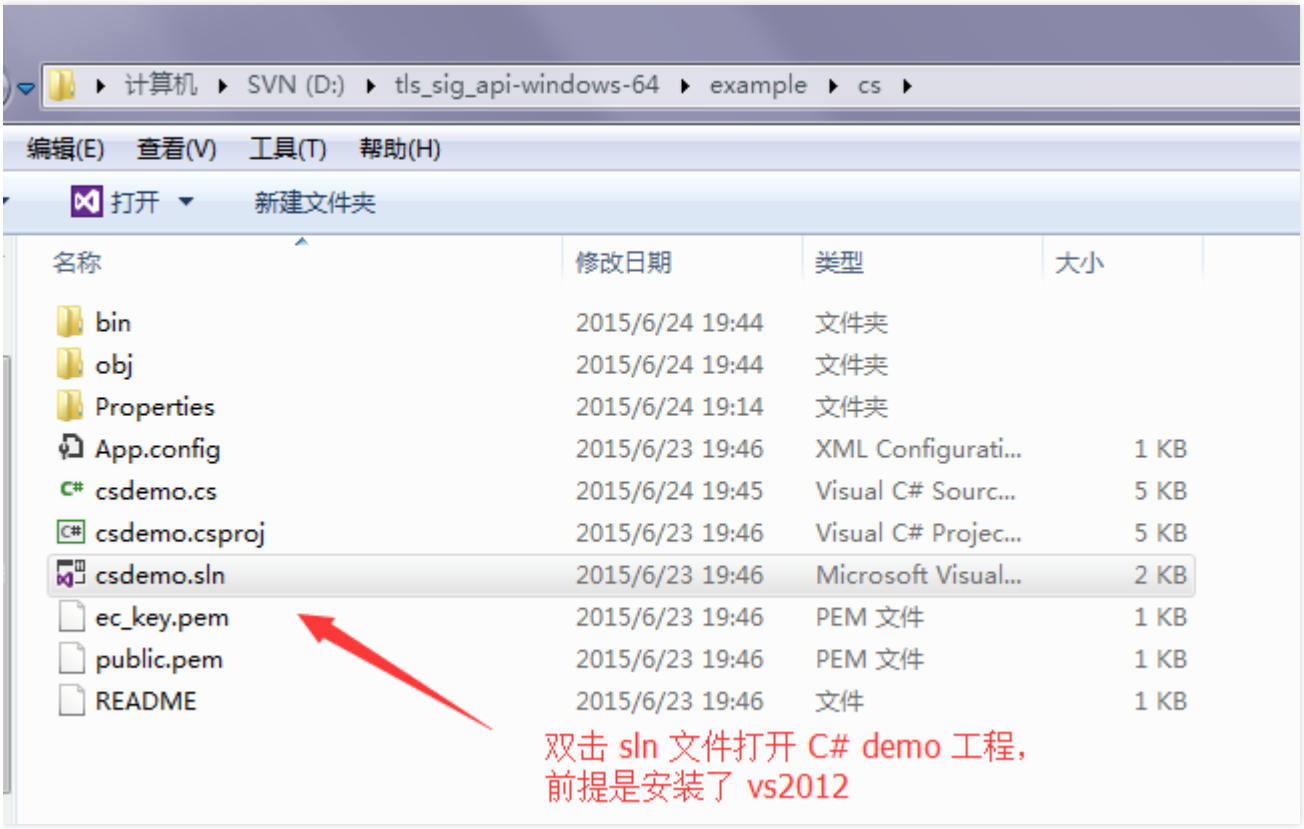
以非托管的方式调用 dll 实现，调用的 dll 为 `lib\libsigcheck\sigcheck.dll` ，C 样式接口的参数与返回值说明参见 `include\sigcheck.h` 头文件，接口的转换方式如下：

```
class sigcheck
{
[DllImport(dllpath.DllPath, EntryPoint = "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, EntryPoint = "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
    );
    }
```

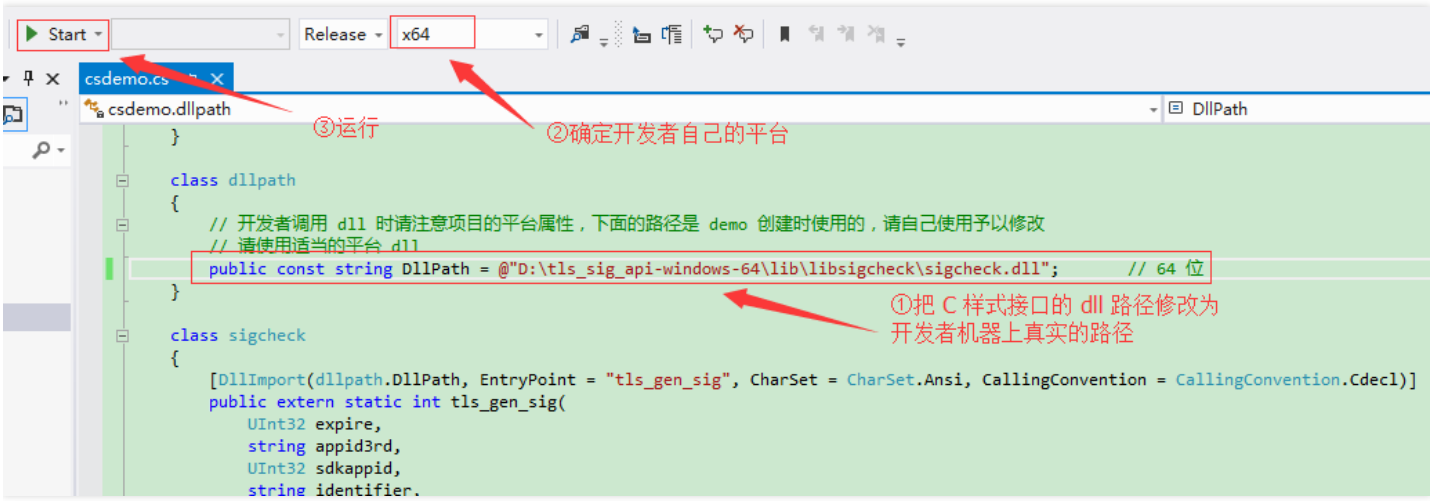
其中 `dllpath.DllPath` 指明了 dll 的路径，详细请参见 `example\cs\csdemo.cs` 。关于 Demo 的使用方法参见 `example\cs\README` 。下面是演示截图：

> **注意：**
> 如果选择 Any CPU 平台，请默认加载 32 位 dll。

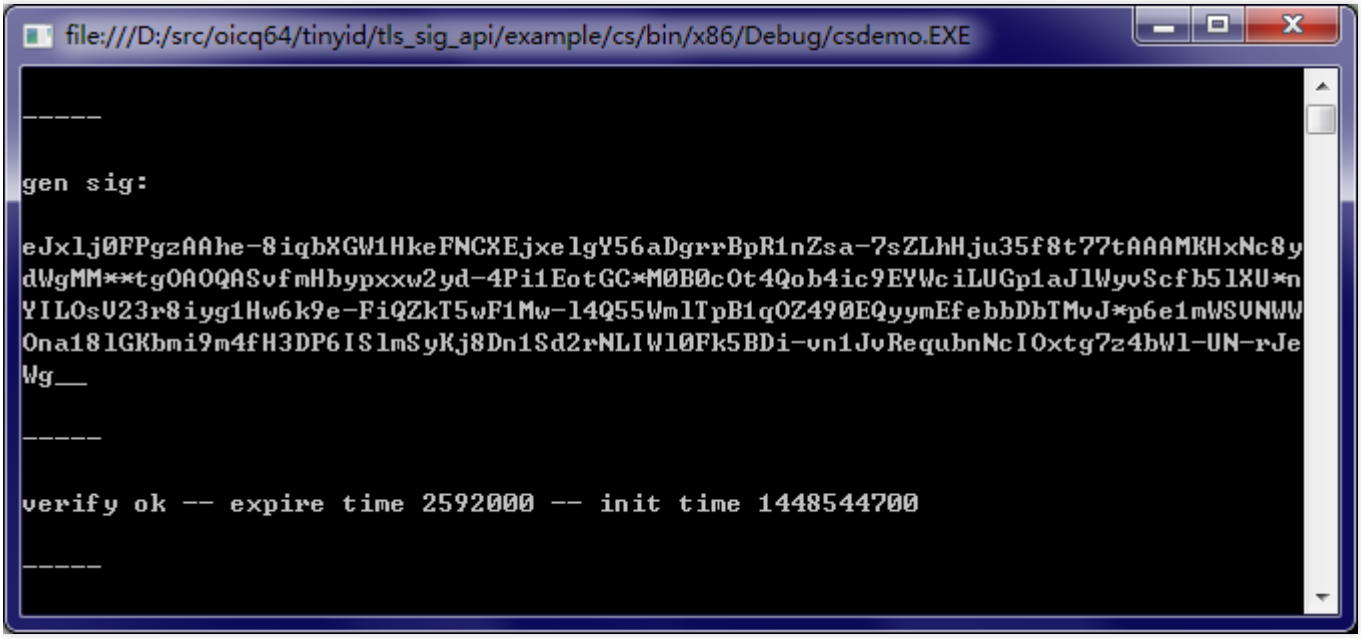



```
using System.Runtime.InteropServices;

// 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：



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：

> 注：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。

```
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($sdkappid)
. ' ' . escapeshellarg($identifier);
$ret = exec($command, $out, $status);
if ($status == -1)
{
return null;
}
return $out;
}
```

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。

> 注：若上述实现 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 腾讯云论坛 另一种与 openssl 和系统无关的实现。

其他平台接口

- JavaScript
- Python
- Go
- Java
- PHP

TLS 后台 API 下载

下载 TLS 后台 API 。

联系我们

腾讯云论坛 的一些信息可能对您有帮助，如需更多帮助，请提交工单。