**SSLVPN客户端SDK使用手册**

**Android**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 时间 | 作者 | 备注 |
| 2.0.0 | 2015.12.01 | 刘建平 | 初始化 |
| 2.5.0 | 2017.06.22 | 刘建平 | 支持会话状态广播、国密、两步认证  重新整理文档结构 |
| 2.5.1 | 2017.07.31 | 刘建平 | 添加第二步360ID安全口令认证支持 |

# SSLVPN SDK概述

SSLVPN有两种工作模式

* 代理连接模式，基于连接工作。这种工作模式只支持TCP连接，每个SSLVPN客户端与服务器间的SSL连接对应一个APP的TCP连接，并且这个SSL连接与APP的TCP连接生命周期保持一致。
* NC网络模式，基于网络工作。工作在TCP/IP协议栈第三层，这种模式下，SSLVPN通过虚拟网卡的方式，将APP终端与APP服务器连接成一个网络，可以支持所有的IP层协议（TCP、UDP、ICMP等）且还能支持APP服务器反连APP终端，配置相应的路有表按需将APP终端的流量导入此虚拟网卡。SSLVPN客户端与服务器之间是一个SSL长连接（可以认为是一根虚拟的网线），整个APP终端上所有APP客户端与APP服务器间的IP包都经过这个SSL连接进行通信。

*注：由于移动终端特性，整个系统中只能一个虚拟网卡工作，因此如果有多个APP需要经过VPN时，可只一个APP集成SDK拨号建立NC隧道，其他APP无需集成SDK直接使用此NC隧道即可。*

SSLVPN客户端SDK旨在APP客户端与服务器之间建立透明化的安全隧道，不影响应用开发的任何逻辑。



* APP-1集成SDK且启用了NC虚拟网卡，APP-1可通过自身SDK中的代理模式访问APP服务器-1，也可通过NC网络模式访问APP服务器-1
* APP-2集成SDK不启用NC虚拟网卡，APP-2可通过自身SDK中的代理模式访问APP服务器-2，也可通过APP-1的虚拟网卡访问APP服务器-2
* APP-3没集成SDK，可通过APP-1的虚拟网卡访问APP服务器-3

SSLVPN客户端SDK配合SSLVPN服务器提供以下几种安全隧道模式

* **代理模式**
* Hook模式，在这种模式下，SSLVPN客户端SDK底层会自动Hook APP的所有网络连接，APP无需任何额外的开发工作量。但是，因为整个Android体系的碎片化缘故，在部分环境下Hook不能工作（可通过SPVPNClient.isHookWorking()检查Hook是否可用，目前只有极少数情况下Hook不成功）。
  + 优点：无任何额外开发工作量
  + 缺点：有适配性问题，在极少数系统下可能会无法工作
* Proxy模式，SSLVPN客户端SDK提供标准的代理服务器，APP发出的连接通过SDK的代理服务器到达服务器
  + HTTP-Proxy服务器

遵循标准<https://tools.ietf.org/html/rfc2068>，支持任何TCP连接（HTTP/HTTPS/其他Socket）的代理。

* + SOCKS代理服务器

支持无认证的SOCKS4/SOCKS4a/SOCKS5，目前仅支持TCP连接代理，暂时不支持UDP和BIND代理。

SOCKS5 <https://tools.ietf.org/html/rfc1928>

* + 支持PAC自动代理

PAC地址为<http://127.0.0.1:port/get_pac>

* + 优点：使用标准协议，无任何适配性问题
  + 缺点：相比Hook模式，有一点点的额外开发工作量（请参考后面指导代码）
* Mapping模式，这种模式下，SDK将应用服务器地址和端口，映射到一个本地环回地址127.0.0.1的端口上，例如应用服务器的访问地址是<http://192.168.88.88:80/>，经过映射后，可以这样访问该服务器的地址就变为<http://127.0.0.1:1234/>（这里假设映射的端口是1234）。
  + 优点：无法Hook且使用了第三方库并且无法控制其网络连接时（这个库不支持设置代理服务器），可使用这种模式
  + 缺点：很多跟主机名有关的应用协议尤其是HTTP协议（例如WebView会自动将URL中的主机名作为HTTP请求头的Host字段，而Web服务器上有虚拟主机判断时可能就不能正常工作）等，可能工作不正常，有一点点的额外开发工作量
* **NC网络模式**

这种模式下，SDK会在系统内建立一个虚拟网卡，并配置上由VPN服务器分配的IP地址，并指定内网APP服务器的IP路由到此虚拟网卡，因此所有到这些受保护IP的连接都会进入该虚拟网卡，从而进入SDK转发层，SDK再通过SSL将这些连接经过SSLVPN服务器与内网APP服务器连通。由于虚拟网卡是系统层的功能，因此NC网络模式可以让整个移动终端内的所有APP都能访问到内网APP服务器。NC网络模式需要VPN服务器上添加相应的NC网络类型服务和NC网络策略配置。

我们推荐APP开发者使用Proxy模式，其次是Hook模式，万不得已时才使用Mapping模式。如果需要系统内其他APP也能访问内网APP服务器则可使用NC网络模式。

# 快速集成

## 集成库文件

集成SDK必须的库文件有两个，一个是java层的jar文件**libsecure\_portal.jar**，另一个是native层的so文件**libsecure\_portal.so**。因为SDK本身需要Android支持库v4，因此还需要java层的android-support-v4.jar，可以从Google官方下载最新的android-support-v4.jar也可以直接使用演示例子中的此文件。

集成native库文件libsecure\_portal.so时，需注意，如果你的APP自身已经有native so库文件，且你的so文件没有支持全平台，例如只支持了armeabi，则你只需要集成SDK中的libs/armeabi/即可。如果你的so支持armeabi和arm64-v8a，则需集成SDK中的libs/armeabi和libs/arm64-v8a。以此类推按需集成。如果你的APP没有native so库文件，则可集成SDK的整个libs目录，这样SDK能支持的平台你的APP就都能支持。

目前，SDK提供的库目录详情如下表（暂时不支持x86\_64、mips和mips64平台）

|  |
| --- |
| **libs/**libsecure\_portal.jar  **libs/armeabi/**libsecure\_portal.so  **libs/armeabi-v7a/**libsecure\_portal.so  **libs/x86/**libsecure\_portal.so  **libs/arm64-v8a/**libsecure\_portal.so |

## 修改APP权限

修改Androidmanifest.xml

|  |
| --- |
| <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"/>  <uses-permission android:name="android.permission.INTERNET"/>  <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>  <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"/>  <uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"/>  <uses-permission android:name="android.permission.READ\_PHONE\_STATE"/>  <!-- 如果需要阻止上网策略，则需要有杀其他进程的权限 -->  <uses-permission android:name="android.permission.KILL\_BACKGROUND\_PROCESSES" /> |

## 添加Service

修改Androidmanifest.xml

|  |
| --- |
| <!-- SDK 后台服务Service -->  <service android:exported="false"  android:name="com.secure.sportal.sdk.app.SPTunnelService"/>  <service android:exported="false"  android:name="com.secure.sportal.service.SPIntentService"/>  <!-- 如果需要支持NC网络模式则需要添加VPNService -->  <service  android:name="com.secure.sportal.sdk.vpn.SPNCService"  android:permission="android.permission.BIND\_VPN\_SERVICE">  <intent-filter>  <action android:name="android.net.VpnService" />  </intent-filter>  </service> |

## 拨号登录SSLVPN

拨号登录SSLVPN的步骤如下

* 引用SDK类

|  |
| --- |
| import com.secure.sportal.sdk.SPVPNClient; |

* 准备连接和认证参数

|  |
| --- |
| Properties param = new Properties();  // SSLVPN 服务器地址，支持主机名和IP地址  param.setProperty(SPVPNClient.PARAM\_VPN\_HOST, "172.16.1.233");  // SSLVPN 服务器端口号  param.setProperty(SPVPNClient.PARAM\_VPN\_PORT, "443");  // SSLVPN 认证服务器名称（如果不填，则SDK自动使用第一个作为默认值）  param.setProperty(SPVPNClient.PARAM\_AUTH\_SERVER, "本地认证");  // SSLVPN 登录用户名  param.setProperty(SPVPNClient.PARAM\_AUTH\_USERNAME, "test");  // SSLVPN 登录密码  param.setProperty(SPVPNClient.PARAM\_AUTH\_PASSWORD, "123456"); |

* 拨号登录

|  |
| --- |
| // 可直接在UI主线程调用login()方法，SDK内部自动在非UI线程完成工作，  // 并通过异步回调方式通知执行结果。login()会立即返回，不会阻塞。  SPVPNClient.login(VPNClientDemoActivity.this, param,  new SPVPNClient.OnVPNLoginCallback(){  …  }); |

* 通过回调获取结果

|  |
| --- |
| new SPVPNClient.OnVPNLoginCallback()  {  @Override  public void onVPNLoginMessage(int msgid, String msg)  {  if ( msgid==SPVPNClient.MSGID\_LOGIN\_SUCC )  {  // SSLVPN 登录成功  }  else if ( msgid==SPVPNClient.MSGID\_LOGIN\_FAIL )  {  // SSLVPN 登录失败  }  }  } |

注：SDK会确保onVPNLoginMessage()回调函数在APP主线程执行。

* 接收SDK中的会话状态广播通知

|  |
| --- |
| private class SessionStatusReceiver extends BroadcastReceiver  {  @Override  public void onReceive(Context context, Intent intent)  {  String topic = intent.getStringExtra("topic");  String content = intent.getStringExtra("content");  Log.d("demo", "SSLVPN-Session-State topic: " + topic);  Log.d("demo", "SSLVPN-Session-State content: " + content);  if ("session\_status".equals(topic))  {  JSONObject json = SPJSONUtil.parseObject(content);  if ("offline".equals(json.optString("session\_state")))  {  // "提示", "SSLVPN下线了  }  }  }  }  …  // 注册接收器  SPVPNClient.resitSessionStateReceiver(this, sessionStatusReceiver);  …  // 反注册接收器  SPVPNClient.unregistBroadcastReceiver(this, sessionStatusReceiver); |

**注：随着Android 7.0开始Broadcast广播消息的发送和接收增加了限制，SSLVPN SDK从2.4.2版本开始，全部使用android.support.v4.content.LocalBroadcastManager实现广播，即所有广播消息仅限APP内部可收发。SDK内提供com.secure.comm.utils.SPIntentUtil类实现了对android.support.v4.content.LocalBroadcastManager的简单封装。**

# APP开发演示和用例

## HTTP连接演示

### Hook模式

|  |
| --- |
| // SDK底层自动Hook连接  HttpURLConnection conn = (HttpURLConnection) new URL("http://192.168.88.88/")  .openConnection(); |

### Proxy模式

|  |
| --- |
| // 通过SDK提供的代理服务器  HttpURLConnection conn = (HttpURLConnection) new URL("http://192.168.88.88/")  .openConnection(SPVPNClient.getHttpProxy()); |

### 适配模式

|  |
| --- |
| if (SPVPNClient.isNetHooking())  {  // 如果SDK Hook成功  HttpURLConnection conn = (HttpURLConnection) new URL("http://192.168.88.88/")  .openConnection();  }  else  {  // 如果SDK Hook失败，则使用代理服务器  HttpURLConnection conn = (HttpURLConnection) new URL("http://192.168.88.88/")  .openConnection(SPVPNClient.getHttpProxy());  } |

## TCP连接演示

### API模式

|  |
| --- |
| // 通过SPVPNClient提供的API创建一个TCP连接  Socket sock = SPVPNClient.sconnect("192.168.88.88", 80, 30000); |

### Hook模式

|  |
| --- |
| // 直接连接，SDK底层自动Hook此连接  Socket sock = new Socket();  sock.connect(new InetSocketAddress(host, port), 30000); |

### SOCKS Proxy模式

|  |
| --- |
| // 通过SOCKS代理服务器创建一个TCP连接  java.net.Proxy proxy = new Proxy(java.net.Proxy.Type.SOCKS,  new InetSocketAddress("127.0.0.1", SPVPNClient.getProxyPort()));  Socket sock = new Socket(proxy);  sock.connect(new InetSocketAddress("192.168.88.88", 80), 30000); |

### HTTP Proxy模式

|  |
| --- |
| // Android的Socket只支持SOCKS代理服务器，因此HTTP代理服务器只能自己实现  Socket sock = new Socket();  sock.connect(SPVPNClient.getHttpProxy().address(), 30000);  // 发送代理连接请求  String request = **"CONNECT 192.168.88.88:80 HTTP/1.1\r\n\r\n";**  sock.getOutputStream().write(request.getBytes());  // HTTP/1.1 200 OK\r\n\r\n  byte[] bytes = new byte[128];  InputStream input = sock.getInputStream();  int len = 0;  while (len < 128)  {  bytes[len++] = (byte) input.read();  if (len > 4 && bytes[len - 4] == '\r' && bytes[len - 3] == '\n' && bytes[len - 2] == '\r'  && bytes[len - 1] == '\n')  {  break;  }  }  String response = new String(bytes, 0, Math.min(len, bytes.length), "UTF-8");  if (!response.startsWith("HTTP/1.1 200 "))  {  sock.close();  throw new IOException("Connect failed");  } |

### Mapping模式

|  |
| --- |
| // 首先，将远程服务器地址映射到换回地址  int loop\_port = SPVPNClient.mapToLoopback("192.168.88.88", 80);  Socket sock = new Socket();  // 其次，直接连接到环回地址  sock.connect(new InetSocketAddress("127.0.0.1", loop\_port), 30000); |

## WebView演示

可以什么都不做直接使用WebView，也可以为WebView设置代理服务器指向SDK提供的代理服务器地址

|  |
| --- |
| // 为webview设置代理服务器  // 也可以不用设置代理服务器，SDK底层自动Hook连接  SPVPNClient.setWebViewProxy(mWebView);  …  mWebView.loadUrl("http://192.68.88.88/"); |

详情请参考演示例子中的DemoWebViewActivity.java。

# 进阶集成

## 启用NC网络模式

SDK通过Android系统的VpnService（需要Android 4.0以上版本），实现虚拟网卡。因此需要修改AndroidManifest.xml增加一个Service节点

|  |
| --- |
| <!-- 首先修改AndroidManifest.xml 添加 VPNService -->  <service  android:name="com.secure.sportal.sdk.vpn.SPNCService"  android:permission="android.permission.BIND\_VPN\_SERVICE">  <intent-filter>  <action android:name="android.net.VpnService" />  </intent-filter>  </service> |

VpnService的启动是需要用户交互参与的，APP首次启用此功能时Android系统会弹出一个确认界面。因此，如果需要启用NC网络模式，需要你在代码中通过startActivityForResult ()触发确认界面，然后通过onActivityResult()接收用户的确认结果并启动VPNService。

这个工作需要在login()完成后进行，即在收到SPVPNClient.MSGID\_LOGIN\_SUCC回调后。

|  |
| --- |
| // 先检查是否需要启用NC网络模式  // （只有当前VPN账号分配了NC网络模式服务资源时才需要）  if ( SPVPNClient.needsVPNTunnel() )  {  // 试图启动VpnService  Intent intent = VpnService.prepare(getApplicationContext());  if (intent != null)  {  // 如果本APP是首次启动intent!=null，需要用户确认  startActivityForResult(intent, SPVPNClient.REQUEST\_CODE\_VPN\_TUNNEL);  }  else  {  // 如果用户已经确认过本APP此功能，则intent==null，可直接启动VpnService  onActivityResult(SPVPNClient.REQUEST\_CODE\_VPN\_TUNNEL, RESULT\_OK, intent);  }  } |
| @Override  protected void onActivityResult(int requestCode, int resultCode, Intent data)  {  // 检查requestCode是NC网络模式请求码  if (requestCode == SPVPNClient.REQUEST\_CODE\_VPN\_TUNNEL)  {  if (resultCode == Activity.RESULT\_OK)  {  // 开启NC网络模式的VPN隧道  SPVPNClient.startVPNTunnel(this, true, 10);  }  else  {  // 用户不同意启用VpnService  }  }  else  {  super.onActivityResult(requestCode, resultCode, data);  }  } |

监听NC网络模式VPN隧道状态变化

|  |
| --- |
| // 定义接收类  private class VPNTunnelStateReceiver extends BroadcastReceiver  {  @Override  public void onReceive(Context context, Intent intent)  {  StringBuilder msg = new StringBuilder("VPN隧道状态变化 ");  int state = intent.getIntExtra(SPVPNClient.EXTRA\_KEY\_NC\_TUNNEL\_STATE, 0);  if (state == SPVPNClient.NC\_TUNNEL\_STATE\_CONNECTING)  {  msg.append("连接中");  }  else if (state == SPVPNClient.NC\_TUNNEL\_STATE\_CONNECTED)  {  msg.append("已连接");  }  else if (state == SPVPNClient.NC\_TUNNEL\_STATE\_ERROR)  {  msg.append("出错了");  }  else  {  msg.append("已停止");  }  Toast.makeText(context, msg.toString(), Toast.LENGTH\_LONG).show();  Log.d("demo", msg.toString());  }  }  …  // 注册接收器  SPVPNClient.registVPNTunnelStateReceiver(this, tunnelStateReceiver);  …  // 反注册接收器  SPVPNClient.unregistBroadcastReceiver(this, tunnelStateReceiver); |

更详细的使用方法请参考演示用例中的VPNClientDemoActivity.java。

**注：调用startActivityForResult(intent, SPVPNClient.REQUEST\_CODE\_VPN\_TUNNEL);后，不能切换Activity，不然无法接收结果。您需要在onActivityResult()中再跳转Activity。**

## 断开VPN连接

当只工作在代理连接模式时，SSLVPN SDK与SSLVPN服务器间不是长连接，而SSLVPN的用户会话本身具有时效性，因此SDK的会话可能已经失效。SDK内部会保持用户会话令牌，当用户会话失效但APP又有连接请求时，SDK会自动通过SSLVPN服务器续租用户会话令牌，以确保APP能继续使用。但是，有些情况下是无法续租用户会话令牌的，例如用户修改了密码，而用旧密码认证的会话就无法续租（APP可通过SessionStatusReceiver广播消息收到离线通知），这时就需要重新用新密码login才能生效。

当APP无须再使用VPN时（例如APP退出），可调用SPVPNClient的logout()接口，退出SSLVPN登录从而彻底挂断VPN连接。

|  |
| --- |
| SPVPNClient.logout(); |

## 两步认证

SDK支持两步认证，其中

* 第一步是用户名密码方式（包括移动证书模式）
* 第二步是验证码、动态口令方式、360ID安全口令

第二步认证也是通过SPVPNClient.login()方法实现

* 通知SDK接收验证码的手机号码或电子邮件

|  |
| --- |
| Properties param = new Properties();  // 这里param里面只需要一个字段  param.setProperty(SPVPNClient.PARAM\_VCODE\_TARGET, "139012345678");  // 或者是电子邮件  // param.setProperty(SPVPNClient.PARAM\_VCODE\_TARGET, "zhangsan@example.com");  SPVPNClient.login(VPNClientDemoActivity.this, param,  new SPVPNClient.OnVPNLoginCallback(){…}); |

* 用接收的验证码或动态口令完成第二步认证

|  |
| --- |
| Properties param = new Properties();  // 这里param里面只需要一个字段  param.setProperty(SPVPNClient.PARAM\_VERIFY\_CODE, "123456");  SPVPNClient.login(VPNClientDemoActivity.this, param,  new SPVPNClient.OnVPNLoginCallback(){…}); |

可通过login()的回调方法中onVPNLoginMessage(int msgid, String msg)中的msgid确定下一步需要进行的任务，msgid的值在SPVPNClient中定义如下

|  |  |
| --- | --- |
| SPVPNClient.MSGID\_LOGIN\_SUCC | 登录成功完成 |
| SPVPNClient.MSGID\_LOGIN\_NEED\_PHONE\_NUM | 需要电话号码，以用于发送短信验证码 |
| SPVPNClient.MSGID\_LOGIN\_NEED\_EMAIL\_ADDR | 需要电子邮件，以用于发送Email验证码 |
| SPVPNClient.MSGID\_LOGIN\_NEED\_VCCODE\_SMS | 需要使用接收到的短信验证码进行下一步验证 |
| SPVPNClient.MSGID\_LOGIN\_NEED\_VCCODE\_EMAIL | 需要使用接收到的Email验证码进行下一步验证 |
| SPVPNClient.MSGID\_LOGIN\_NEED\_VCCODE\_TOKEN | 需要使用动态口令进行下一步验证 |
| SPVPNClient.MSGID\_LOGIN\_NEED\_SECID\_TOKEN | 需要使用360ID安全口令进行下一步验证 |
| SPVPNClient.MSGID\_LOGIN\_FAIL | 登录失败 |

演示代码如下，可参考演示用例中的VPNClientDemoActivity.java

|  |
| --- |
| case SPVPNClient.MSGID\_LOGIN\_NEED\_SECID\_TOKEN:  // 需要动态口令，可以用户直接手动输入，如果是 360ID 口令，还可以调用 SDK 直接获取  // 方式一，提示用户输入动态口令  SPPopupInputBox.inputBox(this, "请输入动态口令", "", new SPPopupInputBox.SPInputBoxCallback()  {  @Override  public void OnInputBoxText(String text, boolean remember)  {  if (!TextUtils.isEmpty(text))  {  // 用接收到的验证码进行第二步验证，这里还是调用 SPVPNClient.login() 方法，  // 参数只需要一个 SPVPNClient.PARAM\_VERIFY\_CODE  Properties param = new Properties();  param.setProperty(SPVPNClient.PARAM\_VERIFY\_CODE, text);  SPVPNClient.login(VPNClientDemoActivity.this, param, VPNClientDemoActivity.this);  }  }  });  // 方式二，调用360ID获取动态口令，通过onActivityResult()获取结果  LibSecIDSDKLite.generateToken(this, "", 443, "");  break; |
| // 需要接收短信验证码的手机号码  case SPVPNClient.MSGID\_LOGIN\_NEED\_PHONE\_NUM:  SPPopupInputBox.inputBox(this, "请输入接收短信验证码的手机号码", "",  new SPPopupInputBox.SPInputBoxCallback()  {  @Override  public void OnInputBoxText(String text, boolean remember)  {  if (!TextUtils.isEmpty(text))  {  // 将接收短信验证码的手机号码通知VPN，  // 这里还是调用 SPVPNClient.login() 方法，  // 参数只需要一个 SPVPNClient.PARAM\_VCODE\_TARGET  Properties param = new Properties();  param.setProperty(SPVPNClient.PARAM\_VCODE\_TARGET, text);  SPVPNClient.login(VPNClientDemoActivity.this, param,  VPNClientDemoActivity.this);  }  }  });  break; |
| // 需要接收验证码的电子邮件地址  case SPVPNClient.MSGID\_LOGIN\_NEED\_EMAIL\_ADDR:  SPPopupInputBox.inputBox(this, "请输入接收验证码的电子邮件地址", "",  new SPPopupInputBox.SPInputBoxCallback()  {  @Override  public void OnInputBoxText(String text, boolean remember)  {  if (!TextUtils.isEmpty(text))  {  // 将接收验证码的电子邮件地址通知VPN，  // 这里还是调用 SPVPNClient.login() 方法，  // 参数只需要一个 SPVPNClient.PARAM\_VCODE\_TARGET  Properties param = new Properties();  param.setProperty(SPVPNClient.PARAM\_VCODE\_TARGET, text);  SPVPNClient.login(VPNClientDemoActivity.this, param,  VPNClientDemoActivity.this);  }  }  });  break; |
| // 需要短信验证码，这时 msg 是接收短信的手机号  case SPVPNClient.MSGID\_LOGIN\_NEED\_VCCODE\_SMS:  SPPopupInputBox.inputBox(this, "请输入[" + msg + "]接收到的短信验证码", "",  new SPPopupInputBox.SPInputBoxCallback()  {  @Override  public void OnInputBoxText(String text, boolean remember)  {  if (!TextUtils.isEmpty(text))  {  // 用接收到的验证码进行第二步验证，  // 这里还是调用 SPVPNClient.login() 方法，  // 参数只需要一个 SPVPNClient.PARAM\_VERIFY\_CODE  Properties param = new Properties();  param.setProperty(SPVPNClient.PARAM\_VERIFY\_CODE, text);  SPVPNClient.login(VPNClientDemoActivity.this, param,  VPNClientDemoActivity.this);  }  }  });  break; |
| // 需要 EMail 验证码，这时 msg 是接收验证码的邮件地址  case SPVPNClient.MSGID\_LOGIN\_NEED\_VCCODE\_EMAIL:  SPPopupInputBox.inputBox(this, "请输入[" + msg + "]接收到的 EMail 验证码", "",  new SPPopupInputBox.SPInputBoxCallback()  {  @Override  public void OnInputBoxText(String text, boolean remember)  {  if (!TextUtils.isEmpty(text))  {  // 用接收到的验证码进行第二步验证，  // 这里还是调用 SPVPNClient.login() 方法，  // 参数只需要一个 SPVPNClient.PARAM\_VERIFY\_CODE  Properties param = new Properties();  param.setProperty(SPVPNClient.PARAM\_VERIFY\_CODE, text);  SPVPNClient.login(VPNClientDemoActivity.this, param,  VPNClientDemoActivity.this);  }  }  });  break; |
| // 需要动态口令，可以用户直接手动输入，  // 如果是 360ID 口令，还可以调用 SDK 直接获取  case SPVPNClient.MSGID\_LOGIN\_NEED\_VCCODE\_TOKEN:  // 提示用户输入动态口令  SPPopupInputBox.inputBox(this, "请输入接收到的 EMail 验证码", "", new SPPopupInputBox.SPInputBoxCallback()  {  @Override  public void OnInputBoxText(String text, boolean remember)  {  if (!TextUtils.isEmpty(text))  {  // 用接收到的验证码进行第二步验证，  // 这里还是调用 SPVPNClient.login() 方法，  // 参数只需要一个 SPVPNClient.PARAM\_VERIFY\_CODE  Properties param = new Properties();  param.setProperty(SPVPNClient.PARAM\_VERIFY\_CODE, text);  SPVPNClient.login(VPNClientDemoActivity.this, param,  VPNClientDemoActivity.this);  }  }  });  break; |

## 证书登录和国密支持

SDK支持使用软证书、硬件证书和签名验签三种证书登录

* 软证书，证书以文件（例如.pfx、.p12）等方式存放，APP可直接获取到证书的完整数据，包括证书的公钥和私钥。**SSLVPN SDK与SSLVPN服务器间的SSL连接使用此证书做双向认证。**
* 硬件证书，证书存放在TF卡中，APP需要通过专用API才能获取证书的公钥数据，且私钥数据通常不能获取到，目前仅支持国密SM2证书。**SSLVPN SDK与SSLVPN服务器间的SSL连接使用此证书做双向认证。**
* 签名验签，这种方式是通过证书公钥，证书公钥的签名（用此证书的私钥进行签名）进行认证。其中如果是国密SM2证书则使用国密SM3签名，其他则使用RSA签名。

详细使用方法请参考演示用例中的DemoCertLoginActivity.java.java

***注：如果使用TF卡中的硬件证书，需集成TF卡厂商的相应驱动so文件到libs***

### 软证书登录

SDK通过复用用户名和密码字段实现证书登录，在第一步登录中

* 用户名字段，
  + 传入经过base64编码的证书内容，

例如Base64.encodeToString(SPFileUtil.readFile("/storage/emulated/0/mycert.pfx "), Base64.NO\_WRAP)

* + 或证书文件在手机中的全路径，例如/storage/emulated/0/mycert.pfx
* 密码字段，软证书私钥的密码

用例如下

|  |
| --- |
| Properties params = new Properties();  params.setProperty(SPVPNClient.PARAM\_VPN\_HOST, "172.16.1.233");  params.setProperty(SPVPNClient.PARAM\_VPN\_PORT, "443");  params.setProperty(SPVPNClient.PARAM\_AUTH\_SERVER, "CERT\_AUTH");  // pfx证书文件内容base64编码  String filename = SPFileUtil.getSdCardPath() + "/mycert.pfx";  String pfx = Base64.encodeToString(SPFileUtil.readFile(filename), Base64.NO\_WRAP);  params.setProperty(SPVPNClient.PARAM\_AUTH\_USERNAME, pfx);  // 也可直接传入证书文件路径作为用户名  // params.setProperty(SPVPNClient.PARAM\_AUTH\_USERNAME, SPFileUtil.getSdCardPath() + "/mycert.pfx");  // 私钥密码  params.setProperty(SPVPNClient.PARAM\_AUTH\_PASSWORD, "aaaaaa");  SPVPNClient.login(…); |

### 硬件证书登录和国密加密

目前的硬件证书都是国密SM2证书，需要特殊的native层libsecure\_portal.so库才能实现SSL连接硬件证书双向认证。启用SM2硬件证书时，请使用SDK包中libs\_sm2下的libsecure\_portal.so库文件。

使用硬件证书登录时，无须传入用户名和密码字段，但是需要告知SDK 硬件TF卡的PIN码（注意如需启用硬件证书在SSLVPN服务器上需要配置上相应的应用名和容器名）。

用例如下

|  |
| --- |
| Properties params = new Properties();  params.setProperty(SPVPNClient.PARAM\_VPN\_HOST, "172.16.1.233");  params.setProperty(SPVPNClient.PARAM\_VPN\_PORT, "443");  params.setProperty(SPVPNClient.PARAM\_AUTH\_SERVER, "CERT\_AUTH");  // TF卡PIN码  params.setProperty(SPVPNClient.PARAM\_SDCARD\_PIN, "66666666");  SPVPNClient.login(…); |

目前数据通信加密支持RSA-SM4-SHA1、ECC-SM1-SM3和ECC-SM4-SM3三种加密方式，其中RSA-SM4-SHA1和ECC-SM1-SM3是硬件加密需要相应的国密硬件TF卡及使用libs\_sm2下的native层libsecure\_portal.so。

### 签名验签登录

签名验签登录，支持调用者传入证书公钥和签名，这种情况复用用户名和密码字段，用户名字段是证书公钥PEM内容的base64编码，密码字段是签名串。也支持调用者穿入TF卡PIN码SDK自行从TF卡中读取证书公钥及计算签名。

* 调用者传入方式

|  |
| --- |
| Properties params = new Properties();  params.setProperty(SPVPNClient.PARAM\_VPN\_HOST, "172.16.1.233");  params.setProperty(SPVPNClient.PARAM\_VPN\_PORT, "443");  params.setProperty(SPVPNClient.PARAM\_AUTH\_SERVER, "CERT\_AUTH");  // 用户名字段是证书公钥PEM内容的base64编码  params.setProperty(SPVPNClient.PARAM\_AUTH\_USERNAME,  Base64.encodeToString(public\_key,Base64.NO\_WRAP));  // 密码字段是签名串  params.setProperty(SPVPNClient.PARAM\_AUTH\_PASSWORD, "这里是公钥的签名串");  SPVPNClient.login(…); |

* SDK自行读取TF卡硬件证书和计算签名方式

|  |
| --- |
| Properties params = new Properties();  params.setProperty(SPVPNClient.PARAM\_VPN\_HOST, "172.16.1.233");  params.setProperty(SPVPNClient.PARAM\_VPN\_PORT, "443");  params.setProperty(SPVPNClient.PARAM\_AUTH\_SERVER, "CERT\_AUTH");  // 需要告诉SDK硬件TF卡的PIN码  params.setProperty(SPVPNClient.PARAM\_SDCARD\_PIN, "66666666");  SPVPNClient.login(…); |

SDK中的SPVPNTool类提供了API可供调用者解析国密SM2证书、读取TF硬件卡证书公钥数据等，具体请参考后面章节的API说明。

## 扩展和附加数据

在有些定制版场景时，SSLVPN服务器需要客户端传递一些特殊标示数据，SSLVPN服务器可根据这些特定的标示数据实现一些定制功能。SDK允许在第一步登录时的param参数中输入一些特殊字段数据，而SDK对这种数据不做任何识别和处理，直接透传到SSLVPN服务器端。SDK的处理方式是param数据中，所有字段名称以"extra\_"开头的数据，都会将字段名及对应的值在登录消息中透传到SSLVPN服务端。

例如

|  |
| --- |
| …  params.setProperty("extra\_branch", "XXX办事处");  params.setProperty("extra\_pkgname", "com.example.OA");  …  SPVPNClient.login(…); |

## 监听用户会话状态

前面已经描述过可通过SPVPNClient.resitSessionStateReceiver(this, sessionStatusReceiver);注册Receiver接收SSLVPN用户会话状态消息。这里说明一下消息中的topic和content字段详细定义：

|  |
| --- |
| @Override  public void onReceive(Context context, Intent intent)  {  String topic = intent.getStringExtra("topic");  String content = intent.getStringExtra("content");  } |

* topic，目前只有”session\_status”
* content，是一个JSON字符串，当topic为session\_status时定义入下

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 必填 | 说明 |
| session\_state | string | YES | 表示当前会话状态，有两个值："online", "offline"，当为offline时其他proxy\_tunnel字段都无效（因为会话已经无效） |
| proxy\_tunnel\_errcode | string | NO | 当前代理连接模式通道状态，为”0”或不存在时，表示工作正常，否则表示错误码 |
| proxy\_tunnel\_errmsg | string | NO | 当proxy\_tunnel\_errcode存在且不为”0”时对应的错误描述信息 |

样例

|  |
| --- |
| {"session\_state":"online", "proxy\_tunnel\_errcode":"0", "proxy\_tunnel\_errmsg:""}  {"session\_state":"offline", "proxy\_tunnel\_errcode":"0", "proxy\_tunnel\_errmsg:""}  {"session\_state":"online", "proxy\_tunnel\_errcode":"33555460", "proxy\_tunnel\_errmsg:"用户凭据无效"} |

# API说明

## SPVPNClient

### login

|  |  |
| --- | --- |
| public static void login(Context context, Properties param,  SPVPNClient.OnVPNLoginCallback callback) | |
| 功能 | 登录SSLVPN服务器（拨号） |
| 参数 | Context context  Properties param 认证参数，具体需要的的参数字段名，参考前面章节  SPVPNClient.OnVPNLoginCallback callback 回调监听器 |
| 返回值 | 无 |
| 备注 |  |

### logout

|  |  |
| --- | --- |
| public static void logout() | |
| 功能 | 退出SSLVPN登录 |
| 参数 | 无 |
| 返回值 | 无 |
| 备注 |  |

### loadNativeLibrary

|  |  |
| --- | --- |
| public static void loadNativeLibrary(String name) | |
| 功能 | 加载一个APK/libs下的native层so动态库文件 |
| 参数 | String name 库名称，例如文件名是libxwalkcore.so，则是库名是xwalkcore |
| 返回值 | 无 |
| 备注 | 由于受Android 7.0开始的安全机制影响，SDK无法在Android 7.0以后的版本对APP类动态加载的so文件进行Hook，调用者可以在登录SSLVPN前调用此方法预先加载后续会使用的native动态库文件，确保SDK在登录后能正确Hook。 |

### needsNCVpnTunnel

|  |  |
| --- | --- |
| public static boolean needsNCVpnTunnel() | |
| 功能 | 检测是否需要启动NC网络模式VPN隧道 |
| 参数 | 无 |
| 返回值 | boolean 是否需要启动NC网络模式VPN隧道 |
| 备注 |  |

### startVPNTunnel

|  |  |
| --- | --- |
| public static void startVPNTunnel(Context context, boolean showToast, int tryTimes) | |
| 功能 | 启动NC网络模式VPN隧道 |
| 参数 | Conext context  boolean showToast 是否在状态变化时显示toast消息  int tryTimes 启动失败或中断时的重试次数 |
| 返回值 | 无 |
| 备注 |  |

### registVPNTunnelStateReceiver

|  |  |
| --- | --- |
| public static void registVPNTunnelStateReceiver(Context context, BroadcastReceiver receiver) | |
| 功能 | 注册NC网络模式VPN隧道状态监听器 |
| 参数 | Context context  BroadcastReceiver receiver |
| 返回值 | 无 |
| 备注 |  |

### resitSessionStateReceiver

|  |  |
| --- | --- |
| public static void resitSessionStateReceiver(Context context, BroadcastReceiver receiver) | |
| 功能 | 注册SSLVPN用户会话状态监听器 |
| 参数 | Context context  BroadcastReceiver receiver |
| 返回值 | 无 |
| 备注 |  |

### unregistBroadcastReceiver

|  |  |
| --- | --- |
| public static void unregistBroadcastReceiver(Context context, BroadcastReceiver receiver) | |
| 功能 | 反注册广播消息接听器 |
| 参数 | Context context  BroadcastReceiver receiver |
| 返回值 | 无 |
| 备注 |  |

### isNetHooking

|  |  |
| --- | --- |
| public static boolean isNetHooking() | |
| 功能 | SDK底层Hook是否正常工作 |
| 参数 | 无 |
| 返回值 | boolean Hook是否正常工作 |
| 备注 | 在一些极少数机型或系统下，SDK的Hook机制无法工作，可用此方法检测。 |

### getHttpProxy

|  |  |
| --- | --- |
| public static java.net.Proxy getHttpProxy() | |
| 功能 | 获取SDK提供的代理服务器信息 |
| 参数 | 无 |
| 返回值 | java.net.Proxy SDK提供的代理服务器信息 |
| 备注 |  |

### getProxyPort

|  |  |
| --- | --- |
| public static int getProxyPort() | |
| 功能 | 获取SDK提供的代理服务器端口（地址是127.0.0.1） |
| 参数 | 无 |
| 返回值 | int SDK提供的代理服务器端口 |
| 备注 |  |

### setSSLTimeout

|  |  |
| --- | --- |
| public static void setSSLTimeout(int conn\_timeout, int so\_timeout) | |
| 功能 | 设置SDK底层SSL连接的超时时间 |
| 参数 | int conn\_timeout 连接超时，单位为秒，默认是60秒  int so\_timeout 数据超时，单位为秒，默认是30秒 |
| 返回值 | 无 |
| 备注 |  |

### addBypassPackageName

|  |  |
| --- | --- |
| public static void addBypassPackageName(String pkgName) | |
| 功能 | 添加不能进入NC网络模式VPN隧道的APP包名 |
| 参数 | String pkgName 不能进入NC网络模式VPN隧道的APP包名 |
| 返回值 |  |
| 备注 | 需要Android 5.0以上版本才支持，指定包名的APP的网络连接不会进入NC网络模式的VPN隧道 |

### addBypassAddress

|  |  |
| --- | --- |
| public static void addBypassAddress(String host, int fromPort, int toPort) | |
| 功能 | 添加旁路目标地址 |
| 参数 | String host 旁路主机名或IP地址  int fromPort 旁路端口范围起始端口  int toPort 旁路端口范围结束地址 |
| 返回值 | 无 |
| 备注 | 设置旁路目标地址   * 到这些目标地址端口范围内的连接都不会进入VPN的代理连接隧道 * 这些目标地址不会进入NC网络模式隧（NC只能基于IP地址级过滤）   而是直接连接。 |

### setWebViewProxy

|  |  |
| --- | --- |
| public static void setWebViewProxy(WebView webview) | |
| 功能 | 为WebView设置代理服务器，让WebView使用SDK提供的代理服务器 |
| 参数 | WebView webview |
| 返回值 | 无 |
| 备注 |  |

### getServiceList

|  |  |
| --- | --- |
| public static List<SPServiceInfo> getServiceList() | |
| 功能 | 获取SSLVPN用户可用的内部服务器资源列表 |
| 参数 | 无 |
| 返回值 | List<SPServiceInfo> 当前登录SSLVPN账号可用的内部服务器资源列表 |
| 备注 |  |

### sconnect

|  |  |
| --- | --- |
| public static Socket sconnect(String host, int port, int timeout, int sotimeout) throws IOException | |
| 功能 | 建立一个到内部服务器的TCP连接 |
| 参数 | String host 内部服务器主机名或IP  int port 内部服务器端口  int timeout 连接超时  int sotimeout 数据超时 |
| 返回值 | Socket 已经连上内部服务器的TCP Socket |
| 备注 |  |

### mapToLoopback

|  |  |
| --- | --- |
| public static int mapToLoopback(String host, int port) | |
| 功能 | 映射一个内部服务器的地址到本地手机环回地址12.0.0.1上 |
| 参数 | String host 内部服务器主机名或IP  int port 内部服务器端口 |
| 返回值 | int  >0 成功映射后在127.0.0.1上的监听端口  其他 映射失败 |
| 备注 |  |

### updateAuthExtras

|  |  |
| --- | --- |
| public static void updateAuthExtras(Properties extras) | |
| 功能 | 更新额外数据，SDK内部在续租用户会话令牌时会将更新后的额外数据发送到SSLVPN服务器 |
| 参数 | Properties extras 字段名以"extra\_"开头的额外数据 |
| 返回值 | 无 |
| 备注 |  |

## SPVPNTool

### encryptFile

|  |  |
| --- | --- |
| public static boolean encryptFile(String srcFile, String dstFile, String password) | |
| 功能 | 加密文件 |
| 参数 | String srcFile 明文文件路径  String dstFile 已加密目标文件路径，如果为空，则覆盖srcFile  String password 加密密码 |
| 返回值 | 是否成功 |
| 备注 |  |

### decryptFile

|  |  |
| --- | --- |
| public static boolean decryptFile(String srcFile, String dstFile, String password) | |
| 功能 | 解密文件 |
| 参数 | String srcFile 密文源文件路径  String dstFile 已解密目标文件路径，如果为空，则覆盖srcFile  String password 解密密码 |
| 返回值 | 是否成功 |
| 备注 |  |

### putSSLFingerprint

|  |  |
| --- | --- |
| public static boolean putSSLFingerprint(String host, int port, String fingerprint) | |
| 功能 | 添加SSL证书指纹 |
| 参数 | String host 服务器主机名或IP  int port 服务器端口  String fingerprint 服务器证书的sha1指纹，  这个指纹是一个40个16进制字符串  例如：5C:E4:DF:B4:16:94:36:C1:50:13:1D:15:1F:FB:FB:9A:94:0B:D2:FD  或：5CE4DFB4169436C150131D151FFBFB9A940BD2FD |
| 返回值 | 是否添加成功 |
| 备注 |  |

### clearSSLFingerprint

|  |  |
| --- | --- |
| public static void clearSSLFingerprint() | |
| 功能 | 清空已添加的SSL证书指纹数据 |
| 参数 | 无 |
| 返回值 | 无 |
| 备注 |  |

### checkSSLFingerprint

|  |  |
| --- | --- |
| public static boolean checkSSLFingerprint(String host, int port, String servername, String fingerprint) | |
| 功能 | 检测服务器SSL证书指纹是否匹配 |
| 参数 | String host 服务器主机名或IP  int port 服务器端口  String servername SSL-SNI主机名，如果为空则使用host  String fingerprint 服务器证书的sha1指纹，  这个指纹是一个40个16进制字符串  例如：5C:E4:DF:B4:16:94:36:C1:50:13:1D:15:1F:FB:FB:9A:94:0B:D2:FD  或：5CE4DFB4169436C150131D151FFBFB9A940BD2FD |
| 返回值 | 服务器SSL证书指纹与输入的是否匹配 |
| 备注 |  |

### extractPEMs

|  |  |
| --- | --- |
| public static String[] extractPEMs(Context context, byte[] pfx, String password) | |
| 功能 | 拆分解析PFX证书，支持国密SM2证书解析 |
| 参数 | Context context  byte[] pfx PFX证书类容  String password 私钥密码 |
| 返回值 | String[] 解析后的证书PEM内容的base64编码，依次为证书公钥、私钥 |
| 备注 |  |

### getTFCertInfo

|  |  |
| --- | --- |
| public static SPX509Info getTFCertInfo(String pin, String app, String container) | |
| 功能 | 读取国密TF卡证书中的证书数据 |
| 参数 | String pin TF卡PIN码  String app 应用名  String container 容器名 |
| 返回值 | SPX509Info 证书信息 |
| 备注 |  |

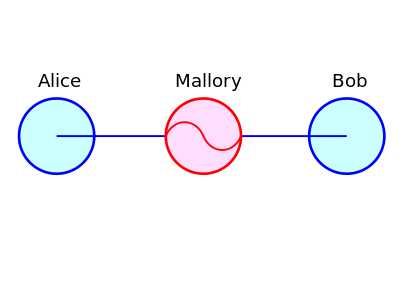
# 其他

## 防止中间人攻击

### 什么是中间人攻击

中间人攻击（英语：Man-in-the-middle attack，缩写：MITM）是指攻击者与通讯的两端分别创建独立的联系，并交换其所收到的数据，使通讯的两端认为他们正在通过一个私密的连接与对方直接对话，但事实上整个会话都被攻击者完全控制。在中间人攻击中，攻击者可以拦截通讯双方的通话并插入新的内容。在许多情况下这是很简单的（例如，在一个未加密的Wi-Fi 无线接入点的接受范围内的中间人攻击者，可以将自己作为一个中间人插入这个网络）。

一个中间人攻击能成功的前提条件是攻击者能将自己伪装成每一个参与会话的终端，并且不被其他终端识破。中间人攻击是一个（缺乏）相互认证的攻击。大多数的加密协议都专门加入了一些特殊的认证方法以阻止中间人攻击。例如，SSL协议可以验证参与通讯的一方或双方使用的证书是否是由权威的受信任的数字证书认证机构颁发，并且能执行双向身份认证。



中间人攻击示意图

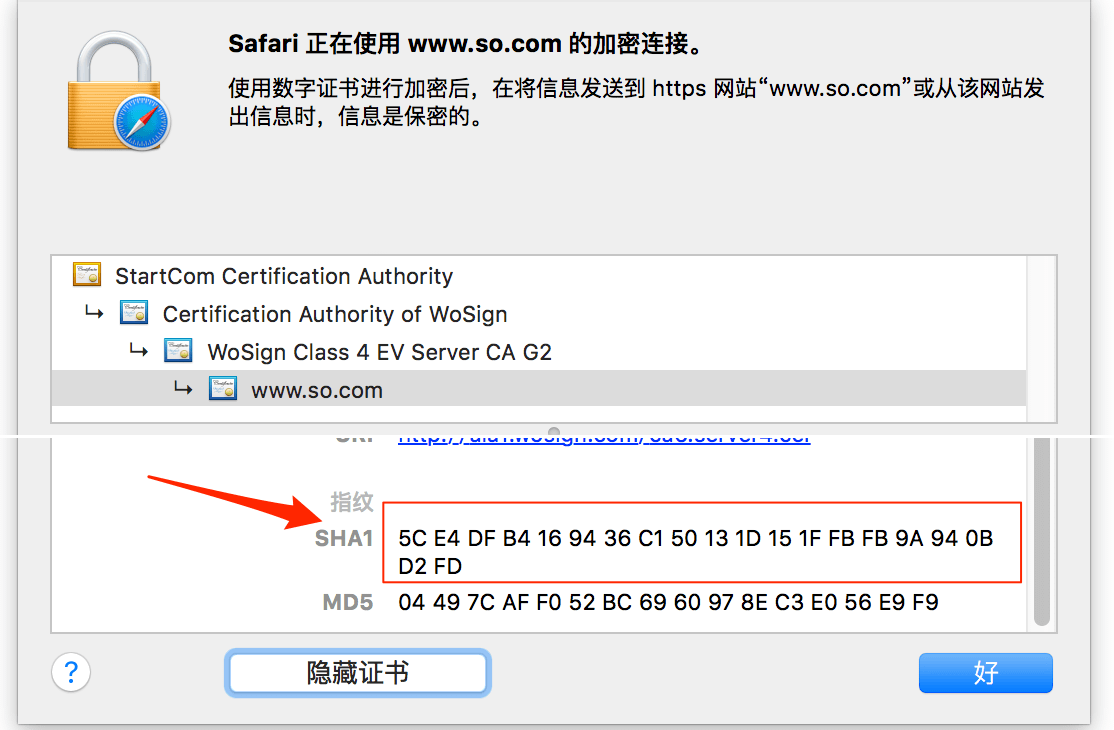
### 通过API添加SSL证书指纹信息

SSLVPN SDK通过对服务器端证书的指纹校验可以有效的防止中间人攻击。实现原理为：

* 应用APP通过SPSDKClient的putSSLFingerprint()方法，将指定SSL服务器的证书SHA1指纹放入SDK（这里我们认为调用者得到的指纹是安全和正确的，因此调用者需要确保这个指纹的安全性），SDK在内存中维护一个指纹列表
* SDK每次连接这些SSL服务器时，从SSL连接中获取到服务器证书的SHA1指纹
* SDK将从连接中得到的服务器证书指纹与内存中对应服务器的指纹进行比对，如果校验失败则直接断开连接。

获取服务端证书指纹的方法有很多，这里我们以服务器<https://www.so.com>为例。

* 方法一，在有服务器端证书情况下，直接得到（导入系统在系统证书列表中查看，或者通过openssl相关命令）
* 方法二，确保与服务器连接环境安全，使用浏览器访问服务器，然后通过浏览器查看证书信息



* 方法三，确保与服务器连接环境安全，使用OpenSSL获取服务器证书指纹

注意：因为www.so.com和so.com是同一个服务器且默认是so.com的证书，因此需要通过-servername指定SNI为**www.so.com**

* + 两步方式

首先获取并保存证书PEM

|  |
| --- |
| openssl s\_client -showcerts -servername www.so.com -connect www.so.com:443 </dev/null 2>/dev/null|openssl x509 -outform PEM >so.pem |

然后计算PEM证书的SHA1值

|  |
| --- |
| openssl x509 -in so.pem -sha1 -noout -fingerprint |
| SHA1 Fingerprint=5C:E4:DF:B4:16:94:36:C1:50:13:1D:15:1F:FB:FB:9A:94:0B:D2:FD |

* + 一步到位

|  |
| --- |
| openssl s\_client -servername www.so.com -connect www.so.com:443 < /dev/null 2>/dev/null | openssl x509 -fingerprint -noout -in /dev/stdin |
| SHA1 Fingerprint=5C:E4:DF:B4:16:94:36:C1:50:13:1D:15:1F:FB:FB:9A:94:0B:D2:FD |

### 自动加载SSL证书指纹信息

SDK还支持自动从配置文件加载SSL指纹数据，在APP的assets目录下创建一个文本文件并命名为sportal\_ssl.conf，这个文件的内容是一个JSON对象，格式如下

|  |
| --- |
| {  "fingerprints":[{  "host":"www.so.com",  "port":"443",  "fingerprint":"5C:E4:DF:B4:16:94:36:C1:50:13:1D:15:1F:FB:FB:9A:94:0B:D2:FD"  }]  } |

其中fingerprints字段值是一个数组，可以配置多条纪录。

## 错误码列表及定义

|  |  |
| --- | --- |
| 0FFFFFFF | 网络错误 |
| FFFFFFFF | 网络错误 |
| 0F000010 | 该服务器不是一个有效的SSLVPN服务器 |
| 0F0000F0 | 服务器SSL证书指纹不匹配 |
| 0F0000F1 | 初始化国密TF加密卡失败 |
| 0F0000F2 | 初始化国密TF加密卡失败:容器错误 |
| 0F0000F3 | 初始化国密TF加密卡失败:PIN码错误 |
|  |  |
|  |  |
| 0100040D | 网关服务已关闭，请联系管理员 |
| 0100040F | 服务授权失败 |
| 01000410 | 不能连接到应用服务器 |
| 02000401 | 无效认证协议 |
| 02000402 | 无效认证服务器 |
| 02000403 | 无效的用户组 |
| 02000404 | 无效用户名 |
| 02000405 | 无效用户名或密码 |
| 02000406 | 无效服务信息 |
| 02000407 | 无效命令 |
| 02000408 | 无效AAPP |
| 02000409 | 无效AACR |
| 02000410 | 无效口令 |
| 0200040A | 访问控制策略不允许用户从该源地址登录 |
| 0200040B | 无效终端安全策略 |
| 0200040C | 无效检查规则 |
| 0200040D | 无效接口 |
| 0200040E | 无效会话 |
| 0200040F | 管理限制 |
| 02000410 | 无效口令 |
| 02000411 | 用户被锁定 |
| 02000412 | VC访问控制策略无效 |
| 02000413 | 该账号已开启硬件绑定功能，请使用允许的硬件设备进行登录 |
| 02000414 | 注册用户失败 |
| 02000415 | 用户重绑定间隔时间错误 |
| 02000416 | 用户没有权限再次绑定，不能登入 |
| 02000417 | 条件失败 |
| 02000418 | 新旧密码不允许相同 |
| 02000419 | 密码强度不符合要求 |
| 0200041A | 密码需要初始化 |
| 0200041E | 要求首次登录修改密码或密码已过期 |
| 0200041F | AD账户密码过期 |
| 02000421 | 系统发送短信失败 |
| 02000420 | 手机号码不存在 |
| 02000422 | 短信密码错误 |
| 02000424 | 证书错误 |
| 02000425 | 系统失败 |
| 02000426 | 系统组件正在初始化，请稍候 |
| 02000427 | 不允许重复登录 |
| 02000428 | 不允许修改密码 |
| 02000429 | 密码即将过期 |
| 0200042A | 远程认证服务器不允许修改密码 |
| 0200042B | AD用户帐号过期 |
| 0200042C | AD用户被锁定 |
| 0200042D | AD帐号被禁用 |
| 0200042E | 帐号被禁用 |
| 0200042F | 无效的用户 |
| 02000438 | IP分配失敗 |
| 020004DC | 用户还未获取短信密码 |
| 020004DD | 用户密码超时 |
| 020004DE | 用户密码错误 |
| 02000432 | 无效的接入门户配置 |
| 02000433 | 无效的证书 |
| 02000434 | 在线用户数超出用户许可限制 |
| 02000435 | 在线会话数超出并发会话许可限制 |
| 02000436 | 不允许的登录方式 |
| 02000437 | 不在允许的时间范围 |
| 0200043A | 动态口令无效 |
| 0200043B | 超过最大共享用户数限制 |
| 0200043C | 短信发送失败 |
| 0200043D | 短信验证失败，请重试 |
| 0200043E | 审批短信已发送，请继续等待审批人回复 |
| 0200043F | 审批超时，您可以重新登录等待审批 |
| 02000440 | 审批人已经拒绝您的请求 |
| 02000441 | 认证未通过 |
| 02000442 | 业务名重复 |
| 02000443 | 业务超过最大限制 |
| 02000448 | 移动终端越狱后禁止访问 |
| 02000449 | 移动终端ROOT后禁止访问 |
| 0200044a | 移动终端病毒扫描不安全禁止访问 |
| 0200044b | 移动终端被管理员禁止访问 |
| 04000401 | 不允许使用 SSL version 2 |
| 04000402 | 套接字数据传输错误 |
| 04000403 | SSL协议错误 |
| 04000404 | 获取证书失败 |
| 04000405 | 证书签名无效 |
| 04000406 | CRL签名无效 |
| 04000407 | 证书不在有效期 |
| 04000408 | 证书已经过期 |
| 04000409 | 证书的“不能早于”字段包含无效时间 |
| 0400040A | 证书的“不能晚于”字段包含无效时间 |
| 0400040B | 证书已经被吊销 |
| 0400040C | SSL创建失败 |
| 0400040D | 不允许转发 |
| 04000410 | 连接应用服务器失败 |
| 04000516 | 登录失败 |
| 04000517 | 因为授权不通过而失败 |
| 04000518 | 因为用户取消了操作而失败 |
| 04000519 | 因为连接中断而失败 |
| 0F0000F0 | 服务器SSL证书指纹不匹配 |
| 0F000009 | 出现并发登录请求（另外一个登录正在进行中） |
| 0F000010 | 不是正确的SSLVPN服务器 |
| 0F000011 | 运营商错误，服务器不运行客户端从其所在运营商网络登录 |