# **Android HTTPS 自制证书实现双向认证(OkHttp + Retrofit + Rxjava)**

由于最近要做一个安全性比较高的项目，因此需要用到HTTPS进行双向认证。由于设计项目架构的时候，客户端是采用MVVM架构，基于DataBinding + Retrofit + Rxjava来实现Android端。

查阅很多资料，基于原生HttpClient实现双向认证的例子很多，但对于Retrofit的资料网上还是比较少，官方文档也是一句带过，没有具体的介绍。

看了 [《Android中https请求的单向认证和双向认证》](http://blog.csdn.net/u011394071/article/details/52880062" \t "_blank)，给了我很大的启发，于是尝试着博主的方式制作证书，再次尝试的时候果然成功了。

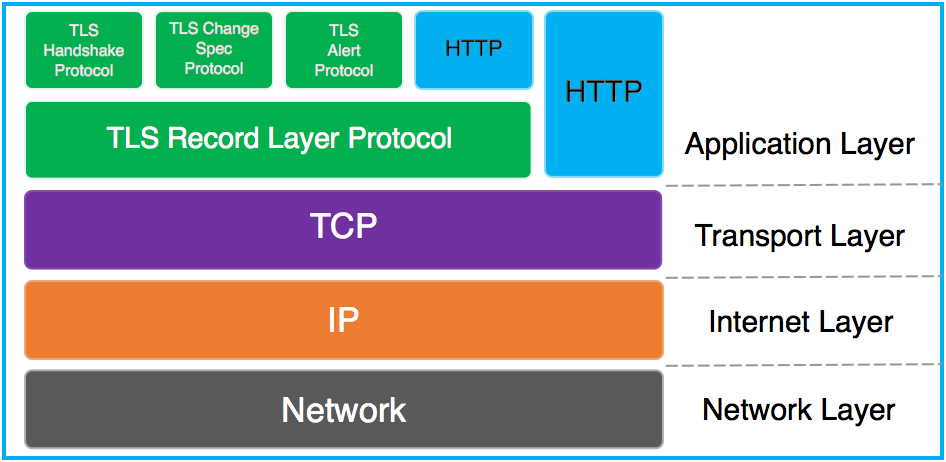
### 科普一下，什么是HTTPS？

简单来说，HTTPS就是**“安全版”**的**HTTP**, HTTPS = HTTP + SSL。HTTPS相当于在应用层和TCP层之间加入了一个SSL（或TLS），SSL层对从应用层收到的数据进行加密。TLS/SSL中使用了RSA非对称加密，对称加密以及HASH算法。

RSA算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但那时想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

**SSL**：（Secure Socket Layer，安全套接字层），为Netscape所研发，用以保障在Internet上数据传输之安全，利用数据加密(Encryption)技术，可确保数据在网络上之传输过程中不会被截取。它已被广泛地用于Web浏览器与服务器之间的身份认证和加密数据传输。SSL协议位于TCP/IP协议与各种应用层协议之间，为数据通讯提供安全支持。  
**SSL协议可分为两层：**  
**SSL记录协议（SSL Record Protocol）：**它建立在可靠的传输协议（如TCP）之上，为高层协议提供数据封装、压缩、加密等基本功能的支持。  
**SSL握手协议（SSL Handshake Protocol）：**它建立在SSL记录协议之上，用于在实际的数据传输开始前，通讯双方进行身份认证、协商加密算法、交换加密密钥等。

**TLS**：(Transport Layer Security，传输层安全协议)，用于两个应用程序之间提供保密性和数据完整性。TLS 1.0是IETF（Internet Engineering Task Force，Internet工程任务组）制定的一种新的协议，它建立在SSL 3.0协议规范之上，是SSL 3.0的后续版本，可以理解为SSL 3.1，它是写入了 RFC的。  
**该协议由两层组成： TLS 记录协议（TLS Record）和 TLS 握手协议（TLS Handshake）**。



TLS处于的位置

## 进入正文

##### 基于Retrofit实现HTTPS思路

由于Retrofit是基于OkHttp实现的，因此想通过Retrofit实现HTTPS需要给Retrofit设置一个OkHttp代理对象用于处理HTTPS的握手过程。代理代码如下：

OkHttpClient okHttpClient = new OkHttpClient.Builder()

.sslSocketFactory(SSLHelper.getSSLCertifcation(context))//为OkHttp对象设置SocketFactory用于双向认证

.hostnameVerifier(new UnSafeHostnameVerifier())

.build();Retrofit retrofit = new Retrofit.Builder()

.baseUrl("https://10.2.8.56:8443")

.addConverterFactory(GsonConverterFactory.create())//添加 json 转换器

.addCallAdapterFactory(RxJavaCallAdapterFactory.create())//添加 RxJava 适配器

.client(okHttpClient)//添加OkHttp代理对象

.build();

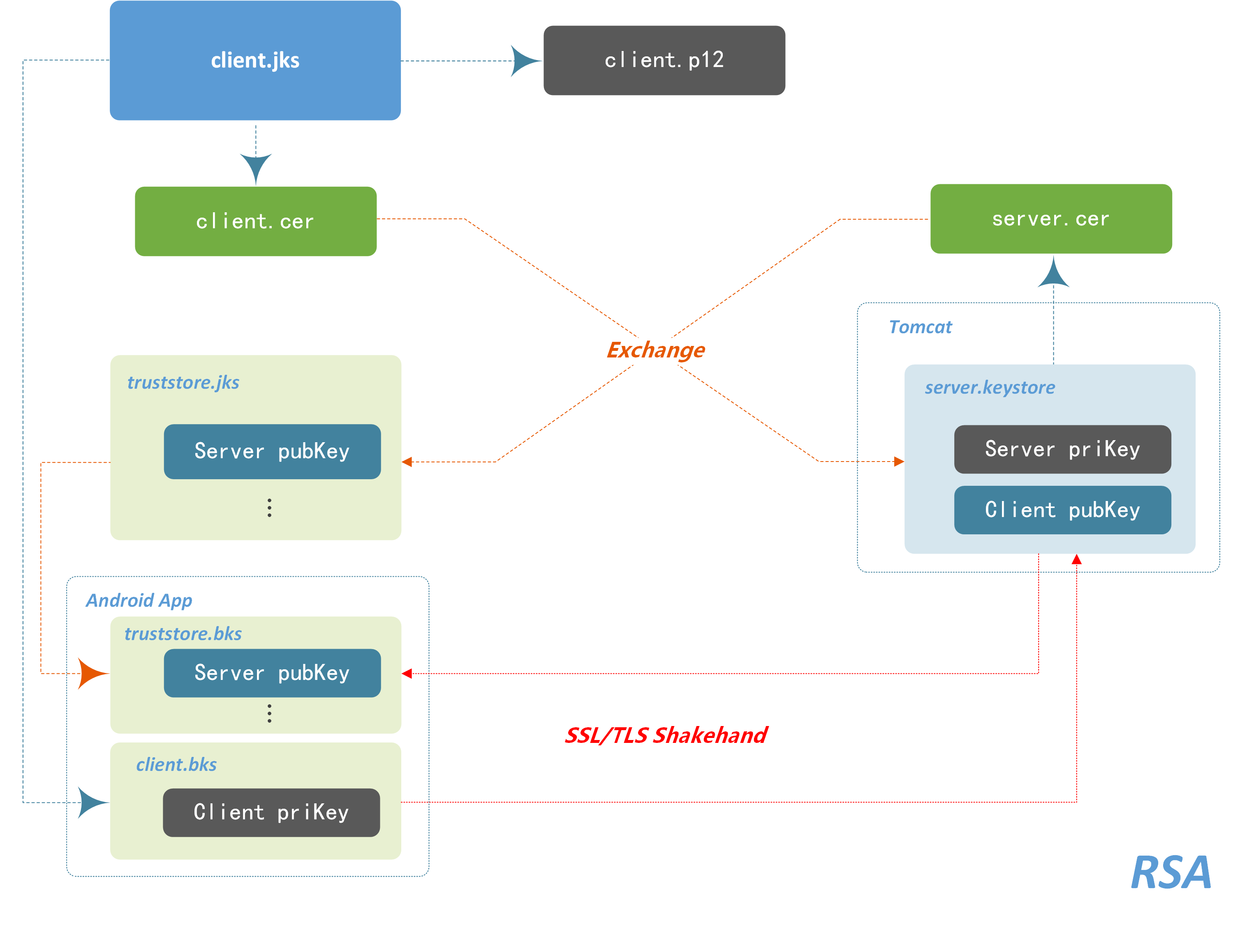
## 证书制作思路：

首先对于双向证书验证，也就是说，  
**客户端持有服务端的公钥证书，并持有自己的私钥**，**服务端持有客户的公钥证书，并持有自己私钥**，  
**建立连接的时候，客户端利用服务端的公钥证书来验证服务器是否上是目标服务器；服务端利用客户端的公钥来验证客户端是否是目标客户端。（**请参考RSA非对称加密以及HASH校验算法**）**  
服务端给客户端发送数据时，需要将服务端的证书发给客户端验证，验证通过才运行发送数据，同样，客户端请求服务器数据时，也需要将自己的证书发给服务端验证，通过才允许执行请求。

##### 下面我画了一个图，来帮助大家来理解双向认证的过程，证书生成流程，以及各个文件的作用，大家可以对照具体步骤来看

相关格式说明  
JKS：数字证书库。**JKS里有KeyEntry和CertEntry，在库里的每个Entry都是靠别名（alias）来识别的。**  
P12：是PKCS12的缩写。同样是一个**存储私钥的证书库**，由**.jks**文件导出的，用户在PC平台安装，**用于标示用户的身份**。  
CER：俗称数字证书，**目的就是用于存储公钥证书**，任何人都可以获取这个文件 。  
BKS：由于Android平台不识别**.keystore**和**.jks**格式的证书库文件，因此Android平台引入一种的证书库格式，BKS。

有些人可能有疑问，为什么Tomcat只有一个server.keystore文件，而客户端需要两个库文件？  
因为有时客户端可能需要访问过个服务，而服务器的证书都不相同，因此客户端需要制作一个**truststore**来存储受信任的服务器的证书列表。因此为了规范创建一个**truststore.jks**用于存储受信任的服务器证书，创建一个**client.jks**来存储客户端自己的私钥。对于只涉及与一个服务端进行双向认证的应用，将**server.cer**导入到**client.jks**中也可。



## 具体步骤如下：

**1.生成客户端keystore**

keytool -genkeypair -alias client -keyalg RSA -validity 3650 -keypass 123456 -storepass 123456 -keystore client.jks

**2.生成服务端keystore**

keytool -genkeypair -alias server -keyalg RSA -validity 3650 -keypass 123456 -storepass 123456 -keystore server.keystore//注意：CN必须与IP地址匹配，否则需要修改host

**3.导出客户端证书**

keytool -export -alias client -file client.cer -keystore client.jks -storepass 123456

**4.导出服务端证书**

keytool -export -alias server -file server.cer -keystore server.keystore -storepass 123456

**5.重点：证书交换**

将客户端证书导入服务端keystore中，再将服务端证书导入客户端keystore中， 一个keystore可以导入多个证书，生成证书列表。

生成客户端信任证书库(由服务端证书生成的证书库)：

keytool -import -v -alias server -file server.cer -keystore truststore.jks -storepass 123456

将客户端证书导入到服务器证书库(使得服务器信任客户端证书)：

keytool -import -v -alias client -file client.cer -keystore server.keystore -storepass 123456

**6.生成Android识别的BKS库文件**

用Portecle工具转成bks格式，最新版本是1.10。

下载链接：https://sourceforge.net/projects/portecle/

运行protecle.jar将client.jks和truststore.jks分别转换成client.bks和truststore.bks,然后放到android客户端的assert目录下

　File -> open Keystore File -> 选择证书库文件 -> 输入密码 -> Tools -> change keystore type -> BKS -> save keystore as -> 保存即可

这个操作很简单，如果不懂可自行百度。

我在Windows下生成BKS的时候会报错失败，后来我换到CentOS用OpenJDK1.7立马成功了，如果在这步失败的同学可以换到Linux或Mac下操作，

将生成的BKS拷贝回Windows即可。

**7.配置Tomcat服务器**

修改server.xml文件，配置8443端口

<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"

maxThreads="150" SSLEnabled="true" scheme="https" secure="true"

clientAuth="true" sslProtocol="TLS"

keystoreFile="${catalina.base}/key/server.keystore" keystorePass="123456"

truststoreFile="${catalina.base}/key/server.keystore" truststorePass="123456"/>

备注: - keystoreFile：指定服务器密钥库，可以配置成绝对路径，本例中是在Tomcat目录中创建了一个名为key的文件夹，仅供参考。

- keystorePass：密钥库生成时的密码

- truststoreFile：受信任密钥库，和密钥库相同即可

- truststorePass：受信任密钥库密码

**8.Android App编写BKS读取创建证书自定义的SSLSocketFactory**

private final static String CLIENT\_PRI\_KEY = "client.bks";private final static String TRUSTSTORE\_PUB\_KEY = "truststore.bks";private final static String CLIENT\_BKS\_PASSWORD = "123456";private final static String TRUSTSTORE\_BKS\_PASSWORD = "123456";private final static String KEYSTORE\_TYPE = "BKS";private final static String PROTOCOL\_TYPE = "TLS";private final static String CERTIFICATE\_FORMAT = "X509";

　public static SSLSocketFactory getSSLCertifcation(Context context) {

SSLSocketFactory sslSocketFactory = null;

try {

// 服务器端需要验证的客户端证书，其实就是客户端的keystore

KeyStore keyStore = KeyStore.getInstance(KEYSTORE\_TYPE);// 客户端信任的服务器端证书

KeyStore trustStore = KeyStore.getInstance(KEYSTORE\_TYPE);//读取证书

InputStream ksIn = context.getAssets().open(CLIENT\_PRI\_KEY);

InputStream tsIn = context.getAssets().open(TRUSTSTORE\_PUB\_KEY);//加载证书

keyStore.load(ksIn, CLIENT\_BKS\_PASSWORD.toCharArray());

trustStore.load(tsIn, TRUSTSTORE\_BKS\_PASSWORD.toCharArray());

ksIn.close();

tsIn.close();

//初始化SSLContext

SSLContext sslContext = SSLContext.getInstance(PROTOCOL\_TYPE);

TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance(CERTIFICATE\_FORMAT);

KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance(CERTIFICATE\_FORMAT);

trustManagerFactory.init(trustStore);

keyManagerFactory.init(keyStore, CLIENT\_BKS\_PASSWORD.toCharArray());

sslContext.init(keyManagerFactory.getKeyManagers(), trustManagerFactory.getTrustManagers(), null);

sslSocketFactory = sslContext.getSocketFactory();

} catch (KeyStoreException e) {...}//省略各种异常处理，请自行添加

return sslSocketFactory;

}

**9.Android App获取SSLFactory实例进行网络访问**

private void fetchData() {

OkHttpClient okHttpClient = new OkHttpClient.Builder()

.sslSocketFactory(SSLHelper.getSSLCertifcation(context))//获取SSLSocketFactory

.hostnameVerifier(new UnSafeHostnameVerifier())//添加hostName验证器

.build();

Retrofit retrofit = new Retrofit.Builder()

.baseUrl("https://10.2.8.56:8443")//填写自己服务器IP

.addConverterFactory(GsonConverterFactory.create())//添加 json 转换器

.addCallAdapterFactory(RxJavaCallAdapterFactory.create())//添加 RxJava 适配器

.client(okHttpClient)

.build();

IUser userIntf = retrofit.create(IUser.class);

userIntf.getUser(user.getPhone())

.subscribeOn(Schedulers.io())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(new Subscriber<UserBean>() {

//省略onCompleted、onError、onNext

}

});

}private class UnSafeHostnameVerifier implements HostnameVerifier {

@Override

public boolean verify(String hostname, SSLSession session) {

return true;//自行添加判断逻辑，true->Safe，false->unsafe

}

}

## 结束语

由于双向认证涉及的原理知识太多，有些地方我也是一笔带过，本文想着重介绍证书的制作以及应用。在此奉劝各位，如果不了解RSA非对称加密，对称加密以及HASH校验算法 的同学，最好还是先看书学习一下。  
了解原理对于进步来说是十分有帮助的，网上的资料鱼龙混杂，不了解原理的话你根本无从分辨网上文章的正误。  
(不过我这篇文章绝对是正确的双向认证，大家可以放心)