# 加密传输数据

## 前置知识

客户端要与服务端进行交互，这就涉及到客户端和服务端之前的数据传输。而由于通过网络传递的数据可以通过网络抓包获取到，如果这种网络中传输的数据没有加密处理，就可能面临数据泄漏的问题。

**威胁描述：**

移动应用经常会在各种公共场所使用网络，又由于这样的公共网络往往都是不安全的，如果用户在这样的网络中进行像登录这样的敏感操作，而网络中又存在攻击者进行中间人攻击的话，那么用户的登录账号密码等信息都会被泄漏，这就是很严重的安全问题。

**涉及功能点：**

移动APP

## 修复方案

1. 使用双向认证的方式加密传输数据
2. 在服务端生成服务端证书库和客户端证书库，证书使用RSA2048加密算法，使用SHA256哈希算法
3. 分别从客户端证书库中导出客户端证书、从服务端证书库中导出服务端证书
4. 将客户端证书导入服务端证书库中，这是服务器信任的客户端证书；再将服务端证书导入客户端证书库中，这是客户端信任的服务端证书
5. 将client.jks和truststore.jks分别转换成client.bks和truststore.bks,然后放到客户端的资源目录下
6. 在服务器上配置证书和证书库，设置使用双向证书认证。另外还应设置使用TLS1.2以上版本协议；仅使用安全的加密算法；启用前向加密技术；使用HSTS策略；优先采取服务器算法。
7. 这样服务端持有客户的公钥证书；同时客户端持有服务端的公钥证书
8. 在建立加密连接时，服务端发送其公钥证书到客户端，客户端将此证书和其保存的服务端公钥证书进行比较，若不一致则建立加密连接失败
9. 然后，客户端发送其公钥证书给服务端，服务端也将其保存的客户端证书与收到的公钥证书进行比较，若不一致则建立加密连接失败

## 实施参考

1. 可以参考以下生成密钥库和证书

|  |
| --- |
| //生成服务器证书库  keytool -genkey -validity 365 -alias server -keyalg RSA -keysize 2048 -sigalg SHA256 -keystore server.keystore -storepass 123456 -keypass 123456  //生成客户端证书库  keytool -genkey -validity 365 -alias client -keyalg RSA -keysize 2048 -sigalg SHA256 -storetype PKCS12 -keystore client.jks -storepass 123456 -keypass 123456  //从客户端证书库中导出客户端证书  keytool -export -v -alias client -keystore client.jks -storepass 123456 -rfc -file client.cer  //从服务器证书库中导出服务器证书  keytool -export -v -alias server -keystore server.keystore -storepass 123456 -rfc -file server.cer  //将服务端证书导入客户端证书库  keytool -import -v -alias server -file server.cer -keystore truststore.jks -storepass 123456 -storetype BKS  //将客户端证书导入服务端证书库中  keytool -import -v -alias client -file client.cer -keystore server.keystore -storepass 123456 |

1. 可参考以下Android代码的示例

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  **return** sslSocketFactory**;**  **}** |

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://ipaddress: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  **}**  **}** |