# SDK加密方案分析

SDK中主要有两种交互，第一种是验证开发者，第二种是验证完成后的正常通信（这里称为通信），下面就针对验证开发者中的请求、响应报文和通信的请求、响应报文做分析。

## 加密规则

### 验证开发者请求报文加密规则

type: approveDev

rsaStr: Base64(RSA(randomKey, publicKeyS))

head: Base64(AES256((

appkey: appkey

RandomNum: randomNum

DevId: mac

DevName: hostName

randomWorkKey: randomWorkKey

sign: Base64(MD5withRSA(SHA-256(appId+random+mac+hostName), privateKeyB))

), randomKey))

### 验证开发者响应报文加密规则

head: Base64(AES256((

token: token

sign: Base64(MD5withRSA(token+randomNum, privateKeyS))

), randomKey))

### 通信请求报文加密规则

type: JavaAPISecurityControlSend

rsaStr: Base64(RSA(randomKey, publicKeyS))

head: Base64(AES256((

appkey: appkey

token: token

sign: Base64(MD5withRSA(reqData+token, privateKeyB))

), randomKey))

data: Base64(AES256(

data: reqData

), randomKey))

### 通信响应报文加密规则

head: Base64(AES256((

sign: Base64(MD5withRSA(respData, privateKeyS))

), randomKey))

data: Base64(AES256(

data: respData

), randomKey))

## 密码套件

从报文中可以看出来SDK的加密方案中用到了多种算法。其中转码算法有Base64，加密算法有RSA、AES，散列算法有SHA，签名算法有MD5withRSA。下面结合项目简单介绍一下这些算法在该方案中的应用。

### Base64

该算法是一种转码算法，主要作用是将信息（其中有一部分无法显示的字符）转换为可以

显示的字符，防止类似于回车、反斜线在路由器处理时导致错误，即方便在信息网络中传输。在加密处理后、传输前使用。

### RSA

该算法是一种非对称算法。在该方案中采用1024位密钥（目前700+密钥的RSA算法已经可以

被破解，但1024位密钥还是比较安全的），在传递共享密钥时使用。

### AES

该算法是一种对称算法。该方案中采用256位密钥，分组模式为ECB，补码模式PKCS7Padding。在加密head和data部分信息时使用。

### SHA

该算法是一种散列算法。在该方案中采用的SHA256是SHA-2中的标准算法（如今SHA-1中的标准已经被废弃了，并且最新的标准是SHA-3），散列算法主要作用是生成与某段信息唯一对应的固定长度的标志（这里可以称之为指纹或消息摘要），而SHA256生成的标志就是256位的。在验证开发者签名时使用。

### MD5withRSA

该算法是一种签名算法，其中的MD5也是散列算法。而该签名算法就是把签名信息用MD5提取消息摘要后在使用RSA签名。

## 分析与疑问

在网络安全中，安全方案最主要的三个任务便是防伪装、防窜改、防窃听；而该安全方案是在HTTPS的基础下实施的，因此在协议层的安全已经满足了要求，下面就从这三个方面在业务层的情况进行分析，并且记录一些我对该安全方案的疑问。

### 防伪装

所谓的伪装就是指在网络传输过程当中需要交换双方公钥时，通信第三方采用中间人攻击，将报文拦截下来，并且将自己的公钥发送给请求方和响应方，从而充当通信双方的中间人对双反的通信内容进行修改。要做防伪装有两种做法，第一公钥不在网络上传输，第二有一种机制（数字证书）可以保证该公钥确实是通信对方的公钥。

**分析：**

在该方案中，RSA所用的公私钥都是内置在SDKClient和SDKServer端的，因此在通信过程当中并不需要传输公钥，满足了第一种情况，因此并不会出现伪装公钥的情况，在业务层具有防伪装的功能。

### 防窜改

窜改就是指在通信过程当中，通信的第三方通过拦截报文并修改报文，窜改报文。

**分析：**

在该方案中，采用了数字签名的方法确保了报文不会被修改。主要操作就是发送信息方用自己的私钥对需要防窜改的信息进行签名（本质就是RSA加密），并且接收方以发送方的公钥进行验签（本质就是RSA解密后与报文内容比较），在防伪装的前提下，数字签名可以有效的防窜改。

**疑问：**

1、防窜改的方法还有消息认证码（本质是带密钥的散列函数），在性能上消息认证码比数字签名快很多；但在安全方面考虑，消息认证码不可以做到防否认（消息认证码的生成和校验都是使用同一个密钥，那么就无法确认是谁生成的消息认证码，也就不能防止出现问题后其中一方否认使其生成的消息认证码，而推卸责任），而数字签名可以。但是我认为在本项目中并不需要用到防否认，因为通信双方都是本公司产品，那么为什么不用消息认证码代替数字签名以提高性能呢？

2、在数字签名中签名内容只在验证开发者请求流程中进行了SHA-256散列处理，而其它流程并没有进行SHA-256散列处理。另外签名算法已经使用了MD5withRSA，其中的MD5算法已经进行了散列处理，也就是说在形式上压缩了RSA加密的内容，减少了RSA加密的耗时（RSA加密耗时远比散列算法耗时长），也就减少了整体签名的耗时。那么为什么不去掉验证开发者请求流程时的SHA-256散列处理这个步骤，以提高性能呢？

### 防窃听

窃听指的就是在网络传输过程当中，通信第三方通过监听报文，来查看通信双方的通讯内容，而解决这个问题的方法无疑就是加密。

**分析：**

加密算法从密钥的类型可以分为对称加密和非对称加密（公钥加密）。在同一加密内容的情况下，对称加密算法（主要为异或运算）耗时比非对称加密（主要为指数运算和模运算，并且是大指数）耗时1/1000之下。

对于通信主体内容，在悬殊的性能差异下，通信主体内容必定要用对称加密算法进行加密。

那么涉及到对称加密，就需要考虑共享密钥的问题。首先，如果共享密钥事先商定在通信双方，那么无疑增大了密钥泄露的风险，而且一旦泄露对以后的通信都会造成影响，该方案否定；其次，可以通过在通信前商定本次会话的随机共享密钥，那么即使密钥泄露也不会对以后的会话造成影响，采取该方案。

那么为了保证共享密钥在通信过程当中不被窃听到，就要对共享密钥进行加密，若此时依然采用对称加密，那么这个问题就永无休止了，因此应该采用非对称加密，而对于通信主体来说共享密钥的长度应该是比较短的，因此耗时上还是可以接受的。

总的来说本方案中用到耗时较短的对称加密加密较长的通信主体内容，用耗时较长的非对称加密机密较短的共享密钥，可以说是扬长避短。

另外， 在一次会话的每一次通信中都使用的新的随机密钥作为共享密钥，又进一步的缩小了密钥泄露的辐射范围。