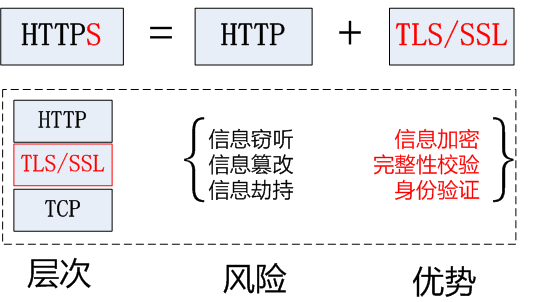
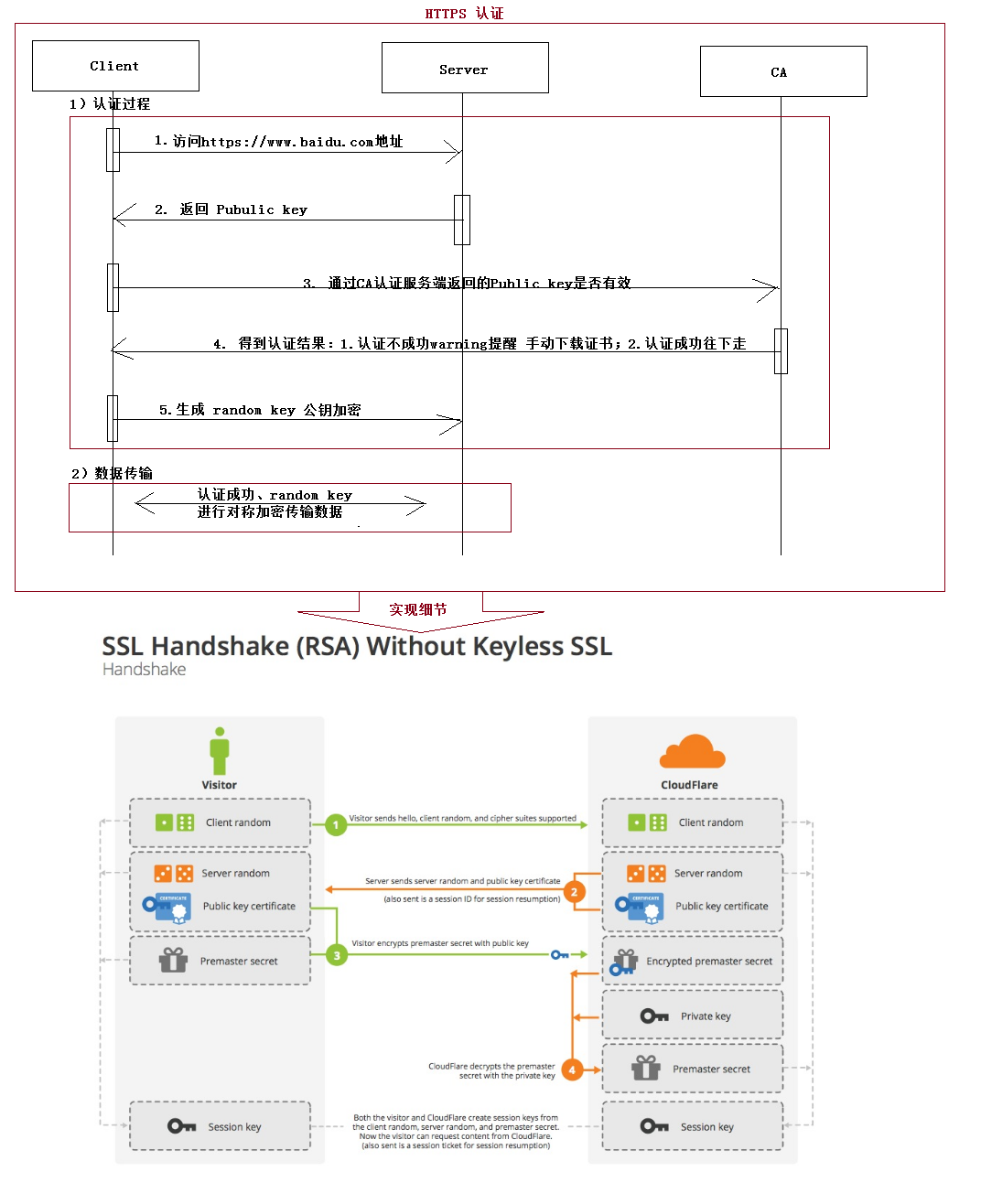
# 接口设计方案

一、采用https传输协议

HTTPS（全称：Hyper Text Transfer Protocol over Secure Socket Layer），是以安全为目标的HTTP通道，简单讲是HTTP的安全版。即HTTP下加入SSL层，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL，它是一个URI scheme（抽象标识符体系），句法类同http:体系。用于安全的HTTP数据传输。

参考地址: <http://www.admin5.com/article/20151231/640434.shtml>





1、https 与http对比

A. 数据传输安全

HTTP协议以明文方式发送内容，不提供任何方式的数据加密，如果攻击者截取了Web浏览器和网站服务器之间的传输报文，就可以直接读懂其中的信息。HTTPS在HTTP的基础上加入了SSL协议，SSL依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。服务端和客户端之间的所有通讯，都是加密的。接下来所有的信息往来就都是加密的。第三方即使截获，也没有任何意义，因为他没有密钥，当然篡改也就没有什么意义了。

B. 端口不同

http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443

C. https协议需要到ca申请证书,并在服务器上面进行证书的安装

D.抓包 HTTP 通信，能够清晰的看到通信的头部和信息的明文，但是 HTTPS 是加密通信，无法看到 HTTP 协议的相关头部和数据的明文信息。

2、HTTPS到底安全吗？

这个答案是肯定的，谷歌公司已经行动起来要大力推广HTTPS的使用，目前主流的大型网站，例如百度、淘宝、谷歌等网站已经逐渐替换为https协议。

3、HTTPS的简单工作原理  
 HTTPS在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。TLS/SSL协议不仅仅是一套加密传输的协议，更是一件经过艺术家精心设计的艺术品，TLS/SSL中使用了非对称加密，对称加密以及HASH算法。

4、https服务配置

nginx目前的可以支持https，流程大概是申请证书，在服务器上面按照ssl证书，nginx配置按照ssl加密模块，修改nginx的配置文件，新增443端口跳转服务器地址。

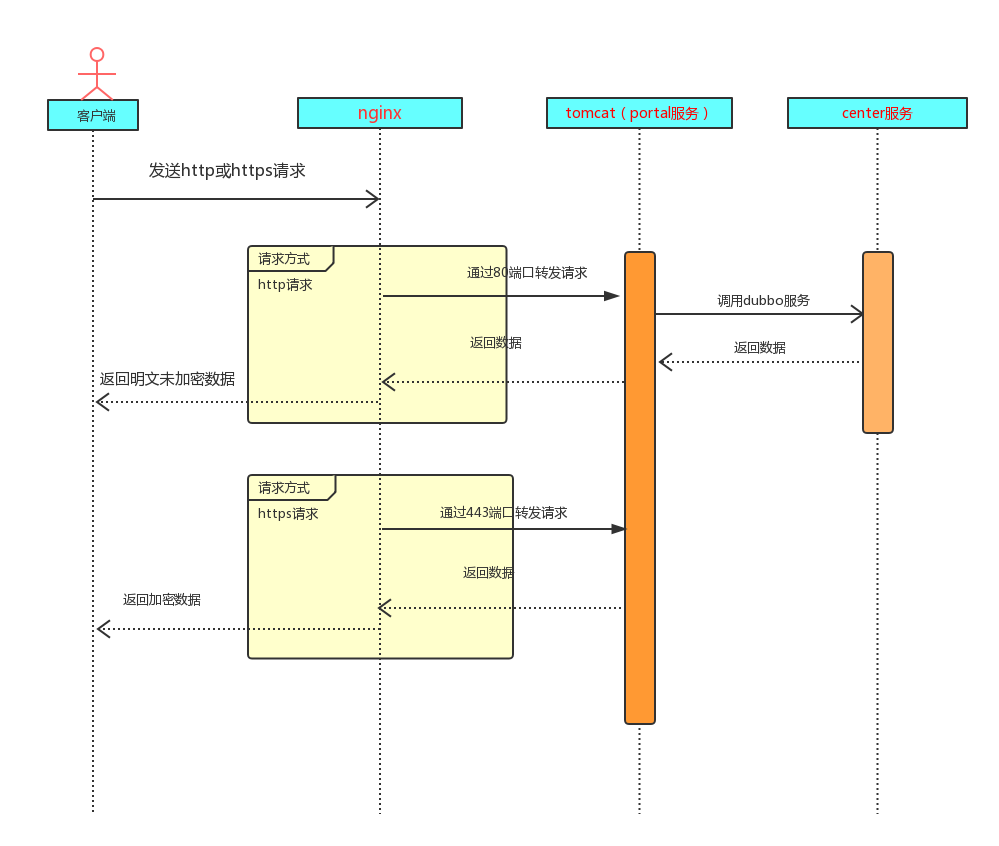
目前的云平台在所有的nginx服务器上面已经配置了证书和ssl模块，我们所要做的就是需要运维人员在nginx配置文件中信息443端口跳转到了服务器的地址，其它配置信息都不用动，目前服务器已经支持了https的加密处理。

还是做一下说明：

a. 只能支持 \*.tclclouds.com ，也就是 最高支持二级域名，三级域名以及以上是不行的

b. 域名中有 - 或 \_ 分割的，最好以后申请域名的时候不要这样命名，可能会导致一些问题

c. http的80端口和https的443端口是可以并存的，旧的http保持原来的跳转不变，新的https采用443的服务跳转。



5、服务端改动

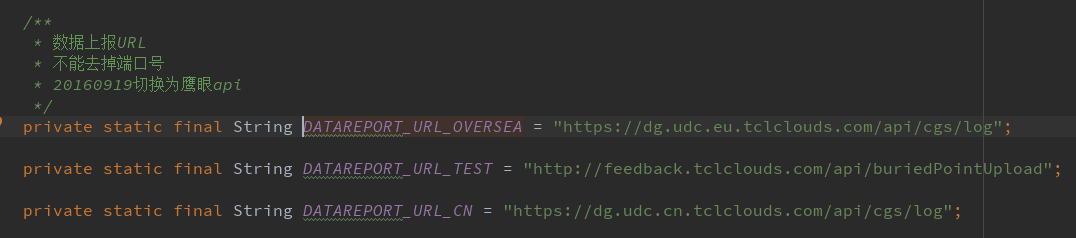
目前接口域名信息如下：

| **环境信息** | **地址** |
| --- | --- |
| 测试环境 | [http://cleanportal-test.tclclouds.com](http://cleanportal-test.tclclouds.com//function/recommend?channelNumber=100001&innerVersion=1) |
| 国内线上 | [http://cleanportal-cn.tclclouds.com](http://cleanportal-cn.tclclouds.com/) |
| 海外线上 | [http://cleanportal.tclclouds.com](http://cleanportal.tclclouds.com/) |

需要申请运维人员在以上域名的nginx的服务器，修改配置信息，新增443跳转即可

6、客户端改动

例如埋点日志，只需要客户端将之前的http域名换成https即可



二、数据加密

1、常见加密算法

a. BASE64 普通加密，没有密钥，java自身提供了相关的加密、解密实现，只要对方知道你是BASE64加密就可以获取数据，没有密钥的使用，数据相对不安全。

b. DES 对称加密，不推荐使用，加密相对AES简单很多，易破解

c. AES 对称加密，推荐使用，破解不容易，相对速度快，有密钥机制保证安全

d. RSA 非对称加密，不推荐使用，因为运算时间成本高，速度比较慢，对接口耗时有一点的影响

e. MD5 哈希算法，常用于文件校验、用户校验等，不可逆，一般不用于接口加密

2、数据加密等级划分

a.数据重要性--高 ★★★★★

b.数据重要性--中 ★★★

c.数据重要性--低 ★

3、常见认证方式

a. OAuth 2.0, OAuth在"客户端"与"服务提供商"之间，设置了一个授权层（authorization layer）。"客户端"不能直接登录"服务提供商"，只能登录授权层，以此将用户与客户端区分开来。"客户端"登录授权层所用的令牌（token），与用户的密码不同。用户可以在登录的时候，指定授权层令牌的权限范围和有效期。

OAuth 2.0适用于开发平台，如新浪微博、微信、谷歌等，基于用户系统，可以做到第三方登录，授权第三方应用访问用户的资源文件，不适用与应用内部的接口设计。

b. token校验，token可以是服务端给调用者的秘钥，也可以是别的内容，目前的云平台以及TCL邮件系统，就是采用这样访问，clientk客户端调用服务端之前申请一个秘钥，每次请求携带秘钥，服务端校验是否存在授权，存在才会处理，不然不处理。

4、接口加密方案

在用户可以使用REST接口之前，首先需要通过向REST接口服务端申请，当获准后会收到两个key：accessKey和secretKey。其中 accessKey相当于用户标识，应用会通过它区分不同用户，而secreKey相当于提供给用户的密码，在接口使用过程中都不会在网络中传输，只有用户和服务端知道。

A. 查询接口

在请求的头信息，新增token字段,token值约定一定的生成规则，请求到服务端，简单的参数校验是否合格，校验通过才会返回数据。

查询数据相关接口的token定义简单点，就是给客户端的标识即可，相对来说查询更频繁，加密请求加密太复杂，会导致接口数据过慢。

 Java Code

|  |  |  |
| --- | --- | --- |
| 1 |  | String token =  "http请求时间戳" +":" +accessKey  ; |

对返回的数据进行加密方式如下，避免第三方获取到数据的内容

1). 采用现有的BASE64加密返回数据，以及进行Gzip压缩

2). 采用AES加密数据，服务端和客户端公用一个秘钥，加密数据进行Gzip压缩返回

3). 两层加密，第一层采用AES加密数据，再将加密后的数据进行异或加密返回

B. 新增、更新、删除操作接口

更新数据操作，例如注册、转账等操作，涉及到数据库里面的数据安全性的，我们要对接口调用做一定的校验，避免第三方恶意调用

1). 结合RESTful接口，我们可以采用接口签名认证的方式实现

 Java Code

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 |  | String accessKey = "服务端提供给客户端的用户标识，区分不同的调用方"; String requestStr = "http请求的动作(post/get)" + "," +                     "http请求的MD5值" + "," +                     "http请求的类型" + "," +                     "http请求资源"; String Signature =  Aes. Encrypt(requestStr,secretKey); String token =  "http请求时间戳" +":" +accessKey  + ":" + Signature; |

上面的token的定义规则如下为，固定的前缀，加上给客户端的标识，再加上数字签名，数字签名包含调用的信息，再对签名进行AES加密，加密密钥是给客户端的另外一个密钥。

服务端验证方式则是根据传输过来的token解析出accessKey和Singature，比对HTTP头中的时间戳，比对服务器时间，如果超过某个阈值，则拒绝访问，同时返回请校准你的应用时间。再根据accessKey得到本地保存的对应secretKey(注意其并未在网络中传输)，然后再重新根据客户端Signature生成方式重新生成与解析的进行比较，相同则认证成功结束，否则， 直接返回错误信息给客户端。

三、现有接口以及改动

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 业务名称 | 接口描叙 | 数据重要性 | 旧接口 | 新接口 | 数据传输协议 | 数据返回加密 | 备注 |
| 目录收集与查询 | 保存一级、二级目录 | ★ | /appDirectory/save |  | https | Base64+GZIP | **之前已加密，其它部门已嵌入该接口，不能改动** |
| 分页查找一级目录名称集合 | ★ | /appDirectory/findDirByPage |  | https | 明文 |  |
| 分页查找二级目录名称集合 | ★ | /appDirectory/findSubDirByPage |  | https | 明文 |  |
| 分页查找一二级目录 | ★ | /appDirectory/findByPage |  | https | 明文 |  |
| 包名收集与查询 | 保存包名列表 | ★ | /appPackage/save |  | https | Base64+GZIP | **之前已加密，其它部门已嵌入该接口，不能改动** |
| 分页查找包名 | ★ | /appPackage/findByPage |  | https | 明文 |  |
| 版本更新 | 检查版本更新 | ★ | /version/update |  | https | 明文 | **TODO: 需要加密** |
| 权限按钮 | 查询授权按钮状态 |  | /authorizeButton/find |  | https |  | **已不再调用** |
| 云端控制 | 查询云端控制 | ★★ | /cloudControl/query |  | https | 明文 | **TODO: 需要加密** |
| 颜色按钮控制 | 查询颜色变化范围 |  | /colorChange/find |  | https |  | **已不再调用** |
| 根据路径查询残留 | 根据路径查询残留 | ★★★★★ | /directoryResidual/query |  | https | AES首层加密+ CRC校验 | **已加密，无须改动** |
| 显示时长 | 显示时长控制 |  | /displayTime/find |  | https |  | **已不再调用** |
| 节日活动接口 | 节日活动接口 | ★★ | /festival/query |  | https | Base64+GZIP | **已加密** |
| 功能推荐 | 功能推荐 | ★★ | /function/recommend |  | https | 明文 | **TODO: 需要加密** |
| 高频库更新 | 高频库更新 | ★★★ | /libraryUpdate/query |  | https | 明文 | **TODO: 需要加密** |
| 通知栏控制 | 通知栏控制 | ★ | /noticeBar/find |  | https | 明文 | 已走新接口，旧接口调用越来越少，不用改动 |
| 弹窗参数控制 | 弹窗参数控制 |  | /popupWindow/findByType |  | https |  | **已不再调用** |
| 按包查询残留 | ​按包查询残留 | ★★★★★ | /packageResidual/query |  | https | AES首层加密+ CRC校验 | **已加密，无须改动** |
| 按包查询缓存 | 按包查询缓存 | ★★★★★ | /packageRefer/cache |  | https | AES首层加密+ CRC校验 | **已加密，无须改动** |
| 用户协议接口 | 用户协议接口 | ★ | /userAgreement/query |  | https | 明文 | **之前未加密，其它部门已嵌入该接口，不能改动** |

四、接口命名规则

1、数据操作命名规范

|  |  |
| --- | --- |
| 数据操作 | 命名规则 |
| 新增 | add |
| 更新 | update |
| 删除 | delete |
| 查询 | query |

2、接口命名规则

**/api/业务名称/数据操作**

**如下面的例子，那么接口名称如下**

|  |
| --- |
| **https://localhost:8080/api/user/add**  **https://localhost:8080/api/user/update**  **https://localhost:8080/api/user/delete**  **https://localhost:8080/api/user/query** |

 Java Code

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 |  | @Controller @RequestMapping("/api/user") public class UserController{     //添加用户     @RequestMapping("/add")     public void add(User user){     }     //更新用户     @RequestMapping("/update")     public void update(User user){     }     //删除用户     @RequestMapping("/delete")     public void delete(int userId){     }     //查询用户     @RequestMapping("/query")     public void query(int userId){     } } |