**开放API接口签名验证**

### 接口安全问题

* 请求身份是否合法？
* 请求参数是否被篡改？
* 请求是否唯一？

### AccessKey&SecretKey （开放平台）

#### 请求身份

为开发者分配**AccessKey**（开发者标识，确保唯一）和**SecretKey**（用于接口加密，确保不易被穷举，生成算法不易被猜测）。

#### 防止篡改

**参数签名**

1. 按照请求参数名的字母升序排列非空请求参数（包含AccessKey），使用URL键值对的格式（即key1=value1&key2=value2…）拼接成字符串stringA；
2. 在stringA最后拼接上Secretkey得到字符串stringSignTemp；
3. 对stringSignTemp进行MD5运算，并将得到的字符串所有字符转换为大写，得到sign值。

请求携带参数**AccessKey**和**Sign**，只有拥有合法的身份AccessKey和正确的签名Sign才能放行。这样就解决了身份验证和参数篡改问题，即使请求参数被劫持，由于获取不到SecretKey（**仅作本地加密使用，不参与网络传输**），无法伪造合法的请求。

#### 重放攻击

虽然解决了请求参数被篡改的隐患，但是还存在着重复使用请求参数伪造二次请求的隐患。

**timestamp+nonce方案**

nonce指**唯一的随机字符串**，用来标识每个被签名的请求。通过为每个请求提供一个唯一的标识符，服务器能够防止请求被多次使用（记录所有用过的nonce以阻止它们被二次使用）。

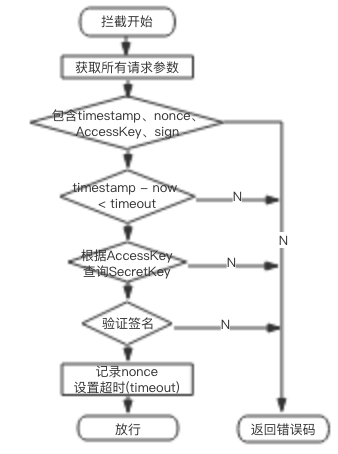
然而，对服务器来说永久存储所有接收到的nonce的代价是非常大的。可以使用**timestamp来优化nonce的存储**。

假设允许客户端和服务端最多能存在15分钟的时间差，同时追踪记录在服务端的nonce集合。当有新的请求进入时，首先检查携带的timestamp是否在15分钟内，如超出时间范围，则拒绝，然后查询携带的nonce，如存在已有集合，则拒绝。否则，记录该nonce，并删除集合内时间戳大于15分钟的nonce（可以使用redis的expire，新增nonce的同时设置它的超时失效时间为15分钟）。

#### 实现

请求接口：http://api.test.com/test?name=hello&home=world&work=java

* 客户端
  1. 生成当前时间戳timestamp=now和唯一随机字符串nonce=random
  2. 按照请求参数名的字母升序排列非空请求参数（包含AccessKey)  
     stringA="AccessKey=access&home=world&name=hello&work=java&timestamp=now&nonce=random";
  3. 拼接密钥SecretKey  
     stringSignTemp="AccessKey=access&home=world&name=hello&work=java&timestamp=now&nonce=random&SecretKey=secret";
  4. MD5并转换为大写  
     sign=MD5(stringSignTemp).toUpperCase();
  5. 最终请求  
     http://api.test.com/test?name=hello&home=world&work=java&timestamp=now&nonce=nonce&sign=sign;
* 服务端



### Token&AppKey（APP）

在APP开放API接口的设计中，由于大多数接口涉及到用户的个人信息以及产品的敏感数据，所以要对这些接口进行身份验证，为了安全起见让用户暴露的明文密码次数越少越好，然而客户端与服务器的交互在请求之间是无状态的，也就是说，当涉及到用户状态时，每次请求都要带上身份验证信息。

#### Token身份验证

1. 用户登录向服务器提供认证信息（如账号和密码），服务器验证成功后**返回Token**给客户端；
2. 客户端将Token保存在本地，后续发起请求时，**携带此Token**；
3. 服务器检查Token的有效性，有效则放行，无效（Token错误或过期）则拒绝。

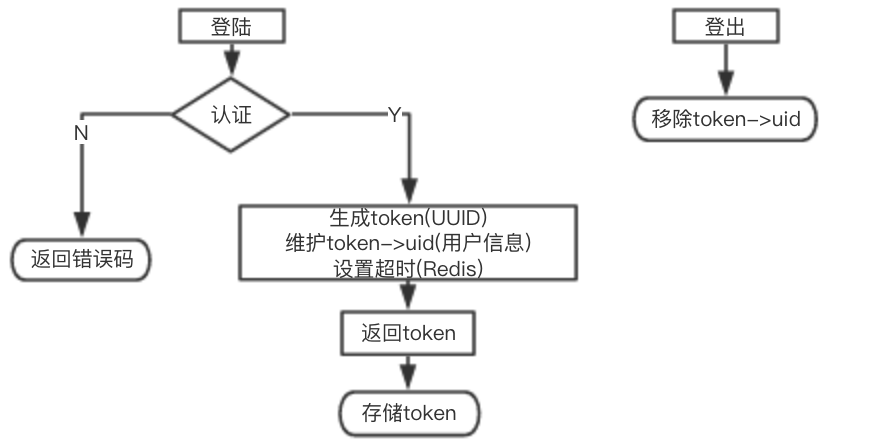
**安全隐患**：Token被劫持，伪造请求和篡改参数。

#### Token+AppKey签名验证

与上面开发平台的验证方式类似，为客户端分配**AppKey**（密钥，用于接口加密，不参与传输），将AppKey和所有请求参数组合成源串，根据**签名算法**生成签名值，发送请求时将签名值一起发送给服务器验证。这样，即使Token被劫持，对方不知道AppKey和签名算法，就无法伪造请求和篡改参数。再结合上述的**重发攻击**解决方案，即使请求参数被劫持也无法伪造二次重复请求。

#### 实现

##### 登陆和登出请求



##### 后续请求

* 客户端  
  和上述开放平台的客户端行为类似，把AccessKey改为token即可。
* 服务端

