保证接口安全一般分为两种，一个是防数据篡改，一个是防数据泄漏，防篡改使用摘要验证，防泄漏使用加密

## token

token=5位随机数+时间戳

aesEnctrypt（token）

（1）验证时间和服务器时间不能超过3分。

（2）同一个时间戳，随机数只能使用一次。

## 对称加密

把参数对称加密 aes

## 签名验证

ras

调用方，要提供公钥，sign（通过双方定义好的[算法](http://lib.csdn.net/base/datastructure)把摘要和参数计算后的值）

我们把post过来的参数先解密，通过制定的算法算出sign

我们看我们算出sign与调用方传过来的sign是否一致，一致则可以进行业务处理，不一致返回签名失败。

sign算法

secretKey是双方定义的摘要

params是参数的hashmap集合

byte[] data=Digest.getDigest(secretKey, params);  
  
public class Digest {  
  
 public static byte[] getDigest(String secretKey, Map<String,String> params) throws Exception{  
  
 Set<String> keySet = params.keySet();  
  
 TreeSet<String> sortSet = new TreeSet<>();  
  
 sortSet.addAll(keySet);  
  
 String keyValueStr = "";  
  
 Iterator<String> it = sortSet.iterator();  
  
 while(it.hasNext()){  
  
 String key = it.next();  
  
 String value = params.get(key);  
  
 keyValueStr += key + value;  
  
 }  
  
 keyValueStr = keyValueStr + secretKey;  
  
 MessageDigest md = MessageDigest.getInstance("SHA-1");  
  
 return md.digest(keyValueStr.getBytes("utf-8"));  
  
 }  
  
}  
  
boolean b=RsaUtil.verify(data, sign, publicKeyString);

b=false 返回签名失败。

以下是rasutil

/\*\*  
  
 \*   
  
 \*/  
  
package com.hlmedicals.app.util;  
  
import Java.io.UnsupportedEncodingException;  
  
import java.security.KeyFactory;  
  
import java.security.PrivateKey;  
  
import java.security.PublicKey;  
  
import java.security.spec.PKCS8EncodedKeySpec;  
  
import java.security.spec.X509EncodedKeySpec;  
  
import com.itextpdf.xmp.impl.Base64;  
  
/\*\*  
  
 \* @author dell  
  
 \*  
  
 \*/  
  
public class RsaUtil {  
  
 public static void main (String [] args){  
  
 try {  
  
 byte[] privateKeyString= IConst.privateKeyString.getBytes("utf-8");  
  
 byte[] publicKeyString= IConst.publicKeyString.getBytes("utf-8");  
  
 String data1 = "testabc";   
  
 //siyao签名  
  
 byte[] data=data1.getBytes("utf-8");  
  
 byte[] s = sign(data, privateKeyString);  
  
 boolean b=verify(data,s,publicKeyString);  
  
 System.out.println(b);  
  
 } catch (Exception e) {  
  
 // TODO Auto-generated catch block  
  
 e.printStackTrace();  
  
 }   
  
 }  
  
 /\*\*   
  
 \* 使用私钥对数据进行加密签名   
  
 \* @param data 数据   
  
 \* @param privateKeyString 私钥   
  
 \* @return 加密后的签名   
  
 \*/   
  
 public static byte[] sign(byte[] data, byte[] privateKeyString) throws Exception {   
  
 KeyFactory keyf = KeyFactory.getInstance("RSA");   
  
 PrivateKey privateKey = keyf.generatePrivate(new PKCS8EncodedKeySpec(Base64.decode(privateKeyString)));   
  
 java.security.Signature signet = java.security.Signature.getInstance("SHA1withRSA");   
  
 signet.initSign(privateKey);   
  
 signet.update(data);   
  
 byte[] signed = signet.sign();   
  
 return Base64.encode(signed);   
  
 }   
  
 /\*\*   
  
 \* 使用公钥判断签名是否与数据匹配   
  
 \* @param data 数据   
  
 \* @param sign 签名   
  
 \* @param publicKeyString 公钥   
  
 \* @return 是否篡改了数据   
  
 \*/   
  
 public static boolean verify(byte[] data, byte[] sign, byte[] publicKeyString) throws Exception {   
  
 KeyFactory keyf = KeyFactory.getInstance("RSA");   
  
 PublicKey publicKey = keyf.generatePublic(new X509EncodedKeySpec(Base64.decode(publicKeyString)));   
  
 java.security.Signature signet = java.security.Signature.getInstance("SHA1withRSA");   
  
 signet.initVerify(publicKey);   
  
 signet.update(data);   
  
 return signet.verify(Base64.decode(sign));   
  
 }   
  
}

4.http 转成htts

5.后台登录要有验证码，没有验证码 ，系统会被爆破，验证码不能被多次使用。

我的项目登录成功后把验证码存在session里。要把session验证码设定为空就可以。

6.系统上传文件时候，应该上传白名单的文件类型，防止jsp、jspx在系统中执行导致系统崩掉。

在开发过程中，肯定会有和第三方或者app端的接口调用。在调用的时候，如何来保证非法链接或者恶意攻击呢？

## 签名

根据用户名或者用户id，结合用户的ip或者设备号，生成一个token。在请求后台，后台获取http的head中的token，校验是否合法（和[数据库](http://lib.csdn.net/base/mysql)或者[Redis](http://lib.csdn.net/base/redis)中记录的是否一致，在登录或者初始化的时候，存入数据库/redis）

在使用Base64方式的编码后，Token字符串还是有20多位，有的时候还是嫌它长了。由于GUID本身就有128bit，在要求有良好的可读性的前提下，很难进一步改进了。那我们如何产生更短的字符串呢？还有一种方式就是较少Token的长度，不用GUID，而采用一定长度的随机数，例如64bit，再用Base64编码表示：

var rnd = new Random();  
  
 var tokenData = userIp+userId;  
  
 rnd.NextBytes(tokenData);  
  
 var token = Convert.ToBase64String(tokenData).TrimEnd('=');

由于这里只用了64bit，此时得到的字符串为Onh0h95n7nw的形式，长度要短一半。这样就方便携带多了。但是这种方式是没有唯一性保证的。不过用来作为身份认证的方式还是可以的（如网盘的提取码）。

## 加密

客户端和服务器都保存一个秘钥，每次传输都加密，服务端根据秘钥解密。

客户端：

​ 1、设置一个key（和服务器端相同）

​ 2、根据上述key对请求进行某种加密（加密必须是可逆的，以便服务器端解密）

​ 3、发送请求给服务器

服务器端：

​ 1、设置一个key

​ 2、根据上述的key对请求进行解密（校验成功就是「信任」的客户端发来的数据，否则拒绝响应）

​ 3、处理业务逻辑并产生结果

​ 4、将结果反馈给客户端

## 第三方支持

比如[spring](http://lib.csdn.net/base/javaee) security－oauth

有兴趣的，可以参考这篇帖子

http://wwwcomy.iteye.com/blog/2230265

多数采用OAuth2，不过对于盗cookie没招，走HTTPS吧

api进行token校验或者签名，另外对于防刷，要进行限流

题主要看你如何定义接口安全了，我们可以看看新浪微博，每天无数的爬虫在爬，他的接口安全么？一般意义上安全指的是传输过程中不被盗取，走https基本可以做到，至于有人拿到api文档一类的东西（app被反向），那怎么样也防不了

所以如同楼上 1.对IP进行过滤，频繁访问的进行限制。 2.使用token令牌进行验证，通过后进行业务逻辑。 3.不对参数过渡暴露，传输的文本根据业务级别进行加密。

没有绝对的安全，题主可以看看微信开发者文档，看看他们的接口是怎么设计的

题主要看你如何定义接口安全了，我们可以看看新浪微博，每天无数的爬虫在爬，他的接口安全么？一般意义上安全指的是传输过程中不被盗取，走https基本可以做到，至于有人拿到api文档一类的东西（app被反向），那怎么样也防不了

所以如同楼上 1.对IP进行过滤，频繁访问的进行限制。 2.使用token令牌进行验证，通过后进行业务逻辑。 3.不对参数过渡暴露，传输的文本根据业务级别进行加密。

没有绝对的安全，题主可以看看微信开发者文档，看看他们的接口是怎么设计的