接口一旦对外暴露，会对后端服务器和数据安全造成威胁，给人搞事的机会。所以防止接口不轻易暴露极为重要，尤其是Android端安全性不高，容易反编译。

先简单讲下思路：有效防止接口暴露就是把接口存放在一个相对安全，必须通过校验才能获取到的空间。以下是我工作中使用过的方法：

1：把接口放在服务器，移动客户端使用公开接口，设置多个客户端参数如IMEI,型号，版本号等参数，根据参数按照一定规则生成SignParameter参数。该接口根据该参数来检验客户端是否合法，通过就返回其他接口。

优点：有效减少接口对外暴露，保护重要接口，接口能动态配置。

缺点：依赖公开接口，一旦公开接口有问题，就不能访问。用户体验不流畅，必须重复请求。

2：接口用NDK写，存放在本地动态库。检验apk签名，SHA256等信息生成密钥，检验密钥通过，就可以调用本地方法根据Key返回对应接口。以下代码：

int cPass = 0;

char \*pLocalSign = "localSign";

int check\_sign(char \*pSign) {

if (pSign == NULL) {

return Error;

}

if (strcmp(pCorrectSign, pSign) == 0) {

cPass = 1;

return 1;

} else {

return -1;

}

}

typedef struct {

const char \*pAPIKey;

const char url[256];

} sApi;

const char \*pError = "ERROR";

static sApi api\_table[] = {

{"apiKey1","/url/value1"},{"apiKey1","/url/value1"},{"apiKey1","/url/value1"},...

}

int get\_rest\_api(const char \*pKey, char \*pUrl) {

if (!cPass) {

strcpy(pUrl, pError);

return strlen(pUrl);

}

int nApiNum = sizeof(api\_table) / sizeof(api\_table[0]);

int n = 0;

for (n = 0; n < nApiNum; n++) {

if (strcmp(api\_table[n].pAPIKey, pKey) == 0) {

strcpy(pUrl, api\_table[n].url);

return strlen(pUrl);

}

}

return 0;

}

优点：本地代码不容易反编译，安全和性能相对较高。

缺点：接口不能动态配置。