



盛大游戏 Game Protect Kit SDK 使用手册 (2.0 版)



目 录

目 录	2
1. GPK 概述	4
1.1. GPK 的功能	4
1.2. GPK 技术特点	4
1.2.1. 内核技术	4
1.2.2. 虚拟机技术	4
1.2.3. 动态加解密技术	4
1.2.4. 智能监测技术	4
1.2.5. 外挂屏蔽技术	4
1.3. GPK 的优势	5
1.3.1. 反外挂能力	5
1.3.2. 反盗号能力	5
1.3.3. 安全性、独立性、灵活性、兼容性、稳定性	5
1.3.4. 商业合作模式	5
2. GPK 实施	5
2.1. 步骤	5
2.2. 实施时间	5
2.3. 集成 GPK 后的反外挂服务	6
3. GPK SDK 相关文件	6
3.1. 头文件	6
3.2. LIB 文件	6
4. GPK 模块客户端 SDK 开发指引	6
4.1. 步骤一：启动 GPK 并创建 IGPKCltDynCode 组件	6
4.2. 步骤二、设置动态加解密数据	7
4.3. 步骤三、加密解密数据	7
4.4. 步骤四、释放组件	8
5. GPK 模块服务端 SDK 开发指引	8
5.1. 步骤一、创建 IGPKSvrDynCode 组件	8
5.2. 步骤二、初始化动态加解密数据	8
5.3. 步骤三、为每个连接的客户端分配一个随机的索引值	9
5.4. 步骤四、取得索引指向的动态代码数据发往客户端	9
5.5. 步骤五、加密解密数据	9
5.6. 步骤六、释放 IGPKSvrDynCode 组件	10
5.7. 不重启服务端更改动态加解密库	10
6. GPK CSAuth 功能	10
6.1. 概述	10
6.2. 服务端 CSAuth 开发指引	10
6.2.1. 生成 IGPKCSAuth 对象	10
6.2.2. 载入 Auth 数据文件	11
6.2.3. 生成 CSAuth 数据	11
6.2.4. 检查 CSAuth 数据	11
6.2.5. 释放 IGPKCSAuth 对象	12



6.2.6.	不重启服务端更新 CSAuth 内容	12
6.3.	客户端 CSAuth 开发指引	12
6.3.1.	处理 CSAuth 事件	12
7.	GPK 模块的部署和作业流程	13
7.1.	升级服务器部署	13
7.2.	游戏服务端部署	13
7.3.	游戏客户端集成	13
7.4.	服务端和客户端作业流程	13
8.	集成 GPK 后的调试和机器人测试	15
8.1.	在调试器中调试程序	15
8.2.	机器人测试程序的编写	15
9.	GPK SDK 接口参考	15
9.1.	GPK 命名空间	15
9.2.	GPKStart 函数	15
9.3.	GPKCreateSvrDynCode 函数	16
9.4.	IGPKSvrDynCode 组件	16
9.4.1.	概述	16
9.4.2.	IGPKSvrDynCode::LoadBinary	16
9.4.3.	IGPKSvrDynCode::GetRandIdx	16
9.4.4.	IGPKSvrDynCode::GetCltDynCode	16
9.4.5.	IGPKSvrDynCode::Encode	17
9.4.6.	IGPKSvrDynCode::Decode	17
9.4.7.	IGPKSvrDynCode::Release	17
9.4.8.	IGPKSvrDynCode::LoadAuthFile	17
9.4.9.	IGPKSvrDynCode::AllocAuthObject	18
9.5.	IGPKCltDynCode 组件	18
9.5.1.	概述	18
9.5.2.	IGPKCltDynCode::SetDynCode	18
9.5.3.	IGPKCltDynCode::Encode	18
9.5.4.	IGPKCltDynCode::Decode	19
9.5.5.	IGPKCltDynCode::Release	19
9.5.6.	IGPKCltDynCode::ProcessAuth	19
9.6.	IGPKCSAuth 组件	20
9.6.1.	概述	20
9.6.2.	IGPKCSAuth::GetAuthData	20
9.6.3.	IGPKCSAuth::CheckAuthReply	20
9.6.4.	IGPKCSAuth::Release()	20



1. GPK 概述

1.1. GPK 的功能

GPK 是一款内核级的游戏保护工具。对目标游戏可以提供如下保护：

- 1、防范各类调试器和各类黑客工具，如 OllyDbg、RootKit 工具等。
- 2、防范键盘模拟类程序对游戏进行控制，如按键精灵等。
- 3、对游戏进程进行保护，防止内存读取和内存写入。
- 4、对游戏程序指定内存段进行 HASH 校验。防止修改。
- 5、对游戏封包进行动态算法加解密。游戏可以不必重新自行设计加密算法而直接使用 GPK 的动态算法。也可以在自己设计的算法上再次用 GPK 的动态算法加解密。
- 6、防止游戏内加速工具的使用。
- 7、防止 Ring0 和 Ring3 级的键盘记录，有效保护游戏帐号安全。
- 8、防木马，木马无法注入和修改游戏进程。
- 9、可以不重启服务端更新动态算法库和反外挂模块，达到快速封杀新外挂效果。

1.2. GPK 技术特点

1.2.1. 内核技术

GPK 运用了各种先进的内核技术，对外挂和木马用到的各种技术进行阻断，对系统的各个关键点进行守护。

1.2.2. 虚拟机技术

GPK 借鉴成熟、先进的虚拟机技术，并将其灵活运用，在不增加系统资源消耗的情况下充分发挥其特点。提高外挂及木马制作者的逆向难度，使其无法逆向或需要花很大的成本才能逆向。

1.2.3. 动态加解密技术

GPK 对游戏封包进行动态密钥和动态算法加解密，提搞了外挂制作者游戏逆向难度。

1.2.4. 智能监测技术

通过各类外挂的绝对特征，GPK 将自动识别各类外挂，将外挂信息反馈于服务器，并根据合作方需求采取措施。

1.2.5. 外挂屏蔽技术

GPK 将保护内存中的关键数据，并对游戏敏感代码实施监控，使外挂无法修改游戏的数据及代码。



1.3. GPK 的优势

1.3.1. 反外挂能力

反外挂是 GPK 拥有且必须拥有的最基本的能力，GPK 从设计之初就已针对各类外挂做了充分的准备，从 GPK 构架就已经杜绝了脱机外挂存在的可能，与此同时，GPK 拥有数十种先进、成熟的技术手段打击各类外挂。

1.3.2. 反盗号能力

GPK 可保护游戏，使其免疫目前绝大多数的盗号木马所采用的技术。

1.3.3. 安全性、独立性、灵活性、兼容性、稳定性

GPK 反外挂系统拥有极强的安全机制，并且一直在不断提高。

GPK 游戏保护系统采用整体或局部动态更新技术，其操作均独立于游戏产品本身，GPK 更新时，游戏同样无需停机更新。

GPK 拥有极高的灵活性，各种功能均以模块实现，开关非常方便，且非常便于扩展各种功能，使 GPK 对外挂的打击极其灵活，并能为合作方完成其它游戏运营所需要的操作。

GPK 反外挂操作系统客户端兼容 Win2000 及以上所有版本 Windows 操作系统；其服务器端可兼容目前所有常用服务器操作系统。在 GPK 服务的数十万用户中，尚未出现过兼容性的问题。

GPK 拥有极强的稳定性。

1.3.4. 商业合作模式

在确保双方利益的情况下，GPK 支持各类商业合作模型。与合作方共赢；GPK 可根据客户的需求为客户定制 GPK 的特别版本。

2. GPK 实施

2.1. 步骤

- 1、服务联系
- 2、为游戏开发者提供 GPK 模块 SDK
- 3、开发者集成 GPK 模块到游戏客户端和服务端
- 4、服务器配置
- 5、测试
- 6、完成

2.2. 实施时间

大约 7-10 天（包含测试，不包含服务器配置）。



2.3. 集成 GPK 后的反外挂服务

GPK 的目的是为了更好的反外挂，虽然 GPK 的主动防御方式已经对很多作弊软件有了很好的免疫力，但是仍不排除有专门针对特定游戏开发的外挂绕开了反外挂的防护，我们需要对这些外挂有快速灵活的响应机制。为此，集成了 GPK 的游戏我们将提供持续的后续服务。

发游戏运营商发现有新的可用外挂出现时，可以把外挂样本发送到我们的服务邮箱中：gpk@snda.com。附上相应的游戏帐号（如果外挂是收费外挂也请尽量附上外挂帐号，这样有助于更快的解决问题）。我们在收到邮件后两小时内作出响应。对于升级的外挂我们会在两个工作日之内完成封杀更新；对于全新的外挂我们会在四个工作日之内完成封杀更新；对于少数非常紧急的事件，我们可以在优先安排在一个工作日之内完成更新。

3. GPK SDK 相关文件

GPK 将提供一些头文件和 lib 以及 DLL 文件给游戏开发方整合，所有文件及其在我们提供的 GPK 模块中的位置总结如下：

3.1. 头文件

头文件位于 include 目录中。

名称	说明
GPKItClt.h	客户端头文件，在 Include 目录下，用于客户端集成
GPKItSvr.h	服务端头文件，在 Include 目录下，用于服务端集成

3.2. LIB 文件

LIB 文件位于 LIB 目录中。

名称	说明
GPKItClt.lib	客户端 lib，在 LIB 目录下，用于游戏客户端集成
GPKItSvr.lib	服务端 lib，在 LIB 目录下，用于 Release 版游戏服务端集成
GPKItSvrD.lib	服务端 lib，在 LIB 目录下，用于 Debug 版游戏服务端集成

4. GPK 模块客户端 SDK 开发指引

4.1. 步骤一：启动 GPK 并创建 IGPKCltDynCode 组件

在主程序的入口函数（务必在游戏窗口消息循环建立之前调用，防止外挂在不 GPK 启动前利用游戏窗口 HOOK），如 WinMain 或 MFC 的 CxxxApp 的构造函数中调用 SDK 的接口函数 [GPKStart](#)，用来启动保护并返回 [IGPKCltDynCode](#) 接口指针。



示例代码：

```
#include "GPKItClt.h"    //GPK 头文件
using namespace SGPK    //GPK 的命名空间

.....

IGPKCltDynCode *pCltDynCode =
GPKStart(http://autopatch-all.sdo.com/woool/IWoool/Woool-MainServer", "woool");
if(NULL == pCltDynCode)
{
    printf("GPKStart failed\n");
}

.....

////创建游戏窗口
```

4.2. 步骤二、设置动态加解密数据

连接上游戏服务器，从游戏服务器收到动态代码数据后，设置 `sndc` 动态代码数据。客户端只有在设置完动态代码数据后才能利用 `IGPKCltDynCode` 进行加解密。调用 [IGPKCltDynCode::SetDynCode](#) 方法设置动态加解密数据。

示例代码：

```
//存放从服务端接收动态代码数据, 动态代码 size<12K
unsigned char* lpDynCode[14000] ;
int nlen; //接收数据的长度。
nlen = ClientSock.Recv(lpData, 14000); //从游戏服务端接收数据到, 长度为 nLen
pCltDynCode ->SetDynCode(lpDynCode, nlen);
//继续游戏客户端其它代码;
```

4.3. 步骤三、加密解密数据

对从服务端收到的数据调用 [IGPKCltDynCode::DeCode](#) 方法解密，对发往服务端的数据调用 [IGPKCltDynCode::EnCode](#) 方法加密。加解密均会覆盖源数据，长度不变，请注意！

示例代码：



```
unsigned char lpData[1024] ; //存放从服务端接收数据
int nlen; //接收数据的长度。
nlen = ClientSock.Recv(lpData, 1024);
pCltDynCode->Decode(lpData, nlen); //解密
//处理数据
//.....
pCltDynCode->Encode(pData, len); //加密
ClientSock.Send(pData, len); //发往服务端
```

4.4. 步骤四、释放组件

游戏客户端退出时，调用 [IGPKCltDynCode::Release](#) 方法;

代码示例：

```
pCltDynCode->Release();
```

5. GPK 模块服务端 SDK 开发指引

5.1. 步骤一、创建 IGPKSvrDynCode 组件

在服务端程序启动后，Socket 服务启动前调用。通过调用 [GPKCreateSvrDynCode](#) 函数创建 IGPKSvrDynCode 组件

示例代码：

```
#include "GPKitSvr.h" //GPK 服务端头文件
using namespace SGPK //GPK 的命名空间

IGPKSvrDynCode * pSvrDynCode = GPKCreateSvrDynCode();
if(NULL == pSvrDynCode)
{
    printf("Create SvrDynCode component failed\n");
}
```

5.2. 步骤二、初始化动态加解密数据

创建 IGPKSvrDynCode 组件后即可通过 [IGPKSvrDynCode::LoadBinary](#) 方法初始化动态加解密库。

示例代码：



```
int nBinCount;

//载入动态代码库

nBinCount = pSvrDynCode->LoadBinary("DynCode\\Server", "DynCode\\Client");

if(0 == nBinCount || -1 == nBinCount);    //出错

    goto fail_ret;
```

5.3. 步骤三、为每个连接的客户端分配一个随机的索引值

当每个客户端和服务端建立连接时。通过 [IGPKSvrDynCode:: GetRndIdx](#) 方法为它分配一个随机的动态代码索引值。以后这个 Session 就用这个索引动态代码加解密。

示例代码：

```
int nCodeIdx  =    pSvrDynCode->GetRndIdx();

//保存这个索引；
```

5.4. 步骤四、取得索引指向的动态代码数据发往客户端

取得随机索引后，通过 [IGPKSvrDynCode:: GetCltDynCode](#) 方法取出索引指向的数据发往客户端。在 GetRndIdx 成功后调用。

示例代码：

```
const unsigned char *pCode = NULL;

int nLen = pSvrDynCode-> GetCltDynCode (nCodeIdx, &pCode) ;

ServerSock. Send (pCode, nLen) ;
```

5.5. 步骤五、加密解密数据

对从客户端数据调用 [IGPKSvrDynCode::DeCode](#) 方法解密，对发往客户端的数据调用 [IGPKSvrDynCode::EnCode](#) 方法加密,加解密均会覆盖源数据，长度不变，请注意！

示例代码：

```
unsigned char* lpData[1024]    ; //存放从客户端接收数据

int nlen; //接收数据的长度。

nLen = ServerSock. Recv(lpData, 1024)

pSvrDynCode->Decode(lpData, nlen, nCodeIdx); //解密

//处理数据

//.....

pSvrDynCode->Encode(pData, len, nCodeIdx); //加密

ClientSock. Send(pData, len); //发往服务端
```



5.6. 步骤六、释放 IGPKSvrDynCode 组件

在服务端退出时通过 [IGPKSvrDynCode:: Release](#) 方法释放 IGPKSvrDynCode 组件

代码示例:

```
pSvrDynCode->Release();
```

5.7. 不重启服务端更改动态加解密库

针对新外挂可能会更新动态加解密库,这时可以不重启服务器更新动态加解密库。以便用户有更好的游戏体验。

首先要正确的配置好服务端(见 6.1)。

备份好原来的 bin 文件。

把新的 bin 文件分别放进相应的目录中(如 DynCodeBin\Clien\t 和 DynCode\Server\)

在服务端程序中增加一个 GM 指令,如 ReloadDynCode 等(可自行定义),格式如下:

ReloadDynCode

服务端程序接收到这个指令应当调用 [IGPKSvrDynCode](#) 组件的 [LoadBinary](#) 方法,重新载入新的 bin 文件。

示例:

输入 GM 指令:

ReloadDynCode

服务端调用 IGPKSvrDynCode 组件的 LoadBinary 方法

LoadBinary(“aproot\\DynCodeBin\\Server”, “aproot\\DynCodeBin\\Client”);把这两个目录的动态代码对重新载入,以后新登录的用户开始使用新的动态加解密库。

6. GPK CSAuth 功能

6.1. 概述

CSAuth 是 GPK 新增加的一项功能,用来弥补原有模式的不足,其核心思想是把对控制权交给游戏服务端,它通过服务端定时产生一个 128 字节的封包,通过游戏协议发到游戏客户端,游戏客户端接到这个封包后交给 GPK 处理,封包可能是一些参数也可能是一些脚本,主要功能包括:检查 GPK 客户端的完整性,防止恶意屏蔽 GPK 的功能;紧急响应一些新出现的外挂(包括正在线使用该外挂的玩家);把解决问题的响应时间降到最低。

GPK 接入时可以选择是否集成 CSAuth。

6.2. 服务端 CSAuth 开发指引

6.2.1. 生成 IGPKCSAuth 对象

为一个连接会话对象生成一个 IGPKCSAuth 对象。

示例代码(pGpk 为 IGPKDynCode 组件):



```
SGPK::IGPKCSAuth *p_gpkcmd;  
p_gpkcmd = pGpk->AllocAuthObject();
```

6.2.2. 载入 Auth 数据文件

载入 Auth 数据文件, GPK 启动时会默认调用这个函数载入 AuthData.dat,如果要重新载入要求再次调用。

示例代码(pGpk 为 IGPKDynCode 组件):

```
pGpk->LoadAuthFile("AuthData.dat");
```

6.2.3. 生成 CSAuth 数据

用来生成 CSAuth 数据, 并可以按照游戏架构设置回调函数(服务端不推荐回调)。

示例代码(p_gpkcmd 为 IGPKCSAuth 对象):

```
const unsigned char *p = NULL;  
int len = p_gpkcmd->GetAuthData(&p,this,SendToClient);  
SendToClient 是回调的对象指针, 也可能为 NULL,具体由游戏架构决定
```

6.2.4. 检查 CSAuth 数据

检查客户端返回的 Auth 数据

示例代码:

```
void GPK_API SendToClient(void * Object,unsigned char *lpData,unsigned long nLen)  
{  
    printf("%d %d %d\n",nLen,* (WORD*)(lpData),*(WORD*)(lpData+2));  
    ((CServerSocket*)Object)->Send(lpData,nLen);  
}  
void CServerSocket::OnReceive(int nErrorCode)  
{  
    unsigned char buff[512];  
    int len = Receive(buff,512);  
    const unsigned char *p = NULL;  
    int ret = p_gpkcmd->CheckAuthReply(&p,buff,len);  
    if( ret != 0 )  
    {  
        printf("result: %s\n",&p[8]);  
    }  
    CAsyncSocket::OnReceive(nErrorCode);  
}
```



```
}
```

6.2.5. 释放 IGPKCSAuth 对象

连接会话对象退出时，释放 IGPKCSAuth 对象

示例代码：

```
p_gpkcmd->Release();
```

6.2.6. 不重启服务端更新 CSAuth 内容

首先要正确的配置好服务端(见 6.1)。

备份好原来的 AuthData.dat 文件。

把新的 AuthData.dat 文件替换。

在服务端程序中增加一个 GM 指令，如 RelaodCSAuth 等（可自行定义），格式如下：

RelaodCSAuth

服务端程序接收到这个指令应当调用 [IGPKSvrDynCode](#) 组件的 LoadAuthFile 方法，重新载入新的 AuthData.dat 文件。

示例：

输入 GM 指令：

RelaodCSAuth

服务端调用 IGPKSvrDynCode 组件的 LoadAuthFile 方法

LoadAuthFile (“AuthData.dat”);

之后的 CSAuth 将应用新的 CSAuth 文件。

6.3. 客户端 CSAuth 开发指引

6.3.1. 处理 CSAuth 事件

调用 ProcessAuth，处理 CSAuth 事件。

示例代码：

```
void GPK_API SendToServer(void * Object,unsigned char *lpData,unsigned long nLen)
```

```
{
```

```
    ((CClientSocket*)Object)->Send(lpData,nLen);
```

```
}
```

//SendToServe 为定义的回调函数（客户端建议使用回调函数）。

```
void CClientSocket::OnReceive(int nErrorCode)
```

```
{
```

```
    unsigned char buff[512];
```



```
int len = Receive(buff,512);  
m_pGPK->ProcessAuth(this,SendToServer,buff,len);  
CAsyncSocket::OnReceive(nErrorCode);  
}
```

7. GPK 模块的部署和作业流程

7.1. 升级服务器部署

把 GPK 文件夹下的所有文件上传到更新服务器即可。

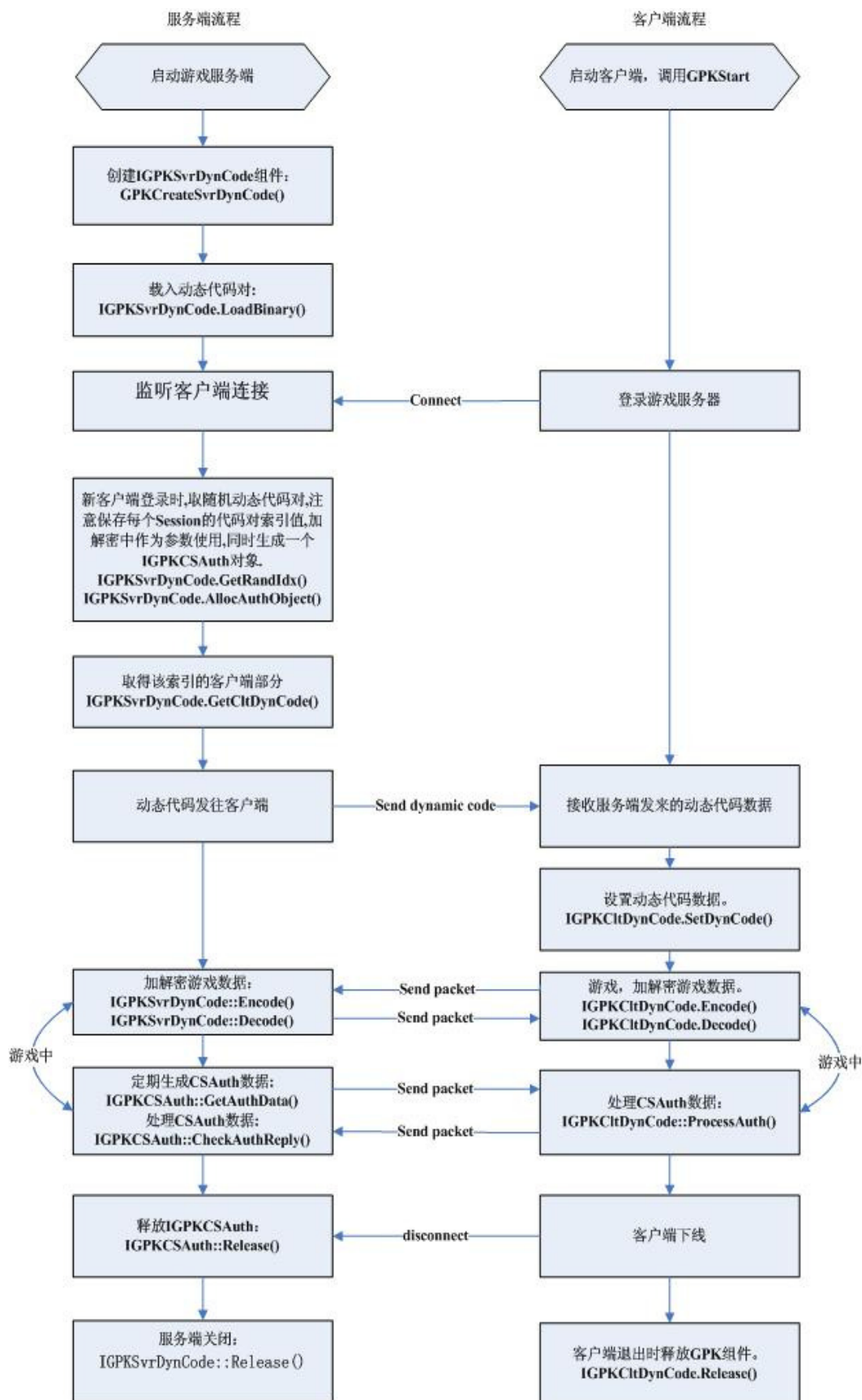
7.2. 游戏服务端部署

- 1、 把 GPKItSvr.dll(Release 版)或者 GPKItSvrD.dll(Debug 版)复制到服务端程序目录。
- 2、 把 AuthData.dat 和 gpkconf.ini 也复制到相同目录下。
- 3、 在服务端新建一个放动态代码目录如 DynCodeBin (可自定义,但必须和 SDK 中的调用相同),并建两个子目录例如 Server 和 Client,分别用来存放动态代码对的服务端和客户端文件,文件的扩展名为.bin,如 01.bin、02.bin,如简介中所指,Server 中的每一个文件都必须在 Client0 中存在一个同名的文件。每两个同名的文件构成一个动态代码对(动态代码对是匹配成对的,文件名相同并不是文件相同,这些文件将由反外挂小组提供)。在 Redist\Server 目录中,我们已经做好了一个 DynCodeBin 目录以及相关文件,可以直接把这个目录复制到服务端目录中。

7.3. 游戏客户端集成

- 1、 把 SDK 包中 Redist\Client 目录下的 gpk 子目录及子目录下所有文件复制到游戏客户端目录下,再把 Redist\Client 目录下的也复制到游戏客户端目录下。如:某游戏的客户端目录是 C:\GameOne,那么会在这个目录中增加一个 gpk 目录,和一个 GPKItCl.dll 文件。
- 2、 通过 GPK SDK 接口启动保护(见第四部分:GPK 模块客户端 SDK 开发指引)。

7.4. 服务端和客户端作业流程





8. 集成 GPK 后的调试和机器人测试

8.1. 在调试器中调试程序

由于 GPK 具备强大的反调试功能，所以，当游戏集成 GPK 后，在调试器中运行游戏程序是无法进行中断的。这给游戏开发中的调试也带来了一些麻烦。以下给出几种解决方案：

- 1、 在客户端和服务端为 GPK 相关代码增加一个宏进行控制。编译调试版本时通过宏关闭 GPK 代码。
- 2、 在 GPK 的 SDK 包中有专门为调试准备的相关文件，位于 Redist\Debug 目录下。内部调试阶段 GPK 使用这些文件，对外发布时使用 Redist\Release 的相关文件。（推荐）

8.2. 机器人测试程序的编写

机器人测试理论上仍然属于调试的一部分，所以 GPK 相关文件仍然使用 SDK 包中的 Redist\Deubg 的文件，同时，考虑到机器人程序在同一个程序里登录大量帐号，我们为机器人测试专门编译了一个新的 GPKItClt.dll，极大的提高了调用 GPKStart 的速度。这个 DLL 位于 Redist\Debug\bot_client 目录中。使用时为每一个机器人帐号调用:GPKStart(NULL,"bot") 创建 IGPKCltDynCode 组件即可。

9. GPK SDK 接口参考

9.1. GPK 命名空间

GPK 模块命名空间为:SGPK

9.2. GPKStart 函数

功能：

自动升级 GPK 组件，启动内核保护进程。并创建 IGPKSvrDynCode 组件

函数声明：

```
IGPKCltDynCode* GPK_API GPKStart(IN LPCSTR UpdateServerURL,  
                                   IN LPCSTR GameCode);
```

参数：

UpdateServer: [in] 更新服务器的 URL 地址；如果本参数为 NULL，则不执行升级（这种为 NULL 情况主要用于调试、测试等）。

GameCode: [in] 游戏代号，8 字节长，加\0 结尾最长为 9 字节长。

返回值：

成功返回 IGPKCltDynCode 组件的接口指针，失败返回 NULL。



9.3. GPKCreateSvrDynCode 函数

功能:

创建 IGPKSvrDynCode 组件

函数声明:

IGPKSvrDynCode * GPK_API GPKCreateSvrDynCode()。

参数:

无。

返回值:

成功返回 IGPKSvrDynCode 接口指针。失败返回 NULL。

9.4. IGPKSvrDynCode 组件

9.4.1. 概述

IGPKSvrDynCode 组件为服务端组件，主要用于动态加解密以及不重启服务端更新动态算法库。

9.4.2. IGPKSvrDynCode::LoadBinary

功能:

载入动态代码文件对

函数声明:

int GPK_API LoadBinary(const char * pszSvrBinDir, const char * pszsClbBinDir)

参数:

pszSvrBinDir : [in]服务端动态代码文件所在的目录。

PszsClbBinDir: [in]客户端动态代码文件所在的目录。

返回值:

成功返回载入动态代码文件对的数量，失败返回-1。

9.4.3. IGPKSvrDynCode::GetRandIdx

功能:

取一个随机的动态代码索引值。

函数描述:

int GPK_API GetRandIdx()。

返回值:

成功返回一个大于等于 0 的整数，表示动态代码对的索引号。
失败返回-1。

9.4.4. IGPKSvrDynCode::GetClbDynCode

功能:

取得准备发往客户端的动态代码的数据。

函数声明:

int GPK_API GetClbDynCode(int nCodeIdx, const unsigned char **ppCodeRet)

参数:

nCodeIdx: [in] 通过 GetRandIdx 方法取得的有效动态代码文件对索引号。



ppCodeRet: [out]输出函数内生成的存放动态代码数据的指针的地址。预先置*ppCodeRet 为 NULL

返回值:

成功返回动态代码数据的长度（大于 0）。

9.4.5. IGPKSvrDynCode::Encode

功能:

加密数据。

函数声明:

```
bool GPK_API Encode(unsigned char * lpData, unsigned long nLen, int nCodeIdx)
```

参数:

lpData:[in,out]指向待加密数据的指针，加密后数据会覆盖原数据，长度不变。

nLen: [in]待加密数据的长度。

nCodeIdx:该客户端使用的动态代码对的索引号

返回值:

成功为 true, 失败为 false.

9.4.6. IGPKSvrDynCode::Decode

功能:

解密数据。

函数声明:

```
bool GPK_API Decode(unsigned char * lpData, unsigned long nLen, int nCodeIdx)
```

参数:

lpData:[in]指向待解密数据的指针，解密后数据会覆盖原数据，长度不变。

nLen: [in]待解密数据的长度。

nCodeIdx:该客户端使用的动态代码对的索引号。

返回值:

成功为 true,失败为 false。

9.4.7. IGPKSvrDynCode::Release

功能:

释放 IGPKSvrDynCode 组件。

函数声明:

```
void GPK_API Release()
```

参数:

无

返回值:

无类型

9.4.8. IGPKSvrDynCode::LoadAuthFile

功能:

载入 Auth 数据文件。

**函数声明:**

```
bool GPK_API LoadAuthFile(const char * pszCmdFileName)
```

参数:

pszCmdFileName: 文件名

返回值:

成功为 true, 失败为 false。

9.4.9. IGPKSvrDynCode:: AllocAuthObject**功能:**

为一个连接会话生成一个 IGPKCSAuth 对象。

函数声明:

```
IGPKCSAuth* GPK_API AllocAuthObject()
```

参数:

无

返回值:

成功返回 IGPKCSAuth 对象的指针, 失败返回 NULL。

9.5. IGPKCltDynCode 组件**9.5.1. 概述**

IGPKCltDynCode 组件用与客户端, 主要用于动态算法加解密, 保护 GPK 内核模块完整性。

9.5.2. IGPKCltDynCode::SetDynCode**功能:**

设置 sndc 动态代码数据。须在初次使用动态代码加解密前调用。

函数声明:

```
bool GPK_API SetDynCode(const unsigned char* lpDynCode, int nLen)
```

参数:

lpDynCode: [in] 从服务端接收到的 sndc 动态数据。

nLen: [in] sndc 动态数据的长度。

返回值:

成功返回 true, 失败返回 false。

9.5.3. IGPKCltDynCode:: Encode**功能:**

加密数据。

函数声明:

```
bool GPK_API Encode(unsigned char * lpData, unsigned long nLen)
```

参数:



lpData:[in,out]指向待加密数据的指针,加密后数据会覆盖原数据,长度不变。

nLen: [in]待加密数据的长度。

返回值:

成功为 true,失败为 false.

9.5.4. IGPKCltDynCode:: Decode

功能:

解密数据。

函数声明:

```
bool GPK_API Decode(unsigned char * lpData, unsigned long nLen)
```

参数:

lpData:[in]指向待解密数据的指针,解密后数据会覆盖原数据,长度不变。

nLen: [in]待解密数据的长度。

返回值:

成功为 true,失败为 false.

9.5.5. IGPKCltDynCode:: Release

功能:

释放 IGPKCltDynCode 组件。

函数声明:

```
void GPK_API Release()
```

参数:

无

返回值:

无类型

9.5.6. IGPKCltDynCode:: ProcessAuth

功能:

处理服务器下发的 CSAuth 数据。

函数声明:

```
Int GPK_API ProcessAuth(void * Object,fnGPK_CltCallBack SendToSvr,unsigned char * lpData, unsigned long nLen)
```

参数:

Object:用于回调的对象指针,可能是一个 socket,也可能是一个用户类等。甚至也可能为 NULL,具体对象类型由游戏架构决定

SendToClc: 回调函数

lpData: [in,out]数据指针,处理后作为输出数据指针

nLen: 数据长度

返回值:

大于 0 为计算后数据长度,否则表示计算失败。



9.6. IGPKCSAuth 组件

9.6.1. 概述

IGPKCSAuth 组件为服务端组件，主要用于检测恶意软件和 GPK 工作状态。

9.6.2. IGPKCSAuth:: GetAuthData

功能：

生成 CSAuth 数据

函数声明：

```
Int GetAuthData(const unsigned char **ppDataRet,void *Object,fnGPK_SvrCallBack  
SendToClt=NULL)
```

参数：

ppDataRet:数据指针

Object:用于回调的对象指针，可能是一个 socket，也可能是一个用户类等。甚至也可能为 NULL,具体对象类型由游戏架构决定

SendToClt: 回调函数

返回值：

大于 0 为数据长度，-1 为异常，0 表示上次的 Auth 检查客户端没有回应。有外挂屏蔽了 Auth 封包。

9.6.3. IGPKCSAuth:: CheckAuthReply

功能：

检查客户端返回的 Auth 数据

函数声明：

```
int CheckAuthReply(const unsigned char **ppDataRet,unsigned char * lpData, unsigned  
long nLen)
```

参数：

ppDataRet:CSAuth 数据指针

lpData:输出的数据指针

nLen:数据长度

返回值：

返回值内容见 GPK_CHECK_INFO

9.6.4. IGPKCSAuth:: Release()

功能：

释放 IGPKCSAuth 组件。

**函数声明:**

```
void GPK_API Release()
```

参数:

无

返回值:

无类型