



SNDA Game Protect Kit SDK Manual (Version2.0)



Contents

Contents	2
1. GPK Introduction	4
1.1. GPK Function	4
1.2. GPK Features	4
1.2.1. Kernel technology	4
1.2.2. VM techonology	4
1.2.3. Dynamic Encryption and Decryption Technology	4
1.2.4. INTEMOR Technology	4
1.2.5. Hack-shielding Technology	5
1.3. GPK Advantages	5
1.3.1. Anti hacking	5
1.3.2. Anti Trojan	5
1.3.3. Security, Independence, Flexibility, Compatibility, Stability	5
1.3.4. Business Cooperation mode	5
2. GPK Implement	5
2.1. Steps	5
2.2. Implement Time	6
2.3. Anti hacking service after GPK integration	6
3. GPK SDK Files	6
3.1. Head File	6
3.2. LIB File	7
4. Development Guide of GPK Module Client SDK	7
4.1. Step One: Start GPK and create IGPKCltDynCode component	7
4.2. Step Two: Set dynamic encryption and decryption data	8
4.3. Step Three: Encryption and decryption data	8
4.4. Step Four: Release Component	8
5. Development Guide of GPK Module Server SDK	9
5.1. Step One: Create IGPKSvrDynCode Component	9
5.2. Step Two: Initialize dynamic encryption and decryption data	9
5.3. Step Three: Assign a random index value to each client	9
5.4. Step Four: get the index dynamic code and send it to the client	9
5.5. Step Five: Encryption and decryption data	10
5.6. Step Six: Release IGPKSvrDynCode Component	10
5.7. Update the dynamic encryption and decryption database without restarting server	10
6. GPK CSAuth Fucntion	11
6.1. Introduction	11
6.2. Development guide for server CSAuth	11
6.2.1. Generation of IGPKCSAuth	11
6.2.2. load Auth data files	11
6.2.3. Generation of CSAuth data	12
6.2.4. Check of CSAuth data	12



6.2.5.	Release of IGPKCSAuth object	12
6.2.6.	Update CSAuth contents without restarting server	13
6.3.	Development guidance of client SCAuth	13
6.3.1.	Deal with CSAuth event.....	13
7.	GPK Module Deployment and Work Flow	14
7.1.	Upgrade Server Deployment	14
7.2.	GS Deployment	14
7.3.	Game Client Integration.....	14
7.4.	Work Flow of Server and Client.....	14
8.	The debugging after GPK integrity and robot testing.....	16
8.1.	Debugging programs in debugger	16
8.2.	The compile of robot test procedure.....	16
9.	Reference to GPK SDK Interface	16
9.1.	GPK Namespace	16
9.2.	GPKStart Function.....	16
9.3.	GPKCreateSvrDynCode Function	17
9.4.	IGPKSvrDynCode Component.....	17
9.4.1.	Introduction.....	17
9.4.2.	IGPKSvrDynCode::LoadBinary	17
9.4.3.	IGPKSvrDynCode::GetRandIdx	17
9.4.4.	IGPKSvrDynCode::GetCltDynCode	18
9.4.5.	IGPKSvrDynCode::Encode	18
9.4.6.	IGPKSvrDynCode::Decode.....	18
9.4.7.	IGPKSvrDynCode:: Release.....	18
9.4.8.	IGPKSvrDynCode:: LoadAuthFile	19
9.4.9.	IGPKSvrDynCode:: AllocAuthObject	19
9.5.	IGPKCltDynCode component.....	19
9.5.1.	Introduction.....	19
9.5.2.	IGPKCltDynCode::SetDynCode	19
9.5.3.	IGPKCltDynCode:: Encode.....	20
9.5.4.	IGPKCltDynCode:: Decode.....	20
9.5.5.	IGPKCltDynCode:: Release.....	20
9.5.6.	IGPKCltDynCode:: ProcessAuth.....	20
9.6.	IGPKCSAuth component.....	21
9.6.1.	Introduction.....	21
9.6.2.	IGPKCSAuth:: GetAuthData	21
9.6.3.	IGPKCSAuth:: CheckAuthReply	21
9.6.4.	IGPKCSAuth:: Release()	22



1. GPK Introduction

1.1. GPK Function

GPK is a kind of ring0 game protection tools. The function is as follow:

- 1 、 Anti debug and all kinds of hacking tools, like Ollydbg, Rootkit.
- 2 、 Prevent controls from Macro & Auto-Mouse.
- 3 、 Protect game process from memory hacking.
- 4 、 Make a HASH check in the memory segment appointed by the game procedure with the anti-modifying result.
- 5 、 Make dynamic encryption and decryption algorithm for game packet. The game is able to adopt the GPK algorithm directly without redesign by yourself. However, it is possible for you to design your own algorithm plus the GPK algorithm.
- 6 、 Anti speed hack.
- 7 、 Anti Key-loggers of Ring0 and Ring3 to efficiently protect the accounts.
- 8 、 Anti Trojan with no Trojan injection or modifying in game process.
- 9 、 Upgrade the dynamic algorithm and anti hacking module without restarting the server to kill the new hacking much faster.

1.2. GPK Features

1.2.1. Kernel technology

GPK adopts all kinds of kernel technology to stop the hacking and give a real-time protection to all key points.

1.2.2. VM techonology

GPK makes full use of the mature and advanced VM technology without increasing the consumption of system resources. It enhances the reverse difficulties in making hacking programs and Trojan and the hackers unable to reverse or able to do so with high cost.

1.2.3. Dynamic Encryption and Decryption Technology

GPK uses dynamic encryption and decryption to enhance the reverse difficulties in making hacking programs.

1.2.4. INTEMOR Technology

By means of all kinds of the hacking characteristics, GPK will automatically identify them and feedback to the server and take measures according to the needs of the



cooperative side.

1.2.5. Hack-shielding Technology

GPK will protect the key data in the memory and monitor the sensitive codes in order to make it impossible for the hacking programs to modify the in-game data or codes.

1.3. GPK Advantages

1.3.1. Anti hacking

Anti hacking is the most basic function of GPK, which is originally designed to make full preparation for many kinds of hacking. In terms of the GPK architecture, it has put an end to the possibilities of the non-client bot, while it attacks all kinds of hacking programs with tens of advanced technologies.

1.3.2. Anti Trojan

GPK makes the game immune to most of the recent Trojan technologies.

1.3.3. Security, Independence, Flexibility, Compatibility, Stability

GPK's anti hacking system boasts the powerful security mechanism with continuous upgrade.

GPK adopts the whole or part dynamic upgrade skills and its control is independent of the game itself. The game upgrade can go without shutdown when GPK is upgrading.

For the high flexibility of GPK, various kinds of function is able to realize by module with convenient switch and extension. Such flexibility can also complete the operation of other games' running for the cooperative side.

GPK anti hacking OS client is compatible with all windows OS from Win2000. Its server is compatible with all common server OS. Up to now, there is no compatible problems from its several hundred thousand of users.

GPK owns strong stability.

1.3.4. Business Cooperation mode

On the condition of assuring the mutual interest, GPK supports all kinds of commercial cooperation mode to get a win-win result. GPK is able to make a special version to meet the needs of the client.

2. GPK Implement

2.1. Steps

- 1 、 Service contact.
- 2 、 Provide GPK module SDK for the game developer.
- 3 、 The developer integrates GPK module into the game server.



4 、 Server configuration

5 、 Test.

6 、 Complete.

2.2. Implement Time

About 7 to 10 days (including test, without service configuration).

2.3. Anti hacking service after GPK integration

The appearance of GPK is for the better anti hacking. Although GPK's active defense mode is immune to many cheating software, it is possible for some hacking ways which are special for the game development to avoid the anti hacking defense. Therefore, it is necessary to form the quick and flexible responding system for such kind of hacking. So we will keep on the after service for those games integrated with GPK.

When the game providers find the new kind hacking, they can send the hacking samples to our service mailbox gpk@snda.com with the relevant game account attachment. If it is charge hacking, pls give the hacking account as much as possible for the quicker solution. After receiving the mail, we will respond within 2 hours. For the upgrading hacking, we will complete the block update within 2 working days, while for the brand-new hacking, within 4 working days. For the few emergent cases, we can finish the update in one day in priority.

3. GPK SDK Files

GPK will provide some head files, lib and DLL files to the game developers. All the files and their position in the GPK modules we provide is are as follow:

3.1. Head File

They are in 'Include' directory.

Name	Description
GPKitClt.h	It is client head file, in 'Include' directory and is used to the integration of the client.
GPKitSvr.h	It is server head file, in 'Include' directory and is used to the integration of the server.



3.2. LIB File

LIB file is in 'LIB' directory.

Name	Description
GPKitClt.lib	It is client lib, in 'LIB' directory and is used to integration of the Release Game Client.
GPKitSvr.lib	It is server lib, in 'LIB' directory and is used to integration of the Release Game Server.
GPKitSvrD.lib	It is server lib, in 'LIB' directory and is used to integration of the Debug Game Server.

4. Development Guide of GPK Module Client SDK

4.1. Step One: Start GPK and create IGPKCltDynCode component

Entry function of main procedure should be called before the game window message loop is established to prevent the hack from using HOOK before starting GPK. For example, the constructor of WinMain or MFC's CxxxApp will call SDK interface function GPKStart to start protection and return the pointer of IGPKCltDynCode interface.

Example code:

```
#include "GPKitClt.h"    //GPK head file  
using namespace SGPK    //GPK namespace
```

```
.....
```

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

```
.....
```

```
////create game window
```



4.2. Step Two: Set dynamic encryption and decryption data

Connect to the GS and get the dynamic code data, then set sndc dynamic code data. The client will use IGPKClItDynCode to decode only after finishing dynamic code data setting. Call IGPKClItDynCode::SetDynCode to set dynamic encryption and decryption data.

Example code:

```
//save the dynamic code data from server, dynamic code size<12K
unsigned char* lpDynCode[14000] ;
int nlen; //Receive data length.
nlen = ClientSock.Recv(lpData, 14000); //receive data from GS, length is nLen
pClItDynCode->SetDynCode(lpDynCode, nlen);
//continue other code of game client;
```

4.3. Step Three: Encryption and decryption data

Call IGPKClItDynCode::DeCode to decode the data received from the server and call IGPKClItDynCode::EnCode to encode the data that will be sent to the server. Notice: there is no change in the length because encryption and decryption will cover the original data !

Example code::

```
unsigned char lpData[1024] ; //save the data from the server
int nlen; //receive the data length
nlen = ClientSock.Recv(lpData, 1024);
pClItDynCode->Decode(lpData, nlen); //decode
//data processing
//.....
pClItDynCode->Encode(pData, len); //encode
ClientSock.Send(pData, len); //send to the server
```

4.4. Step Four: Release Component

When the game client exits, call IGPKClItDynCode::Release;

Example code:

```
pClItDynCode->Release();
```




5. Development Guide of GPK Module Server SDK

5.1. Step One: Create IGPKSvrDynCode Component

After server procedure starts and before Socket starts, call GPKCreateSvrDynCode to create IGPKSvrDynCode component.

Example code:

```
#include "GPKitSvr.h" //GPK server head file
using namespace SGPK //GPK namespace

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

5.2. Step Two: Initialize dynamic encryption and decryption data

Create IGPKSvrDynCode component and call IGPKSvrDynCode:: LoadBinary to initial dynamic encryption and decryption database.

Example code:

```
int nBinCount;

//load dynamic code database

nBinCount = pSvrDynCode->LoadBinary("DynCode\\Server", "DynCode\\Client");
if(0 == nBinCount || -1 == nBinCount); //error

goto fail_ret;
```

5.3. Step Three: Assign a random index value to each client

When each client connects to the server, call IGPKSvrDynCode:: GetRndIdx to assign it a random dynamic code index value. Since then, in this session the index value will be applied to the encryption and decryption.

Example code:

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

//save the index;
```

5.4. Step Four: get the index dynamic code and send it to the client

After getting the random index value, call IGPKSvrDynCode:: GetCltDynCode to get the data pointed by the index value and send it to client. Call it after GetRndIdx success.



Example code:

```
const unsigned char *pCode = NULL;  
int nLen = pSvrDynCode-> GetClitDynCode(nCodeIdx,&pCode);  
ServerSock.Send(pCode,nLen);
```

5.5. Step Five: Encryption and decryption data

Call IGPKSvrDynCode::DeCode to decode the data received from client and call IGPKSvrDynCode::EnCode to encode the data that will be sent to the client. Notice: there is no change in the length because encryption and decryption will cover the original data !

Example code:

```
unsigned char* lpData[1024] ; //save the data from the client  
int nlen; //receive the data length  
nLen = ServerSock.Recv(lpData,1024)  
pSvrDynCode->Decode(lpData,nlen, nCodeIdx); //decode  
//data processing  
//.....  
pSvrDynCode->Encode(pData,len, nCodeIdx); //encode  
ClientSock.Send(pData,len); //send to the server
```

5.6. Step Six: Release IGPKSvrDynCode Component

When exiting the game server, call IGPKSvrDynCode:: Release to release IGPKSvrDynCode component.

Example code::

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

5.7. Update the dynamic encryption and decryption database without restarting server

If the new hacking appears, there is no need to restart the server while the dynamic encryption and decryption database maybe need upgrading, therefore users is able to get better game experience.

At first, the server should have a correct configuration. (see 6.1).

Back up the original bin file.

Set the new bin file to the relative directory, like DynCodeBin\Clie\ and DynCode\Server\.

Add a GM command in the server procedure, like RelaodDynCode (can be



self-defined). The format is as follows:

ReloadDynCode

The server procedure receives the command to call the LoadBinary in IGPKSvrDynCode and reload a new bin file.

Example:

Input GM command:

ReloadDynCode

Server calls the LoadBinary in IGPKSvrDynCode.

LoadBinary("approot\\DynCodeBin\\Server", "approot\\DynCodeBin\\Client"); reload the dynamic code pair in the two directories. Since then, new users will adopt new dynamic encryption and decryption database.

6. GPK CSAuth Fucntion

6.1. Introduction

CSAueh is a new-added function of GPK to make up the defects. Its kernel idea is to transfer the control right to the GS, that is, when getting through the server, it forms a 128-bite timing seal packet and send the packet to the game client through the game protocol. The client gives the packet to GPK. The packet is probably some parameters or some scripts, and it main function includes checking the integrity of GPK client to prevent from the malicious GPK shielding, giving emergency response to some new hackings which include the online players using the hacking and reducing the response time of the problem solving to the minimum.

The integration of CSAuth is a kind of option when accessing GPK.

6.2. Development guide for server CSAuth

6.2.1. Generation of IGPKCSAuth

Generate a IGPKCSAuth object for session connection.

Example code(pGpk is IGPKDynCode component):

```
SGPK::IGPKCSAuth *p_gpkcmd;
```

```
p_gpkcmd = pGpk->AllocAuthObject();
```

6.2.2. load Auth data files

Load Auth data files. It is default to call this function to load AuthData.dat when starting GPK, and it needs to recall when reloading.

Example code(pGpk is IGPKDynCode component):

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



6.2.3. Generation of CSAuth data

To generate CSAuth data and set up callback function in according to the game frame(it is better not to callback on the server.)

Example code(p_gpkcmd is IGPKCSAuth object):

```
const unsigned char *p = NULL;  
int len = p_gpkcmd->GetAuthData(&p,this,SendToClient);
```

SendToClient is the callback object pointer and it is possible to be NULL on the basis of the game frame.

6.2.4. Check of CSAuth data

check the Auth data that returned from the client.

Example code:

```
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. Release of IGPKCSAuth object

When the conversation connection object quits, release IGPKCSAuth object.

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



6.2.6. Update CSAuth contents without restarting server

At first, the server should have a correct configuration(see 6.1).

Back up the original AuthData.dat files.

Replace the new AuthData.dat file.

Add a GM command in the server procedure, like RelaodCSAuth (can be self-defined), the format is as follows:

The server procedure receives the command to call theLoadAuthFile in IGPKSvrDynCode and reload new bin file and reload a new AuthData.dat file.

Example:

Input GM command:

RelaodCSAuth

server calls the LoadAuthFile in IGPKSvrDynCode

LoadAuthFile ("AuthData.dat");

Since then, CSAuth will use the new CSAuth files.

6.3. Development guidance of client SCAuth

6.3.1. Deal with CSAuth event

Call CSAuth, deal with CSAuth event.

Example code:

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

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

//SendToServe is defined callback function(it is recommended to use callback function on the client).

```
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 Module Deployment and Work Flow

7.1. Upgrade Server Deployment

Upload all the files in the directory of Targetdir including the GPK.S file to the update server.

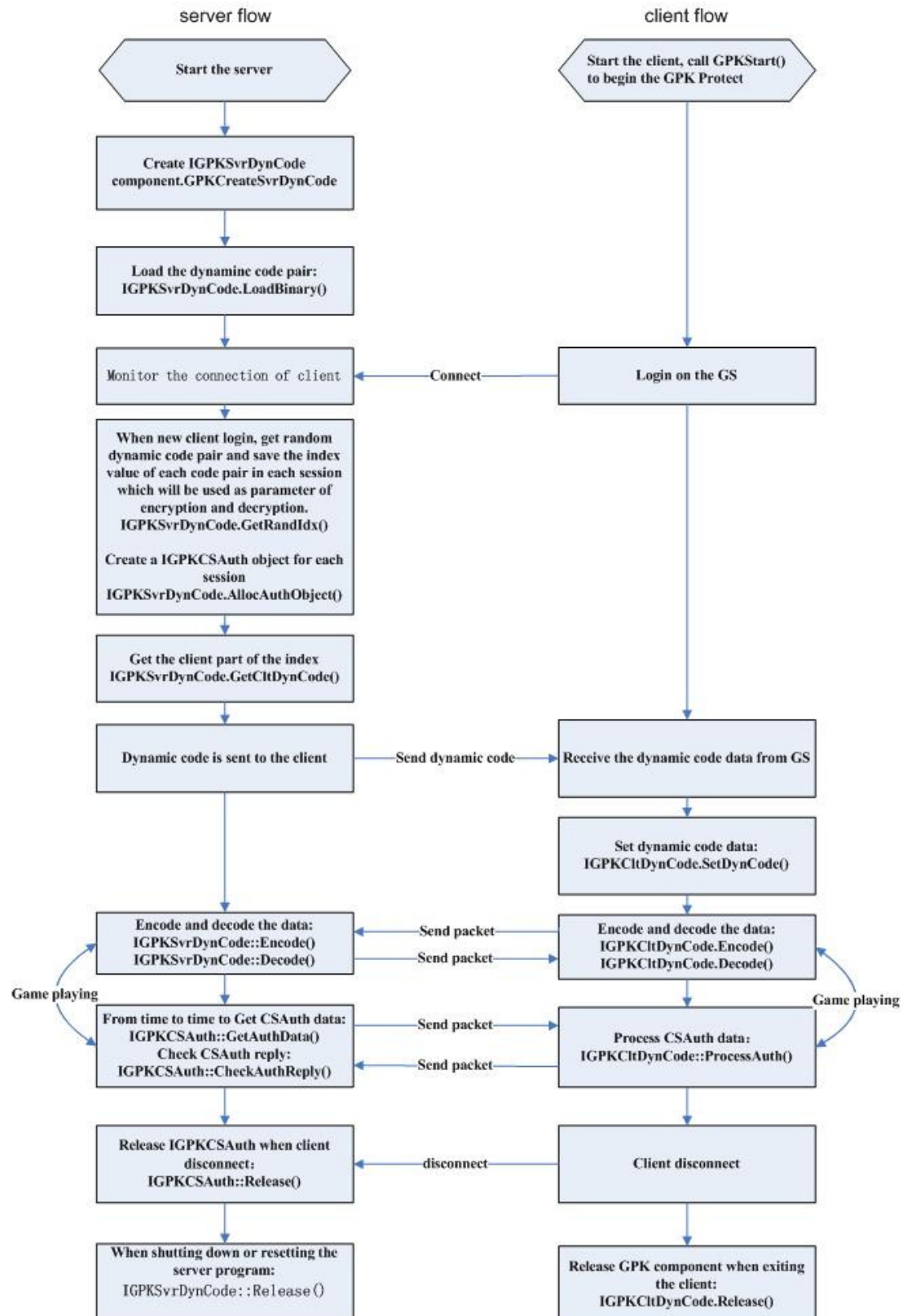
7.2. GS Deployment

- 1 、 Copy GPKitSvr.dll(Release Version) or GPKitSvrD.dll(Debug Version) to the directory of the server program.
- 2 、 Copy SendToServe and gpkconf.ini to the same directoty.
- 3 、 In the server create a directory for dynamic code like DynCodeBin which can be self-defined and must be the same to the call in SDK, and two subdirectories like server and client for the saving of server and client dynamic code files with the file extension name .bin, like 01.bin or 02.bin. Each file in Server0 should have a file with the same name in the client0. The two files with the same name form a pair of dynamic code. The two files with the same name have different content and should be provided by the anti-hacking group. If you want to update the dynamic code without restarting, more subdirectories are to be created, like server1 and client1. Assign the pairs of dynamic codes to these subdirectories. **In Redist\Server, we have prepared a directory DynCodeBin and the related files which can be copied to the directory of the server.**

7.3. Game Client Integration

- 1 、 Copy the files in GPK subdirectory under Redist\Client in SDK packet(seven in all) to the directory of the client and then GPKitClit.dll(Release Version) or GPKitClitD.dll(Debug version, use GPKitClit.dll on the condition of release) to the client directory. For example, a game client directory is C:\GameOne, then add a GPK directory and a GPKitClit.dll file in it. GPK folder includes seven files, AutoUpdate.dll is a must.
- 2 、 Start the protection through GPK SDK interface (refer to the forth part: Development Guide of GPK Module Client SDK).

7.4. Work Flow of Server and Client





8. The debugging after GPK integrity and robot testing

8.1. Debugging programs in debugger

Since the powerful anti-debug function of GPK, the game procedure run in the debugger is unable to be interrupted after the integrity of GPK. Therefore, it brings some trouble in the debug during the game development. The following is some solutions:

- 1、 Add a macro for the relevant GPK code on both client and server to have control. When compiling the debug version, it can close GPK code through the macro.
- 2、 There are some files special ready for the debug in the SDK packet of GPK, and they are under the directory Redist\Debug. During the internal debug, these files are used in GPK, and the relevant files under Ridist\Release will be used when released.

8.2. The compile of robot test procedure

Robot test is still a part of debug, so the files under Ridist\Release in SDK packet will still be used in the relevant GPK files. As in the same program of the robot procedure there are lots of login accounts, we specially compile a new GPKItClt.dll file for the robot test which greatly increases the speed of calling GPKStart. This DLL file is under Redist\Debug\bot_client. If you want to use it, you just call the function GPKStart(NULL,"bot") for each robot account and create IGPKCltDynCode components.

9. Reference to GPK SDK Interface

9.1. GPK Namespace

GPK module namespace is SGPK

9.2. GPKStart Function

Function:

Automatically updating GPK components, start the kernel protection and create IGPKSvrDynCode components.

Function declaration:

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

Parameter:

UpdateServer: [in] update server URL; no update if the parameter is null (The case of NULL is mainly for the debug or test) .

GameCode: [in] game label, length is 8 bytes, if plus \0, the length is 9 bytes at most.

**Return value:**

If success, return the pointer of the IGPKCltDynCode interface, otherwise return NULL.

9.3. GPKCreateSvrDynCode Function

Function:

Create IGPKSvrDynCode

Function declaration:

IGPKSvrDynCode * GPK_API **GPKCreateSvrDynCode**()。

Parameter:

NULL。

Return value:

If success, return the pointer of the IGPKSvrDynCode interface, otherwise return NULL.

9.4. IGPKSvrDynCode Component

9.4.1. Introduction

IGPKSvrDynCode is the server component, mainly used to the dynamic encryption and decryption, and update dynamic algorithm database without server reset.

9.4.2. IGPKSvrDynCode::LoadBinary

Function:

Load dynamic code file pair

Function declaration:

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

Parameter:

pszSvrBinDir : [in] Directory of the server dynamic code files

PszsClcBinDir: [in] Directory of the client dynamic code files

Return value:

If success, return the number of the loading dynamic code files, otherwise return -1.

9.4.3. IGPKSvrDynCode::GetRandIdx

Function:

Get a random dynamic code index value.

Function declaration:

int GPK_API **GetRandIdx**()。

Return value:

If success, return an integer larger than 0 to represent the dynamic code pair index No., otherwise return -1.



9.4.4. IGPKSvrDynCode::GetCltDynCode

Function:

Get the dynamic code data ready to send to the client.

Function declaration:

```
int GPK_API GetCltDynCode(int nCodeIdx, const unsigned char **ppCodeRet)
```

Parameter:

nCodeIdx: [in] Get verified index value of dynamic code file pair by GetRandIdx.

ppCodeRet: [out] Output the pointer ip which generates in the function and save the dynamic code data. Set *ppCodeRet as NULL beforehand.

Return value:

If success, return the length of dynamic code data, otherwise return 0.

9.4.5. IGPKSvrDynCode::Encode

Function:

Encode data.

Function declaration:

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

Parameter:

lpData:[in,out] Set to the pointer of the data being to be encoded. The encoded data will cover the original data with no change in the length.

nLen: [in] Length of the data ready to be encoded.

nCodeIdx: The index value of the dynamic code pair being used by the client.

Return value:

If success, return true, otherwise return false.

9.4.6. IGPKSvrDynCode::Decode

Function:

Decode data.

Function declaration:

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

Parameter:

lpData:[in] Set to the pointer of the data being to be decoded. The decoded data will cover the original data with no change in the length.

nLen: [in] Length of the data ready to be decoded.

nCodeIdx: The index value of the dynamic code pair being used by the client.

Return value:

If success, return true, otherwise return false.

9.4.7. IGPKSvrDynCode::Release

Function:

Release IGPKSvrDynCode component.

Function declaration:



`void GPK_API Release()`

Parameter:

None

Return value:

Void

9.4.8. IGPKSvrDynCode:: LoadAuthFile

Function:

Load AuthData File.

Function declaration:

`bool GPK_API LoadAuthFile(const char * pszCmdFileName)`

Parameter:

pszCmdFileName: AuthFileName

Return value:

If success, return true, otherwise return false.

9.4.9. IGPKSvrDynCode:: AllocAuthObject

Function:

Create IGPKCSAuth object for session connection.

Function declaration:

`IGPKCSAuth* GPK_API AllocAuthObject()`

Parameter:

None

Return value:

If success, return true, otherwise return false.

9.5. IGPKCltDynCode component

9.5.1. Introduction

IGPKCltDynCode components are tailored to the client, mainly used to the encryption and decryption of the dynamic algorithm to protect the integrity of the GPK kernel module.

9.5.2. IGPKCltDynCode::SetDynCode

Function:

Set sndc dynamic code data. It should be called before the first time use of the dynamic code encryption and decryption.

Function declaration:

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

Parameter:

lpDynCode: [in] Receive the sndc dynamic data from the server.



nLen: [in] Length of the sndc dynamic data

Return value:

If success, return true, otherwise return false.

9.5.3. IGPKCltDynCode:: Encode

Function:

Encode data.

Function declaration:

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

Parameter:

lpData:[in,out] Set to the pointer of the data being to be encoded. The encoded data will cover the original data with no change in the length.

nLen: [in] Length of the data ready to be encoded.

Return value:

If success, return true, otherwise return false.

9.5.4. IGPKCltDynCode:: Decode

Function:

Decode data.

Function declaration:

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

Parameter:

lpData:[in,out] Set to the pointer of the data being to be decoded. The decoded data will cover the original data with no change in the length.

nLen: [in] Length of the data ready to be decoded.

Return value:

If success, return true, otherwise return false.

9.5.5. IGPKCltDynCode:: Release

Function:

Release IGPKCltDynCode components.

Function declaration:

void GPK_API Release()

Parameter:

None

Return value:

void

9.5.6. IGPKCltDynCode:: ProcessAuth

Function:

Process Auth Data From Server.

Function declaration:

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

**Parameter:**

Object: Object Pointer for CallBack, can be socket or user's class, etc, also can be NULL. Specific Object Type is decided by Game Framework.

SendToCl: CallBack Function.

lpData: [in,out] Data pointer for Processed Data.

nLen: Length of Data.

Return value:

If success, return the length of data, otherwise return 0.

9.6. IGPKCSAuth component

9.6.1. Introduction

IGPKCSAuth Component is Server Component for detecting malware software and the status of GPK.

9.6.2. IGPKCSAuth:: GetAuthData

Function:

Generate CSAuth Data

Function declaration:

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

Parameter:

ppDataRet: Data Pointer

Object: Object Pointer for CallBack, can be socket or user's class, etc, also can be NULL. Specific Object Type is decided by Game Framework.

SendToCl: CallBack Function

Return value:

If success, return the length of data, otherwise if return -1 means exception, if return 0 means no response in last CheckAuthReply called. Maybe bot blocked the Auth Package.

9.6.3. IGPKCSAuth:: CheckAuthReply

Function:

Check the Auth Data from Client.

Function declaration:

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



Parameter:

ppDataRet:CSAuth Data Pointer

lpData:Data Pointer for output.

nLen:Length of Data.

Return value:

Return value reference to GPK_CHECK_INFO

9.6.4. IGPKCSAuth:: Release()**Function:**

Release IGPKCSAuth Component

Function declaration:

void GPK_API Release()

Parameter:

None

Return value:

Void