MoCpsClient设计说明书

# 历史

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 作者 | 日期 | 描述 |
| V1.0.0 | WuJinlei | 20171124 | Create |
| V1.0.1 | WuJinlei | 20171201 | 根据具体代码实现，修改交互协议中的数据格式；  上传头文件； |
| V1.0.2 | WuJinlei | 20180109 | 修改文件传输时的head和body |

目录

[MoCpsClient设计说明书 1](#_Toc503278306)

[历史 1](#_Toc503278307)

[1.编写目的 3](#_Toc503278308)

[2.业务流程 3](#_Toc503278309)

[2.1.密钥协商 3](#_Toc503278310)

[2.2.控制类的请求 3](#_Toc503278311)

[2.3.数据类的请求 3](#_Toc503278312)

[3.上下文图 3](#_Toc503278313)

[4.序列图 4](#_Toc503278314)

[5.数据（线程、内存） 4](#_Toc503278315)

[5.1.线程 4](#_Toc503278316)

[5.1.1.发送heartbeat的线程 4](#_Toc503278317)

[5.1.2.接收数据的线程 4](#_Toc503278318)

[5.2.内存 4](#_Toc503278319)

[6.交互协议 5](#_Toc503278320)

[6.1.密钥协商 5](#_Toc503278321)

[6.1.1.发送请求的格式(明文) 5](#_Toc503278322)

[6.1.2.返回值的格式(public key加密过的密文) 5](#_Toc503278323)

[6.2.控制类的请求 6](#_Toc503278324)

[6.2.1.发送请求的格式(密文) 6](#_Toc503278325)

[6.2.2.返回值的格式(密文) 6](#_Toc503278326)

[6.3.数据类的请求 6](#_Toc503278327)

[6.3.1.Header 6](#_Toc503278328)

[6.3.2.Body 7](#_Toc503278329)

[7.头文件 7](#_Toc503278330)

[8.流程图 7](#_Toc503278331)

# 1.编写目的

# 2.业务流程

## 2.1.密钥协商

密钥协商，在client使用控制端口和server建立连接之后，必须进行该操作。

本操作可以由server协商出一个加解密算法和密钥，发送到client这里，以后两者之间的通信，都使用该算法和密钥进行加密后进行，保证数据安全。

1. client端，生成一个requestInfo；
2. **明文**形式发送到server；
3. 阻塞等待server的返回值:
   1. 超时时间内，未收到返回值，密钥协商失败，退出；
   2. 收到返回值，step4；
4. 返回值，是server使用public key进行过加密的**密文**，client首先需要用预埋的private key执行解密；
5. 对解密后的数据进行解析，得到加密算法和密钥；

密钥协商就此结束，后续发送和接收数据，都将是使用该算法进行加密过的；

## 2.2.控制类的请求

## 2.3.数据类的请求

# 3.上下文图



上下文图

# 4.序列图

# 5.数据（线程、内存）

存在2个线程，1个可扩展的内存区域占用。

## 5.1.线程

2个线程，分别为发送心跳包的线程，和接收数据的线程。

### 5.1.1.发送heartbeat的线程

该线程，负责在控制端口的socket，连接到server后，向server定时发送一个heartbeat，以此向server声明自己依然存在。

server如果在规定时间内，无法得到client的心跳包，将认为client出现了异常，将单方面关闭和client的连接，client将无法从server得到服务。

### 5.1.2.接收数据的线程

该线程使用数据端口绑定的socket，等待server发送来数据。

该线程接收到server 的数据后，将其组织在内存中，并定期、定量的将数据发送给外部调用者，外部调用者可以通过这些数据重组文件、播放文件等。

## 5.2.内存

client启动时，向内存申请一块4M大小的内存，称为初始内存。

该部分内存，用来存放server发送过来的数据，server发送过来的数据，按块发送。一次最多发送16M的一个块。

如果块的size大于初始内存，内存进行扩展，以4M为一个梯度，持续递增，最多增加到16M.

在持续一段时间，server发送的数据都小于4M的时候，将内存回落，降低到4M以内。

**本部分功能暂未实现，当前只申请了一个4M的空间作为总内存使用；**

# 6.交互协议

## 6.1.密钥协商

### 6.1.1.发送请求的格式(明文)

typedef struct

{

char mark[16]; //MOCPS\_CLIENT

MOCPS\_REQUEST\_TYPE isNeedResp; //1, need response

MOCPS\_CMDID cmdId;

char res[4]; //reserved

}MOCPS\_CTRL\_REQUEST\_BASIC;

typedef struct

{

MOCPS\_CTRL\_REQUEST\_BASIC basicInfo;

MOUTILS\_CHECK\_CRCVALUE crc32; //crc32 value, to check data

}MOCPS\_CTRL\_REQUEST;

### 6.1.2.返回值的格式(public key加密过的密文)

typedef union

{

char desKey[MOCRYPT\_DES\_KEYLEN];

char des3Key[MOCRYPT\_DES\_KEYLEN \* 3]; //can just use 16bytes

// char aesKey[16]; //TODO, this algo. should be done after

char rc4Key[MOCRYPT\_RC4\_KEY\_MAX\_LEN];

}MOCPS\_CRYPT\_KEY\_INFO; //TODO, must assure the key max length of each algo.

typedef struct

{

MOCPS\_CRYPT\_ALGO cryptAlgoNo; //AlgoDes, AlgoDes3, AlgoRc4, and so on

MOCPS\_CRYPT\_KEY\_INFO cryptKey; //The key

unsigned char keyLen; //The length of key

}MOCPS\_CRYPT\_INFO;

typedef struct

{

char mark[MOCPS\_MARK\_MAXLEN]; //MOCPS\_SERVER

MOCPS\_CMDID cmdId; //CMD\_KEYAGREE

MOCPS\_CRYPT\_INFO cryptInfo;

}MOCPS\_KEYAGREE\_RESPONSE;

## 6.2.控制类的请求

### 6.2.1.发送请求的格式(密文)

格式与6.1.1.所述相同，但这里发送时，是发送的密文。加密算法和密钥，来自于6.1.2.中返回的。

### 6.2.2.返回值的格式(密文)

typedef struct

{

char mark[MOCPS\_MARK\_MAXLEN]; //MOCPS\_SERVER

MOCPS\_CMDID cmdId;

char res[3]; //reserved

int bodyLen; //The valid data length in body

char body[2048]; //2K is enough now, if donot enough in future, expand it.

}MOCPS\_CTRL\_RESPONSE\_BASIC;

typedef struct

{

MOCPS\_CTRL\_RESPONSE\_BASIC basicInfo;

MOUTILS\_CHECK\_CRCVALUE crc32; //crc32 to check data

}MOCPS\_CTRL\_RESPONSE;

## 6.3.数据类的请求

client希望从server获取到数据的时候，首先同样是使用6.2.1中的请求格式，发送一条控制命令，server收到命令、处理结束后，会将数据内容，通过数据端口，发送到client来。

因此这里数据类的请求，只有返回值的处理方式，不存在发送请求。

header是密文；body暂时使用明文，减少加解密造成的系统压力；

后续可以考虑，body开始的多少个字节，采用密文，后续明文的形式传输；

### 6.3.1.Header

/\*

To body, we will transport it in following rule:

split to several chunks, then head+chunk being sent from server to client each time;

\*/

typedef struct

{

char mark[MOCPS\_MARK\_MAXLEN]; //MOCPS\_SERVER

MOCPS\_CMDID cmdId;

//The total len of this file

unsigned long long totalLen;

//start with 0, chunk id

unsigned int chunkId;

//except the last chunk, all other chunks, has size MOCPS\_DATA\_BODY\_CHUNK\_MAXSIZE

//To the last chunk, this size being valid, and less than MOCPS\_DATA\_BODY\_CHUNK\_MAXSIZE

unsigned int curChunkSize;

//use check sum algo. to check its right or not;

unsigned char checksum;

}MOCPS\_DATA\_RESPONSE\_HEADER;

一个file，分为若干个chunk传输，每个chunk最大的size为：#define MOCPS\_DATA\_BODY\_CHUNK\_MAXSIZE (1024 \* 1024) //1M；

一个header后，紧跟一个body；

### 6.3.2.Body

紧随header之后，由server发送来的二进制数据，长度由header中的curChunkSize指定。

# 7.头文件



# 8.流程图