## 数据加密的三要素

1、明文/密文

明文：原始数据

密文：加密之后的数据

2、算法

加密算法

解密算法

3、秘钥

一个字符串

不同的加密算法，秘钥长度不同

## 常用加密方式

### 对称加密

加密和解密是使用同一个秘钥，秘钥要保护好，不能泄露

加密效率高

加密强度比较低(相对于非对称加密)

秘钥分发困难(秘钥不能再网络中直接发送)

常见对称加密算法

DES/3DES

AES

RC2/RC4/RC5

IDEA

SKIPJACK

### 非对称加密

暂时留空，不太理解

## Hash算法

**说明：**

Hash算法就是把任意长度的数据转换成固定长度的数据

相同的数据进行哈希运算获得的哈希值相同

同时有很强的抗碰撞性，

不可逆

**通过哈希运算得到的结果被称为**

散列值

哈希值

指纹

**常见哈希算法**

MD4/MD5

SHA-1

SHA-2

## 消息认证码HMAC

通过原始数据加秘钥进行哈希运算，获得哈希值，用于数据校验，确保数据没有被篡改

## 数字签名

不太理解暂时不写

## openSSH的使用windows使用

1、使用安装包安装，64位版本指的是给64位程序使用，32位同前面

2、安装好后，bin目录加入环境变量，解决的是找不到dll库文件

3、指定头文件和lib库文件的目录



4、关联库文件

项目配置里面关联



或者代码关联都可以

#pragma comment(lib,"libssl.lib")

#pragma comment(lib,"libcrypto.lib")

## openSSH的使用linux使用

1、下载原码 ./config make make install 三连

2、需要使用的库名

libssl.so

libcrypto.so

## protobuf的安装

### 语法

[src\项目一\proto语法](src/项目一/proto语法)

### linux安装

1、这里下载安装包

<https://github.com/protocolbuffers/protobuf/tree/master/src>

2、./configure make make install 三连

### windows安装

1、可以直接使用生成好的

2、如果要自己安装目录

[src\项目一\1.md](src/项目一/1.md)

### windows下protobuf使用

1、Protoc生成

2、加入关联lib，dll路径加入环境变量，指定 .h目录

lib名字： libprotobufd.lib

3、加宏 PROTOBUF\_USE\_DLLS

### linux下protobuf使用

1、protoc生成

2、-l指定库名 protobuf，linux下还需要使用线程库pthread

3、加宏 PROTOBUF\_USE\_DLLS，宏不能放在代码里面，要放在预处理器里面

## VS编译常用选项

### windows项目使用

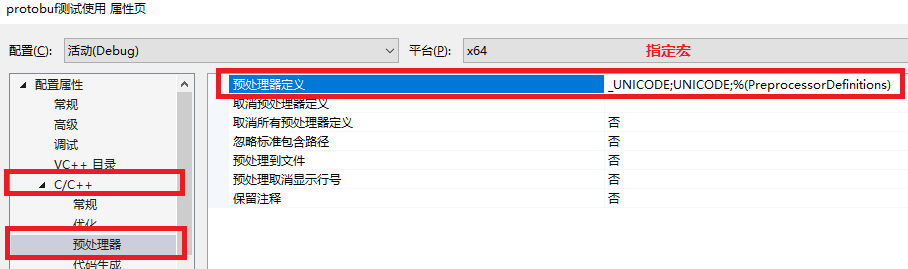
1、指定头文件目录和库文件目录



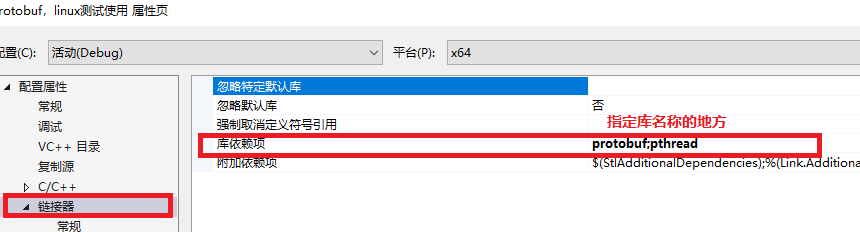
2、关联库名称的地方



3、指定宏的地方



### linux项目使用



## 共享内存

疑惑：共享内存是所有程序共用一内存，还是不同程序使用不同的内存绑定一块共享的内存同步使用（内存映射）

**int shmget(key\_t key, size\_t size, int shmflg);**

功能：创建共享内存，获取共享内存的ID

参数 ：

key: 16进制的数字, 可以简单理解为共享内存的名字, 是唯一的

size: 申请的共享内存大小, 为4k整数倍

shmflag: 创建共享内存参数, 参考open函数

IPC\_CREAT: 创建共享内存

在创建的时候, 需要指定内存的操作权限: IPC\_CREAT|0664

IPC\_EXCL: 创建共享内存的时候检测共享内存是否存在, 必须和IPC\_CREAT一起使用

返回值:

成功: 共享内存的ID

失败: -1,

常用写法：

// 创建共享内存，如果存在，返回共享内存ID

int shmID = shmget(0x12, 100, IPC\_CREAT|0664);

//返回共享内存ID

int shmID = shmget(0x12, 0, 0);

//创建共享内存，创建失败返回-1，创建成功返回共享内存ID

int shmID = shmget(0x12, 4096, IPC\_CREAT|IPC\_EXCL|0664);

//查看共享内存是否存在，无论如何不会返回共享内存

//永远返回-1，如果errno等于17号，表示内存已经存在

int shmID = shmget(0x12, 0, IPC\_CREAT|IPC\_EXCL);

**void \*shmat(int shmid, const void \*shmaddr, int shmflg);**

功能：加入共享内存通信

参数：

shmid: shmget函数的返回值，共享内存的ID

shmaddr: 传入参数, 默认为NULL, 内核会指定虚拟地址空间和共享内存关联的地址

shmflag：对共享内存操作的权限

SHM\_RDONLY: 可以对共享内存进行读操作

0: 读写权限

返回值:

成功返回：共享内存的首地址,

失败返回：(void \*) -1

**int shmdt(const void \*shmaddr);**

功能：退出共享内存通信

参数：

shmaddr: 共享内存的首地址

返回值:

成功返回：0

失败返回：-1

**int shmctl(int shmid, int** **cmd, struct shmid\_ds \*buf);**

功能：共享内存操作，删除共享内存

参数：

shmid: shmget的返回值, 共享内存的ID

cmd: 需要进行的操作

IPC\_STAT: 获取共享内存当前状态

IPC\_SET: 设置共享内存的状态

IPC\_RMID: 标记共享内存要被销毁

buf: 传出参数保存共享内存的状态

PC\_STAT: buf存储数据

PC\_SET: buf中初始化数据 -> 设置到内核

PC\_RMID: 没有用, 传ULL

返回值：

成功返回：0

失败返回：-

注意：

1、删除某块共享内存代码

shmctl(shmID, IPC\_RMID, NULL);

2、共享内存不是马上被删除，要等所有使用共享内存的进程，退出共享内存后共享内存才会被删除

3、标记删除共享内存后共享内存的 key 会变成 0x0 ，此时其他进程不能再加入共享内存通信

4、不加入共享内存通信也可以删除共享内存

5、共享内存被标记删除后，可以使用相同的编号再次创建

**通过ftok生成一个共享内存使用的编号**

key\_t ftok(const char \*pathname, int proj\_id);

- pathname: 路径或文件名, 必须存在

- proj\_id: 只用到了一个字节, 取值范围: 0-255, 也可以传递一字符

2. 思考:

- pathname 是目录还是文件的具体路径，是否可以随便设置？

- 可以, 文件或目录的绝对路径都可以

- pathname 指定的目录或文件的权限是否有要求?

- 没有

- proj\_id 是否可以随便设定，有什么限制条件?

- 取值范围: 0-255

3. 陷阱：

- 误解:

> 只要文件的路径，名称和子序列号不变，那么得到的key值永远就不会变。

- 正解:

> 如果pathname指向的文件或者目录被删除而且又重新创建，那么文件系统会赋予这个同名文件新的inode 节点信息，于是这些进程调用的 ftok() 都能正常返回，但键值key却不一定相同了。

ipcs -a // 打印当前系统中所有的进程间通信方式的信息

ipcs -m // 打印出使用共享内存进行进程间通信的信息 == 常用

================ 以下为了解内容 ================

ipcs -q // 打印出使用消息队列进行进程间通信的信息

ipcs -s // 打印出使用信号进行进程间通信的信息

ipcrm -M shmkey // 移除用shmkey创建的共享内存段

ipcrm -m shmid // 移除用shmid标识的共享内存段

================ 以下为了解内容 ================

ipcrm -Q msgkey // 移除用msqkey创建的消息队列

ipcrm -q msqid // 移除用msqid标识的消息队列

ipcrm -S semkey // 移除用semkey创建的信号

ipcrm -s semid // 移除用semid标识的信号

## UML类图

1、属性在上，函数在下

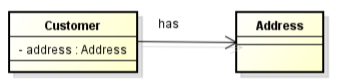
2、 + 公开权限， # 保护权限， - 私有权限

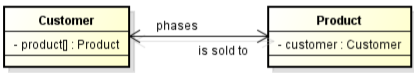
3、继承关系

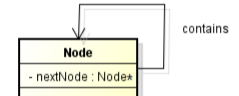


4、抽象类使用斜体字

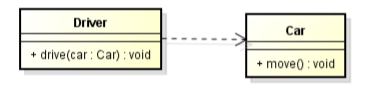
5、一个类使用另外一个类作为成员属性







6、一个类使用了另外一个类的成员方法，属性



## openssl

使用：

1、获取库

2、头文件

哈希

#include <openssl/md5.h>

#include <openssl/sha.h>

aes对称加密

#include<openssl/aes.h>

res非对称加密

include <openssl/rsa.h>

include <openssl/pem.h>

2、库名称

libssl.so;

libcrypto.so

#include <openssl/applink.c>