2017. 09. 28 (周三)

yuanxun @ yuan. engineering 袁询老师

lichangli @ yuan. engineering 李昌立老师 (助教).

服务器: 47.94.19.190. *username: 姓名拼名. password: 待生成

课程任务: ① 连接服务器. ② 文件传输.

	Windows	macOS	Linux.
终端	PuTTY	ssh(1)	ssh(1)
传文件	winSCP.	scp(1)	scp(1).

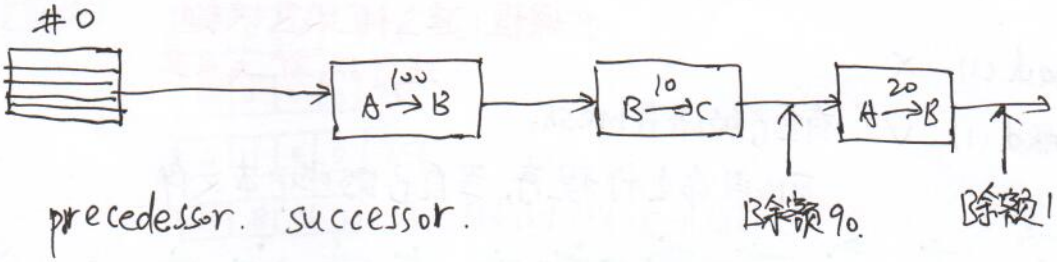
Note: (·) 指地址.

PuTTY 下载地址: https://www.chiark.greenend.org.uk/~sgtatham

macOS 终端命令: ssh 47.94.19.190 -l yuanxun.

(host、username、password)

BC: 是一种数据结构. e.g. BTC. (最长的链) ETH.



Note: ① 想要知道一个帐户总额, 需要知道所有区块中这个帐户的收入、支出。

② 每个区块上大约存储 1800 个交易信息。

③ 每个节点都维持着都有所有的区块. (至今约 150G)

④ 所有交易公开, 如何保护自己的隐私. (e.g. 总余额)

可以通过创建多个地址. (建议: 每一次交易, 创建一个新地址。)

且创建地址是不需要费用的.

ECC: 创建地址的算法, (需要随机数字的产生).

```
a = open("/dev/urandom")    打开文件
b = read(a, 128)            生产随机数
c = get_private(b)          获得私钥
d = get_public(c)           获得公钥
e = get_address(d).         获得地址.
```

Note: 通过 address, 有时可以得到 Public key, 有时不可以.

但通过 address 和 public key.

一定不可以得到 private key.

※私钥很重要.

约 1/4 BTC 因私钥丢失被锁定.

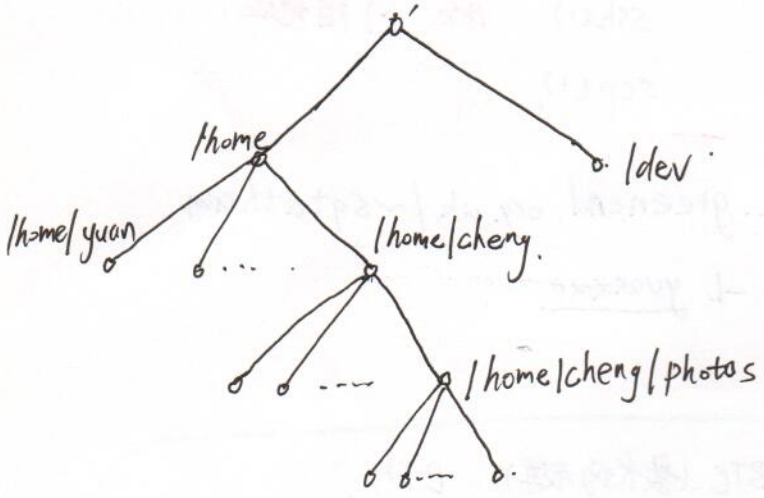
交易费：若高峰期时，交易费少，约几个小时才能完成交易。

BTC优势：可以识别地址的拼写错误。（ETH不可以）

文件夹（卷宗）。

Linux：1991年诞生。
macOS：
Windows：
} 均借鉴Unix。

※UNIX：1979年诞生 其中一个版本是SVR4（1984年）。



文件系统。

※课程：建立班级区块链，进行挖矿、交易等。

命令行 bash(1) ×

mksh(1) √ 有自己的语言mksh，可以用命令行程序，写自己的脚本文件。

协议。

每个BC有3种协议。

A. 格式，交易规则。 eg. 加密/解密算法。
B. 指定BC应用与节点之间的协议。
C. 指定两个节点之间的沟通协议。

BTC { A协议：白皮书、修正、Wiki；（文档不全）
B协议：JSON、RPC。
C协议：TCP。

节点 { 公开节点：别人可访问信息，美国约7000个，中国约几百个。
封闭节点：一般个人节点为封闭的。

※每个节点都备份所有区块。所有节点不一定直接联系，但一定可以间接联系。

区块间隔

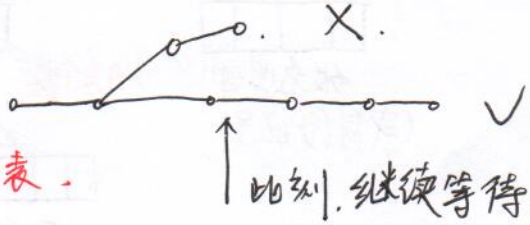
约每10分钟产生一个区块.(约每两周调整难度)

若间隔时间太短,一个区块还没有在全网中传播完成,

另一个区块已开始传播 $\Longrightarrow$ 矛盾.

若某节点,在收到第 400002 个区块之后,收到两个 400003 区块.

此时,应该保留两个区块,待收到第 400004 个区块时,查看它的前继区块即为第 400003.



note:某交易面有几个区块,就说明该交易得到了几次确认

※概率表.

比纠.继续等待.

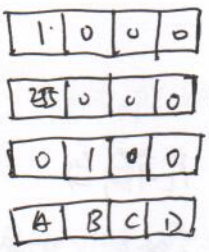
Bitpay:中介.因支付需要20~30分钟确认时间,不方便。

中介公司收取中介费,但承担分支被消灭的风险.

数据.

标准数文件:4个字节,每个字节0~255.

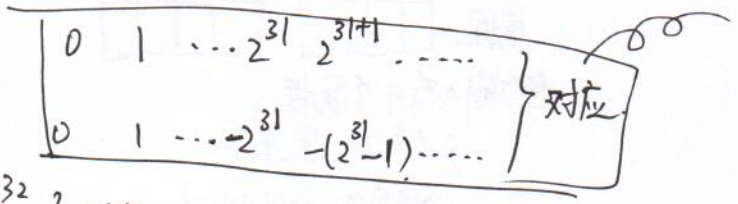
(32位系统):



① 整数:

unsigned(无符号). $0 \sim 2^{32}$. 约40亿.

signed(符号). $-2^{31} \sim 2^{31}$. 包括负数.

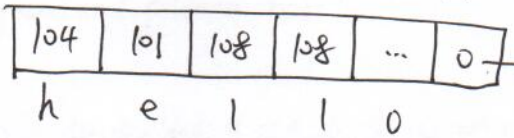


0 1 ···· 2^{31} $2^{31}+1$ $2^{31}+2$ ···· 2^{32}

0 1 ··· -2^{31} $-(2^{31}-1)$ $-(2^{31}-2)$ ···· 0 } 对应.

② 字符串. ASCII码,指定0~255范围内数字→字母对应.

A 65 B 66 ···· a 97 b 98 c 99 d 100 ··· ␣ 32.



→0代表字符串完成.

③中文字. ASCII码已占用了0-127,128-256分配给中文字及其他语言.

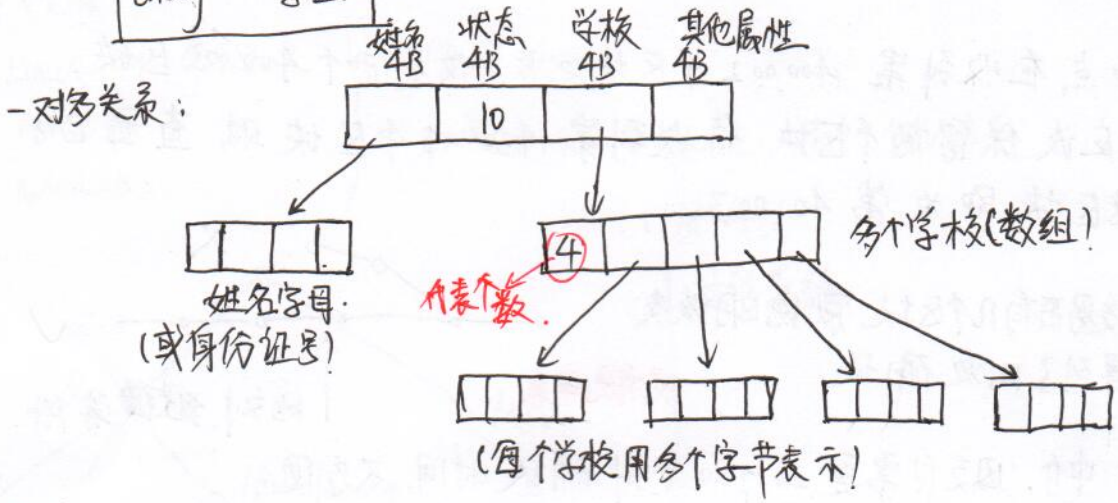
UTF-8 有固定公式,汉字→字节.

田标志：

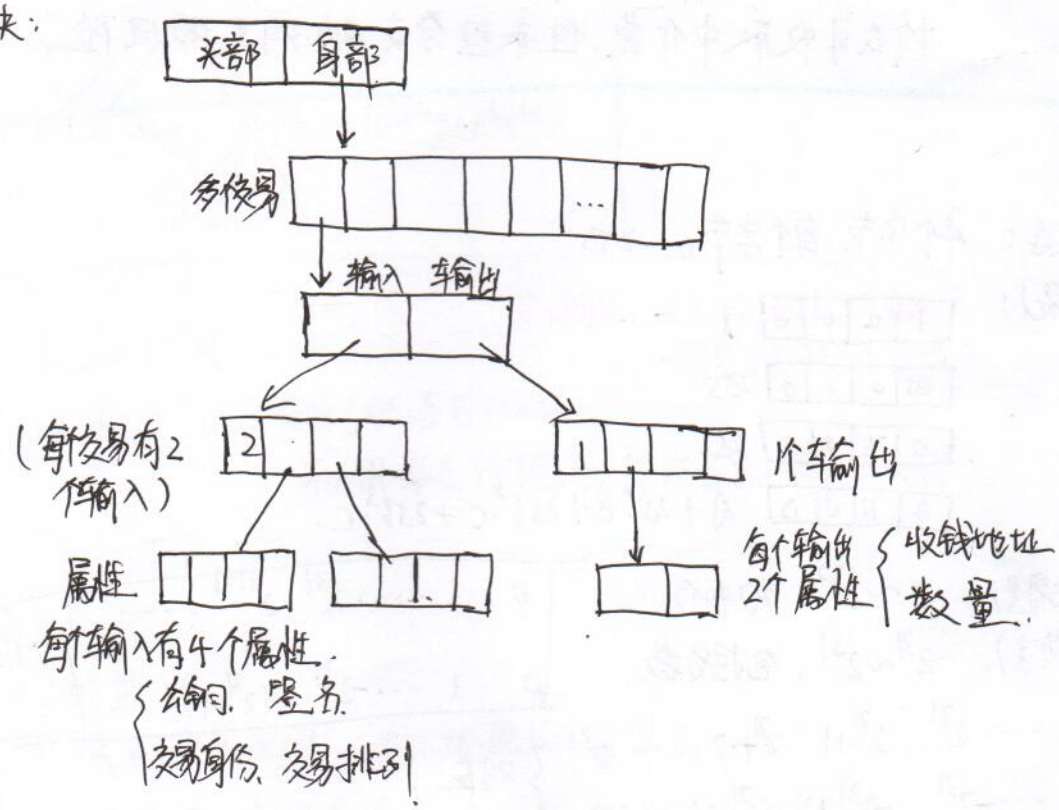
姓名	状态
yuan	老师 → 10
li	财务 → 20
cheng	学生 → 30

状态只有几个固定的选择.

一对多关系：



区块：



2017. 10. 11. 周三.

语言：C

BTC：2008年末，白皮书发布. 2009年初创世区块.

- BTC：只有一条官方链，群体文化不赞成多条链.
- Ethereum：支持有多条链.

- A协议：BC验证交易、规则等.
- B协议：节点与应用
- C协议：节点与节点.

本节任务：主讲A协议和C协议.（B协议作为作业）

目标：用C写一个验证头部的程序.

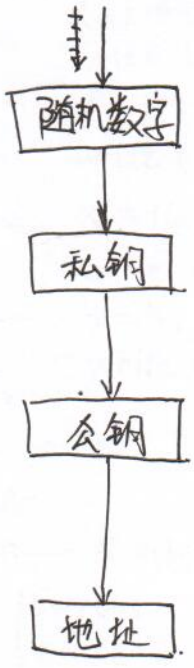
复习上节：

- BTC：碎片.
- Ethereum：使用一个数字代表币.

所有存储在计算机上的数据都是用字节表示的. e.g. [0,256)

system call 系统呼叫

pwd. 当前目录.



f=open(/dev/urandom)
r=read(f, 100)

~~k=~~ y=get_private(r)

z=get_public(y)

a=get_address(z)

※ 两个主要任务：

- 保存 private key. write(b, "hello") △写进b文件 write(1, "hello") △打印在屏幕.
- 显示 address. write(1, a).

eg: c="/home/yuanxun/private.txt"

d=open(c)

write(d, y) 把私钥y保存在自己的private.txt文件里.

(上节复习)

Linux 基本命令：

$ls 显示当前目录下的文件

$pwd 显示当前所在目录.

$ cd. 切换目录. $cd.. 向前一级目录.

$ man 说明书. e.g. man 2 read. man 2 write, man open. (按1是外man命令)

$ nano /tmp/category. (nano是小巧、友好的编辑器)

C: tcc编译器.

$echo SHELL. 显示当前使用的shell. /bin/bash.

$ mksh. 切换到mksh.

$ nano program.c 创建一个新的文件

$ alias nano = "nano -w -x -E -T 3 -I -S" 设置编辑器

(基本命令)

```
#include "stdio.h"
void main()
{   char x = 10;
    printf("Hello, world! %c", 10);   △换行(ASCII码)
    write(1, "hello", 5);   △长度.
    write(1, &x, 1);   △换行.

    char x;
    x=104;
    write(1, &x, 1);   } h
    x=101;
    write(1, &x, 1);   } e
    x=108;
    write(1, &x, 1);   } l.
    x=108;
    write(1, &x, 1);   } l
    x=111;
    write(1, &x, 1);   } o
    x=10;
    write(1, &x, 1);   } 换行.
}
```

命令 { nano program.c
       tcc -run program.c

(打印字符)

http://xiaoma.info.

你 —小写词典→ 4F60 —16进制→10进制→ 20320 —转2进制→ 100 111101 100000.

前加 1110

前加 10

前加 10

111001 00 · 10 111101 10 100000.

转10进制

228 189 160

你.

好 → 597D → 22909 → 101 100101 111101 → 11100101 10100101 10111101

→ 229 165 189.

好

```
void main()
{   char x;
    x=228;
    write(1,&x,1);
    x=189
    write(1,&x,1);
    x=16;
    write(1,&x,1);
    x=10;
    write(1,&x,1);
}
```

你

换行.

(打印中文)

协议A-

HEAD. — 5 Attributes (104字节)

Current	32B	※用 get_SHA256() 的输出总是32字节的. 当前Block的ID.
nonce	4B	※
previous	32B	※前任Block的ID.
target	32B	※
time	4B	※系统时间 timestamp.

※ nonce 和 target 用于 POW. 在链的最初不需要POW, 当有很多个节点创造Block时才用到POW. 此时才用到 nonce 和 POW.

BODY. — ≥1个 Transaction. (大小灵活, 有上限, 可设为1M)

Trancaction — ≥1 input ≥1 output.

※ ECC私钥做的签名大小不是固定的. (73或72或71个字节, 有很小的可能性小于70)

∴不能把 Transaction 大小设为固定.

BODY:

- #1: Coinbase Transaction / Generation Transaction.
- #2: other Transaction
- #3: other Transaction
- ...

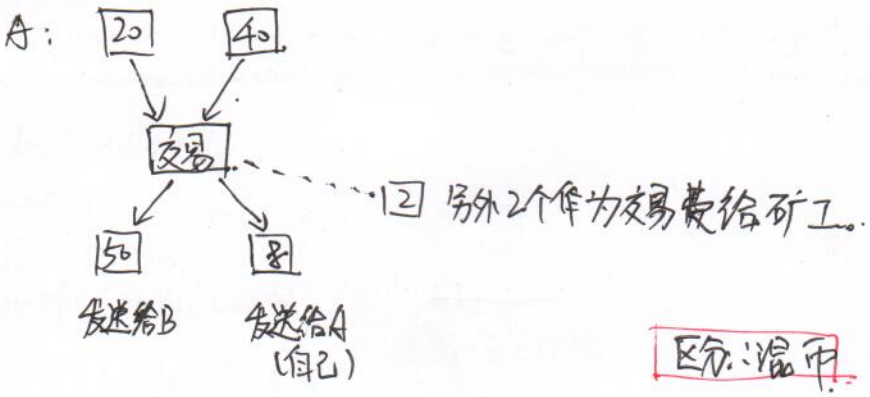
每个区块约 1800个交易. /10分钟 ≈ 3个交易/s.

回奖励: 50 → 25 → 12.5 → 6.25 BTC容量有上限.

Δ now.

BTC的最小单位 $\frac{1}{1亿}$ 比特.

交易过程:



Block #1

[8546] Z → A(50) Δ 奖励给A 50币

Block #2 Z: Δ创币交易,无输入.

[6121] Z → B(50) Δ奖励给B 50币

[8442] A|8546|1 → C(10) A(40) A给C转10币,无手续费

Block #3 输入: A:支付者. 8546/1 该笔钱来自哪个区块的,哪一个交易得到的。

[1680] Z → A(50) A(2) Δ奖励给A 50. 手续费2币

[2044] A|8442|2 → D(30) A(8) Δ转给D 30币. 手续费2币

[4444] C|8442|1 → E(5) C(5)

Block #4

[7777] Z → B(50)

[7801] A|1680|1 A|1680|2 A|2044|2 → X(60)

2017. 10. 14. 周六.

tmux ␣ -S ␣ /tmp/44.

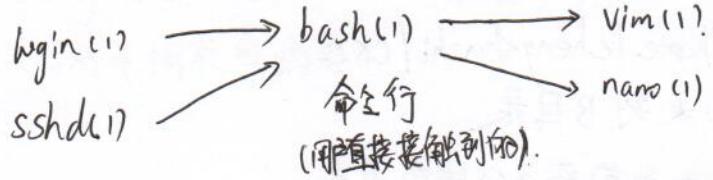
Linux: OS没有用户名和密码的概念.

/etc/passwd. 存储所有账号的身份号和用户名.

/etc/shadow. 存储所有账号的密码(的摘要).

login (1) 近程登录

sshd (1) 远程登录



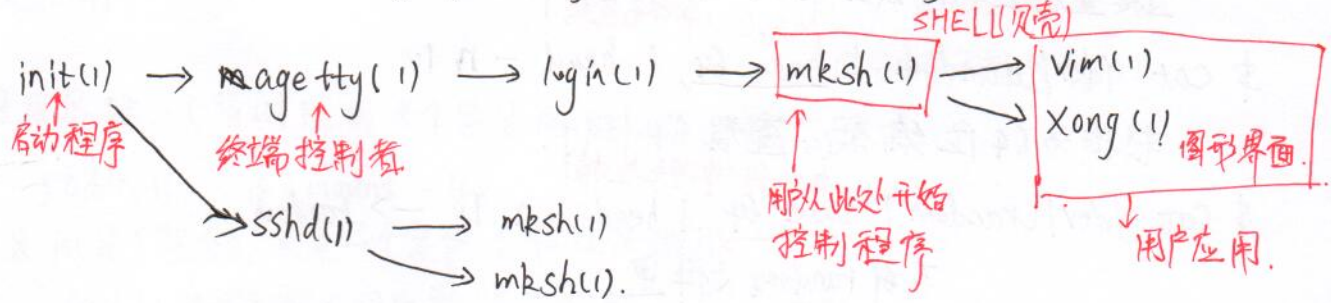
chsh (1) : change shell. 更改默认登录时的 SHELL.

$pwd 当前路径

echo $USER. 用户名 echo $SHELL. 命令程序. ~~$ echo $~~. $exit 退出.

cat /etc/passwd. 所有账号用户名及ID.

cat /etc/passwd ␣ | ␣ grep ␣ chengshuzhi 查找 chengshuzhi



$ cat /etc/passwd ␣ | ␣ wc ␣ -l. 统计当前人数.

$ uname -a. 查看系统配置.

$ rmdir videos 删除 videos 文件夹 $mkdir video 创建 videos 文件夹.

HEAD.

CURRENT 32

NONCE 4.

PREVIOUS 32 → 若是创世区块, 前任全为0.

TARGET 32

TIME 4.

BODY

TRANSACTION #1.

INPUTS: 1

INDEX: 4 → 全为0.

PUBLIC: 0.

SIGNATURE: 0

TRANSACTION: 32 → 全为0

OUTPUT: 1

ADDRESS: 不定, 矿工的地址.

AMOUNT: 8. → 数值是50或100.(区块的奖金)

TRANSACTION #2

INPUTS: ≥1个

INDEX: 4. PUBLIC: 不定

SIGNATURE: 不定 TRANSACTION: 32

OUTPUTS: ≥1个

ADDRESS: 不定. AMOUNT: 8.

Note1: 区块的身份号不在区块里，而是把整个的HEAD放到SHA256()函数里，计算所得。(一个32位的字节)

Note2: BTC：两个数据库
1. 存放区块；
2. 存放交易：为了让交易更容易被搜索。

$ cp /home/yuanxun/note.txt /home/chengshuzhi/cs2.
复制文件从A目录到B目录

$ utmp. wtmp 这两个文件存储当前登录的所有用户.
$ users. 所有用户.

目标：生产随机数字.

$ cat /dev/urandom
直接查看生字节，会出现乱码

$ cat /dev/urandom | base 64 (base64 中间无空格) | head -n 10.
转换为64位编码，查看前10行.

$ cat /dev/urandom | base 64 (base64 中间无空格) | head -n 10 -> random
存到 random 文件里.

Note1: 每一个字母代表6位，每一行有76个字母，每个字节8位.
76×6÷8=57.
⇒ 每一行有57个字节

Note2: BTC：用 Base58函数. 用 RIPEMD-160函数做摘要.

Note3: C语言：① 系统呼叫.(200个)
② 标准函数(1200个)
③ 第三方函数(很麻烦).

生产新的私钥，计算摘要：
~~bip~~ private, public, md5, sha256, base64, base58.

php：最接近于C的动态语言.

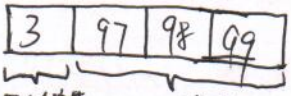
file functions　file_get_contents；file_put_contents.
与文件相关的函数.

array functions：array_push；array_pop；array_shift；array_count；
与数组相关的函数.　array_unshift；array_keys；array_values；

String functions：　substr；strlen
与字符串相关的函数；

字符串：

正当字符串 proper
(Pascal string)

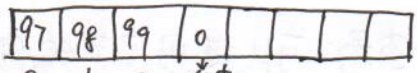


用4个字节存储长度　3个字节

总长度：7.

※php的字符串是正当的，所以，内容可以包括0.

非正当字符串 improper.
(C string)

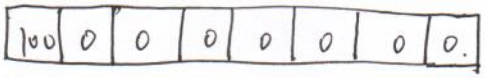


字符串长度：3
占据空间：4.
(字符串里不能包括0)

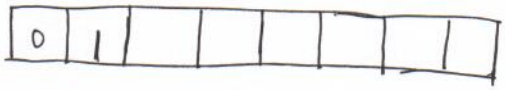
目标：生成区块.（暂时使用8个字母的地址，占8个字节）.

$address　$amount = 100.

※100是个整数，不是一个字节，不能直接插入.
pack()函数把不同的数据类型转换为生字节.



L(低)　(100)　H(高)

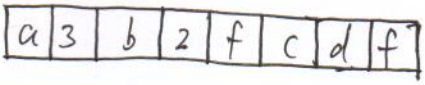


L(低)　(256.)　H(高)
255+1

$X = chr(100).chr(0).chr(0).chr(0).chr(0).chr(0).chr(0).chr(0).

※.代表把字节"粘在一起".

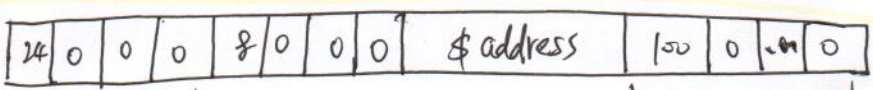
$address.



$address.$X
把地址和金额"粘在一起"

4个字节存储address的长度，其长度为8.
(因为address的长度是灵活的，所以需要存储长度)

再第1个输出output：



总长度　4　⑧.　8.

若address长度为8. 则 total length = 24

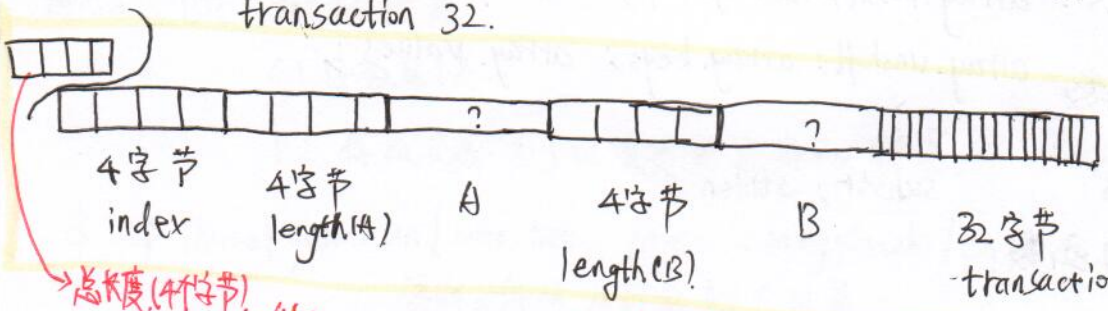
输入：index 4

public A length(A). A

signature B length(B). B.

transaction 32.

} public 和 signature 是灵活的，所以需要标注长度。



总长度(4个字节)

存储数值为：4f 4f 4+length(A)+4+length(B)+32.

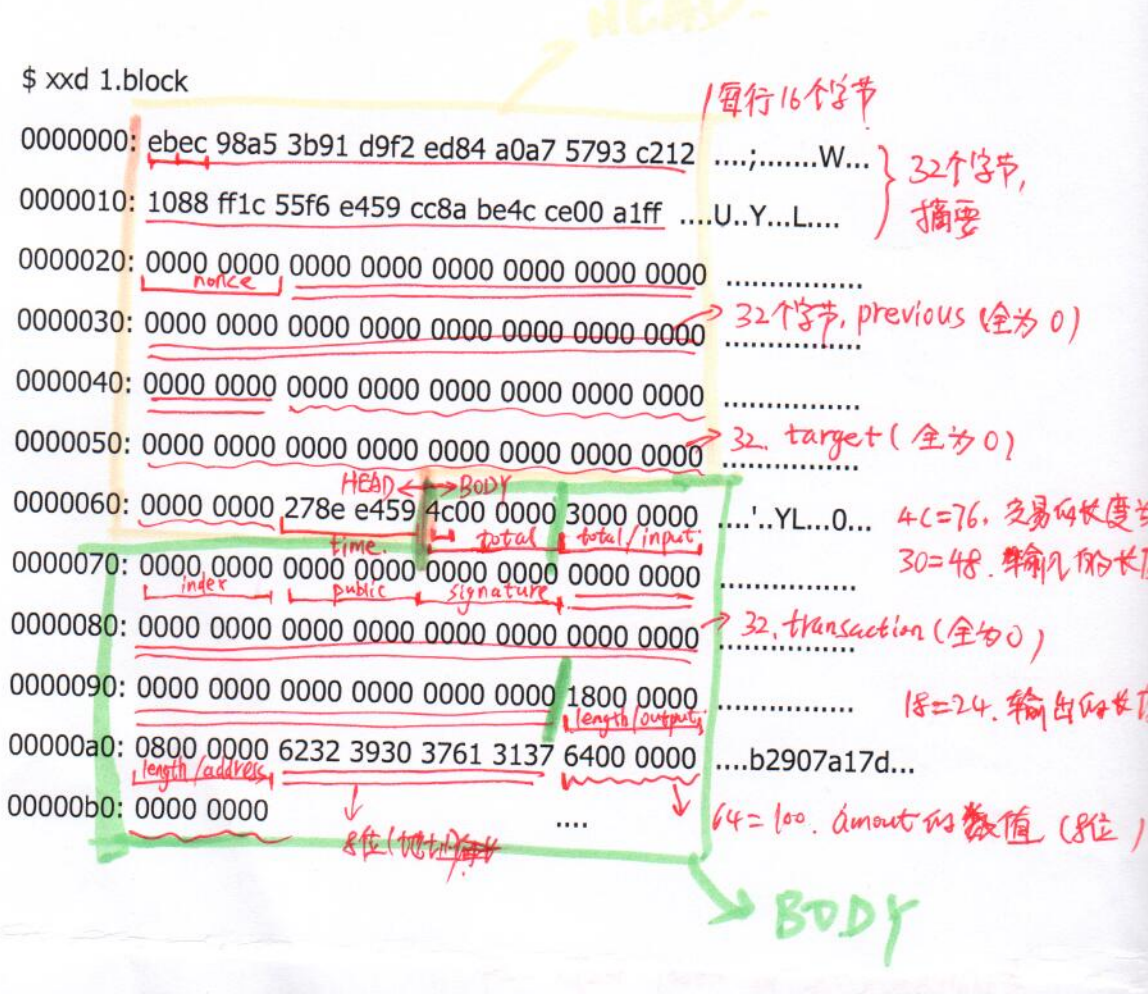
所以：4个属性 → 7个属性 (input)

2个属性 → 4个属性 (output)

第1个input.

1.block 里都是生字节，不能直接查看，可以使用字节编修器 hex editor.

xxd：Linux 自带的字节编修器。eg. xxd 1.block



```php
<?php

//$x = get_coinbase();
//var_dump(strlen($x));
$block = get_head().get_coinbase();
var_dump(strlen($block));
file_put_contents('1.block',$block);
function get_head()
{
 $current = hash('sha256',get_coinbase(),true);
 $nonce = pack('L',0);
 $previous = str_repeat(chr(0),32);
 $target = str_repeat(chr(0),32);
 $time = pack('L',time());
 return $current.$nonce.$previous.$target.$time;
}
function get_coinbase()
{
 $x = get_input();
 $y = get_output();
 $total = pack('L',4+strlen($x)+strlen($y));
 return $total.$x.$y;
}
function get_input()
{
 $a = pack('L',0);
 $b = pack('L',0);
 $index = pack('L',0);
 $transaction = str_repeat(chr(0),32);
 $total = pack('L',48);
 return $total.$index.$a.$b.$transaction;
}


function get_output()
{
 $amount = chr(100).chr(0).chr(0).chr(0).chr(0).chr(0).chr(0).chr(0);
 $address = substr(md5(file_get_contents('public.txt')),0,8);
 $a = chr(8).chr(0).chr(0).chr(0);
 $total = chr(24).chr(0).chr(0).chr(0);
 return $total.$a.$address.$amount;
}
```

```php
<?php
//$random =  file_get_contents('/home/chengshuzhi/csz/random');
//var_dump($random);
//$random = base64_decode($random);
//var_dump(strlen($random));
//srand(intval($random));

//var_dump(rand());
//var_dump(rand());
//var_dump(rand());

function get_private()
{
        openssl_pkey_export($_b4b7 = openssl_pkey_new(), $_eeb0);
        return array($_b4b7, $_eeb0);
}

function get_public($_b065)
{
        $_cb64 = openssl_pkey_get_details($_b065);
        return $_cb64['key'];
}

$private = get_private();
var_dump($private[1]);
var_dump(get_public($private[0]));
```

```
var_dump(substr(md5(get_public($private[0])),0,8));
```

```
$x = hash('sha256','my name is chengshuzhi');
var_dump($x);
```

```php
<?php

//$x = get_coinbase();
//var_dump(strlen($x));
$block = get_head().get_coinbase();
var_dump(strlen($block));
file_put_contents('1.block',$block);
function get_head()
{
 $current = hash('sha256',get_coinbase(),true);
 $nonce = pack('L',0);
 $previous = str_repeat(chr(0),32);
 $target = str_repeat(chr(0),32);
 $time = pack('L',time());
 return $current.$nonce.$previous.$target.$time;
}
function get_coinbase()
{
 $x = get_input();
 $y = get_output();
 $total = pack('L',4+strlen($x)+strlen($y));
 return $total.$x.$y;
}
function get_input()
{
 $a = pack('L',0);
 $b = pack('L',0);
 $index = pack('L',0);
 $transaction = str_repeat(chr(0),32);
 $total = pack('L',48);
 return $total.$index.$a.$b.$transaction;
}


function get_output()
{
 $amount = chr(100).chr(0).chr(0).chr(0).chr(0).chr(0).chr(0).chr(0);
 $address = substr(md5(file_get_contents('public.txt')),0,8);
 $a = chr(8).chr(0).chr(0).chr(0);
 $total = chr(24).chr(0).chr(0).chr(0);
 return $total.$a.$address.$amount;
}
```

```php
<?php
//$random =  file_get_contents('/home/chengshuzhi/csz/random');
//var_dump($random);
//$random = base64_decode($random);
//var_dump(strlen($random));
//srand(intval($random));

//var_dump(rand());
//var_dump(rand());
//var_dump(rand());

function get_private()
{
        openssl_pkey_export($_b4b7 = openssl_pkey_new(), $_eeb0);
        return array($_b4b7, $_eeb0);
}

function get_public($_b065)
{
        $_cb64 = openssl_pkey_get_details($_b065);
        return $_cb64['key'];
}

$private = get_private();
var_dump($private[1]);
var_dump(get_public($private[0]));
```

```
var_dump(substr(md5(get_public($private[0])),0,8));

$x = hash('sha256','my name is chengshuzhi');
var_dump($x);
```

20171018

笔记本： 我的第一个笔记本
创建时间： 2017/10/18 星期三 20:50 **更新时间：** 2017/10/19 星期四 12:46
作者： shuzhichengspace

一、理论知识

1. 系统呼叫：open(1),close(1),read(1),write(1)。
2. 套接字socket：几乎所有经过网络的过程都经过socket。
3. 每一个网络编程中，都包括连接者和被动连接者（监听者）。

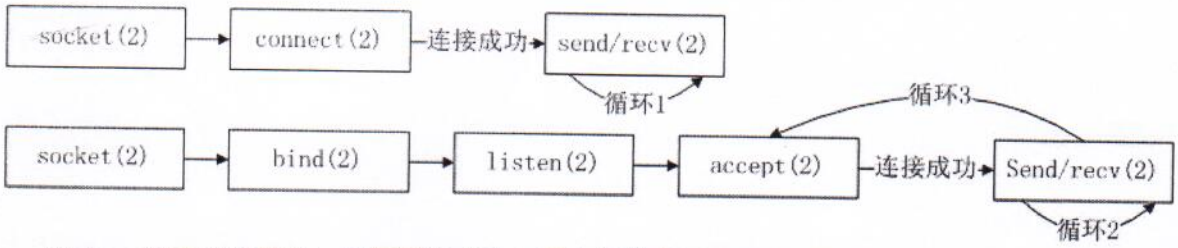
问题：一个监听者在单线程的情况下，如何监听多个连接者？

其中一个解决方法是排队。在监听一个连接着的时候，其他连接者被中断，排队等待。

三个方式实现并行处理：创建新的进程、创建新的线程、使用两个特殊的系统呼叫select(2)或者poll(2)。

4. 课程目标：编写连接者程序和监听者程序，用c语言实现不同的系统呼叫。

如图，为连接者程序和监听者程序。



Note：循环1和循环2，表示连接成功之后会传送或者接收多个文件。

循环3，表示一个监听者会监听多个连接者，所以循环执行accept和send/recv。

socket：制造一个新的套接字。三个参数，socket(family,type,protocol).

family：家族，family的取值为2，表示互联网。

type：类型，取值为1时表示tcp，取值为2时表示udp。tcp更安全，但是udp更快。游戏和网络视频中，一般使用udp，因为更在乎速度。一般上午数据使用tcp，因为更在乎安全。

我们课程使用udp做一个钱包，很快，但是可能会丢包。使用udp再使用udp检查丢包，相当于是一个tcp，但是时间成本远低于tcp。

udp没有连接的概念，不需要使用connect函数。

protocol：协议，取值为0。

domain socket：区域套接，不能上网的套接。

netlink socket：网链套接，也不能上网，只有LInux存在，为了让程序和驱动器沟通。

INET socket：即internet socket，互联网套接，我们正在使用的。

connect：连接，三个参数，connect(x,目标，目标的大小)。

其中，x=socket（family，type，protocol）。

呼叫connect时，系统会卡住等待连接，所以每次做connect时，需要做多线程。这样当系统等待连接卡住的时候，其他线程不会被影响。

y=connect（x，目标，目标的大小）。表示是否连接成功。

send：发送数据，三个参数，send(x,内容，内容的大小，flags)。其中，flags表示一些

特殊的要求。

recv：接收数据，四个参数，recv（x，缓冲，缓冲大小，flags）。其中，flags表示一些特殊的要求。

5. 问题：先写连接者程序，未写监听者程序的时候，如何测试：

使用nc(1)，既可以做连接者，又可以做监听者。当测试连接者时，nc(1)作为监听者。当测试监听者时，nc(1)作为连接者。

6. 一端使用nc -l 127.0.0.1 6000启动监听程序，另一端使用nc 127.0.0.1 6000 启动连接程序，则两端能互相看到对方的输入。

ctrl+c可以退出监听。

一端使用nc -l 127.0.0.1 6000启动监听程序，另一端运行自己的程序，则可以被监听到。

7. 补充书籍与网站

beej's Guide to Network Programming(2016)
UNIX Network Programming(1990) by W Richard Stevens
https://www.kernel.org/doc/man-pages Michael kerrish维护，网站分为8个部分

二、代码实践

1.连接者程序(不添加参数)

一端使用nc -l 127.0.0.1 6004启动监听程序，另一端使用tcc -run MyConnect.c 发送消息，则可以被监听到。

```
#include "stdio.h"
#include "sys/socket.h"
#include "arpa/inet.h"

void main()
{
  int s ;
  struct sockaddr_in a;

  s = socket(2,1,0);//2表示互联网，1表示tcp，0表示协议
  if (s == -1)//s== -1表示套接字失败
    exit(1);

  a.sin_family = 2;
  a.sin_port = htons(6004);
  a.sin_addr.s_addr = inet_addr("127.0.0.1");

  int f = connect(s, (struct sockaddr *) &a, sizeof a);//第一个参数是套接字的结果，第二个参数是连接目标，第三个参数是连接目标的大小。
  if (f == -1)
    exit(2);
  send(s,"hello ",6,0);//加空格，长度为6
  send(s,"world ",6,0);
}
```

2. 连接者程序(带参数)

一端使用nc -l 127.0.0.1 6004启动监听程序，另一端使用tcc -run MyConnect.c 127.0.0.1 6004 发送消息，则可以被监听到。

```
#include "stdio.h"    //printf() fopen() fread() fwrite() fclose()
#include "stdlib.h"   //malloc(oc() free()
#include "string.h"   //strlen() strtok()
#include "sys/socket.h"    //socket
#include "arpa/inet.h"     //socket intnet

int main (int count, char **arguments)
{
  int s ;
  struct sockaddr in a;
  if (count != 3){  //函数名也作为一个参数，所以一共需要三个参数。
    printf("error: there must be three arguments%c",10);
    exit(3);
  }

  s = socket(2,1,0);
  if (s == -1){//s==-1,unsuccessful
    exit(1);
  }

  a.sin family = 2;
  a.sin port = htons(atoi(arguments[2]));  //atoi把参数中的字符类型转换为整型
  a.sin_addr.s_addr = inet_addr(arguments[1]);  //两个点，是属性中的属性

  int f = connect(s,(struct sockaddr *) &a, sizeof a);
  if (f == -1)
    exit(2);

  send(s,"hello ",6,0);
  send(s,"world ",6,0);
}
```

3. 监听者程序

一端使用tcc -run MyListen.c 6006 运行监听者程序，另一端使用nc 127.0.0.1 6006 启动连接，则连接者的输入可以被监听者监听到。

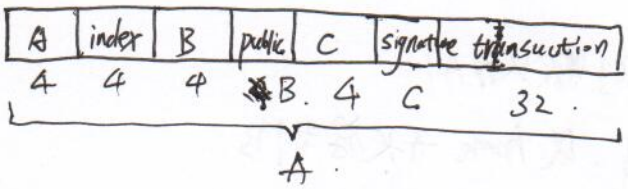
```
#include "stdio.h"
#include "sys/socket.h"
#include "arpa/inet.h"

int main(int count, char **arguments)
{
  int s,error,lenqth,opponent;
  struct sockaddr_in a;
```

```
 char buffer[1000]; //长度为1000的缓存，如果发送消息的长度超过1000，则被分割为很多个长度为1000的块。

 s = socket(2,1,0);
 if (s == -1)
  exit(1);

 a.sin_family = 2;
 a.sin_port = htons(atoi(arguments[1]));
 a.sin_addr.s_addr = inet_addr("0.0.0.0"); //0.0.0.0表示可以监听任何地址。127.0.0.1表示只可以监听自己的地址。

 error = bind(s,(struct sockaddr *) &a,sizeof a);  //bind函数的目的是设置
 if (error == -1)
 {
  exit(1);
 }
 //listen函数的目的是开始监听
 error = listen(s,1024);  //1024表示允许监听的连接者的树木的最大值。第一个被监听，之后的1023个排队等待，第1025个被立即拒绝。
 if (error == -1)
 {
  exit(1);
 }
 //accept被呼叫之后，连接才会被收到
 opponent = accept(s,0,0); //后面两个参数表示对方的地址和地址长度，0,0表示不关心对方的地址
 if (opponent == -1)
 {
  exit(1);
 }
 while(1) //不断的循环，接听到一句话之后，依然继续链接，而不会立刻断开。
 {
   memset(buffer,0,1000);
//使用缓冲之前，必须把之前的内容职位0，memset（buffer,0，1000）函数赋值为0
//因为c语言新申请到的内存不是空的，是之前的垃圾信息
   length = recv(opponent, buffer, 1000, 0);
   if (length == -1 || length == 0) //如果length的值为-1或者0的时候，则链接已经断开
  {
    break;
  }
  printf("%s",buffer);
 }
}
```

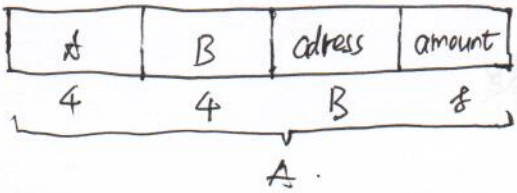
2017.10.21 周六.

数据格式:

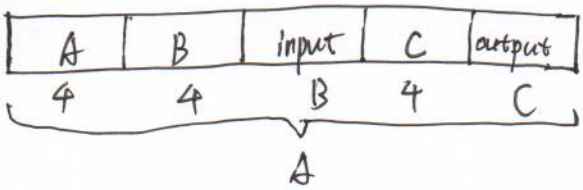
input



output



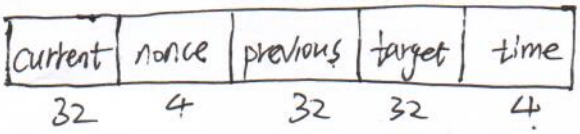
transaction



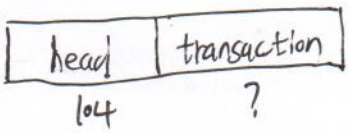
B: 所有input的总长(而不是有几个input)

C: 所有output的总长.

head.

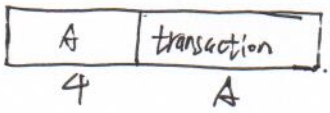


Block.



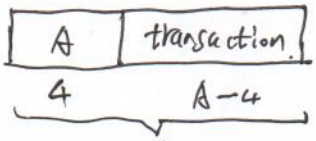
问题: 不知道transaction的长度.

解决: body.



A: transaction的长度.

or: body



A: 总长度.

问题2: input中的签名问题.

解决: 先把signature的长度置为0,(则signature不存在)

进行一次签名 把这个签名导进signature, 然后再进行一次签名.

问题3: php的ECC系统与C语言的ECC系统不兼容.

用php签名, C不能验证, 反之亦然.

因为库的设计不同. php用的是openssl库, C语言用LibTomCrypt库.

两个库的导出方式不同.

课堂: 用C写引入第三方函数. 目标: ①创造新的私钥 → ②从私钥得到公钥 → ③用私钥的签章 → ④用公钥验证签章 → ⑤私钥导出 → ⑥公钥导出

问题4: 用LibTomCrypt不能直接查看私钥,因为是生字节。只有导出私钥才能查看。而LibTomCrypt的导出和openssl的导出是不匹配的。所以用Libtomcrypt的的签章,不能被openssl验证,反之亦然.

1. 文件系统：

类 Unix：文件是只有一个根的，各个不同的盘都在同一个根之下

/ A，选 A 文件作为根目录，则新建文件时默认为 A。

/home B， 默认为 A，以/home 开头存到 B 盘之下。

/home/chengshuzhi ，默认为 A，以/home/chengshuzhi 开头存 C 盘之下。

windows 文件分为各个硬盘，A，B，C。

normal file 正常文件（包括执行文件和非执行文件（文档））

folder 文件夹，卷宗

pipe 管道

domain socket 区域套接，不能上网，像文件一样可以打开删除

dence driver node 驱动设备节点

2. 编译器的作用，把非执行的文件转化为执行的文件。执行文件的类型：

exe，可以直接运行，在 Linux 下无后缀

dll，半成品，不可以直接运行的一套函数，在程序运行中可以引用，

在 Linux 下后缀为.so

lib，需要重编译，不能在动态环境下直接运行，在 Linux 下无后缀.a

obj，半成品，编译一个程序，但是还没有和别的程序连接，还不能运行。

在 Linux 下后缀.o

.a 包含很多.o，.so 是加工之后的.a，exe 是再加工之后的.so。

```c
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "tomcrypt.h" //libtomcrypt 函数库，不能查看生字节

char *get_file(char *path){

        FILE *file;
        int length;
        char *contents;

        file = fopen(path,"r+");
        fseek(file,0,SEEK_END); //跳转到文件结尾
        length = ftell(file); //给出文件长度
        fseek(file,0,SEEK_SET); //再跳回文件开头.

        contents = malloc(length+1);
        fread(contents,1,length,file);

        fclose(file);
        return contents;

}

void main()  //给对文本进行 Hash 加密
{
        char *input, *output;
        int length;

        output = malloc(1000); //分配1000字节的内存
        length = 1000;
        input = get_file("essay.txt"); //从文件中输入.

        register_hash(&sha256_desc); //激活哈希函数，必须!
        hash_memory(find_hash("sha256"),input ,strlen(input),output,&length);

        //printf("%s",output);
        printf("%d%c",length,10); //长度
        printf("%02x%c",output[0],10); //第一个字符. //printf("%s%c",output,10);此时输出为生字节.
}
```

十六进制.

系统呼叫：open. close. read. write
（第一方函数）

标准函数：fopen fclose. fread. fwrite.
（第二方函数）

执行命令.

```
tcc -c program.c -o program.o
tcc program.o libtomcrypt.a -o program.elf
./program.elf
```

系统工具 sha256sum
sha256sum Essay.txt
结果：cca7322 ...

```c
#define TFM_DESC //必需!!!
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "tomcrypt.h"
#include "tfm.h"
char *get_file(char *path){

        FILE *file;
        int length;
        char *contents;

        file = fopen(path,"r+");
        fseek(file,0,SEEK_END);
        length = ftell(file);
        fseek(file,0,SEEK_SET);

        contents = malloc(length+1);
        fread(contents,1,length,file);

        fclose(file);
        return contents;

}

void main()
{
        char *input, *output;
        int length;

        prng_state random; //随机生产器的状态.
        ecc_key key;       //保存私钥和公钥,合并到一个数据结构中.
        ltc_mp = tfm_desc;

        output = malloc(1000);
        length = 1000;
        input = get_file("essay.txt");

        register_prng(&sprng_desc); //激活函数.
        register_hash(&sha256_desc);
        //hash_memory(find_hash("sha256"),input ,strlen(input),output,&length);

        char *buffer;
        buffer = malloc(1000);
        length = 1000;

        ecc_make_key(&random,find_prng("sprng"),32,&key);
        ecc_export(buffer,&length,PK_PUBLIC,&key); //导出到buffer中.
        //ecc_ansi_x963_export(&key,buffer,&length);

        //printf("%s",output);
        printf("%d%c",length,10); //.
        printf("%s%c",buffer,10);
        //printf("%02x%c",output[0],10);
}
```

保存随机生成器的状态.
随机生成器(系统的)
保存位置.
(长度为77)
这种导出方式是对节,不能打印
//标准化之后的生字节(长度为65),这里是公钥.

执行命令:

```
tcc -c program.c -o program.o
tcc program.o libtomcrypt.a libtfm.a -o program.elf
./program.elf
```

```c
#define TFM_DESC
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "tomcrypt.h"
#include "tfm.h"
char *get_file(char *path){

        FILE *file;
        int length;
        char *contents;

        file = fopen(path,"r+");
        fseek(file,0,SEEK_END);
        length = ftell(file);
        fseek(file,0,SEEK_SET);

        contents = malloc(length+1);
        fread(contents,1,length,file);

        fclose(file);
        return contents;

}


unsigned char *get_private(unsigned int *length)
{
        prng_state random; // prng的状态(随机生字器).
        ecc_key key;
        unsigned char *buffer;

        buffer = malloc(*length =1000);
        ecc_make_key(&random,find_prng("sprng"),32,&key);
        ecc_export(buffer,length,PK_PRIVATE,&key);

        return buffer;
}
unsigned char *get_public(unsigned *length, unsigned char *private)
{
        ecc_key key;
        unsigned char *buffer;

        ecc_import(private,*length,&key);
        buffer = malloc(*length = 1000);
        ecc_export(buffer,length,PK_PUBLIC, &key);

        return buffer;
}

unsigned char *get_input(unsigned int *total)
{
        //A index B pulbic C signature transaction

        int length;
        int *x;
        unsigned char  *y; //造两个指针.

        char *public,*private, *complete;

        private = get_private(&length);
        public = get_public(&length,private);

        complete = malloc(*total = 48+length);
```

```
        x = malloc(4); //A
        *x = 48 + length;
        memcpy(complete,x,4);

        x = malloc(4); //index
        *x = 0;
        memcpy(complete+4,x,4);

        x = malloc(4); //B
        *x = length;
        memcpy(complete+8,x,4);

        y = malloc(length); //public
        memcpy(y,public,length);
        memcpy(complete+12,y, length);

        x = malloc(4); //C,c=0,signature not exist
        *x = 0;
        memcpy(complete+12+length, x,4);

        y = malloc(32); //transaction
        *y = 0;
        memcpy(complete+16+length,y,32);

        return complete;
}
void main()
{
        int length;

        prng_state random;
        ecc_key key;
        ltc_mp = tfm_desc;

        register_prng(&sprng_desc);
        register_hash(&sha256_desc);
        //hash_memory(find_hash("sha256"),input ,strlen(input),output,&length);

//      unsigned char *public, *private;
//      private = get_private(&length);
//      printf("%d%c",length,10);
//      public = get_public(&length,private);
//      printf("%d%c",length,10);

        unsigned char *input;
        input = get_input(&length);
//      printf("%d%c",length,10);
        write(1,input,length);
}
```

A	index	B	public	C	signature	transaction
4	4	4	length	4	0	32

total: 48 + length.

//两个激活函数.

//输出private长度

//输出public长度.

//输出长度

//写文件.

执行命令: ./program.elf > input.

xxd input.

```
0000000: 7e00 0000 0000 0000 4e00 0000 304c 0302  ~.......N...0L..
0000010: 0700 0201 2002 2009 86a4 74a0 2b62 51a8  .... . ...t.+bQ.
0000020: 0d71 5131 547b 06c9 b19d 9a32 935a 6074  .qQ1T{.....2.Z`t
0000030: 9723 66f3 9f22 f602 2100 c0b5 b8c7 3b71  .#f.."..!.....;q
0000040: 2283 2457 53d7 800c fa56 0a5f 39c0 7a7c  ".$WS....V._9.z|
0000050: aa2a c4f5 600a 70c1 3954 0000 0000 0000  .*..`.p.9T......
0000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000070: 0000 0000 0000 0000 0000 0000 0000       ..............
```

7e=126. A.

index

4e=78. B.

B=78, public的长度为78. 48+78=126

C=0. signature不存.

transaction, 长度为32.

2017.10.22.. 周日. 密码学

一、什么是哈希

1. 定义：Hk(x)=y,(Hk 属于 K，x 属于 X，y 属于 Y)

Note：X 是无限大的，而 Y 是有限的。

哈希家族：MD4、MD5、SHA-0、SHA-1、SHA-2、SHA-3。

2. 结构：

（1）SHA-256：Merkle Damgrad。

M：This is the Message 00100

特点：

(1)输出长度固定。

(2)哈希速度快。

(3)防碰撞。如果输出信息的长度为 n 的时候，则防碰撞的级别是 1/（2 的(n/2)次方）

则 SHA-256 的输出长度为 256，则需要 2 的 128 次方的计算量才能找到碰撞。

（2）SHA-3：Keccak。

M： This is the Message 00100

3. 安全性：

（1）原像稳固：给定哈希函数和 y，很难找到 x，使得 H(x)=y。

（2）第二原像稳固：给定哈希函数和 x，很难找到另外一个 x1 不等于 x，使得 H(x1)=H(x)。

（3）碰撞稳固：给定哈希函数，很难找到满足 H(x)=H(x1)的二元组(x,x1)。

4. 特点：

（1）任意长输入，对应一个固定长的输出。

public key（公钥）对应 512 比特，H(pk)对应 256 比特。

应用：地址的生成。原因：减少数据长度，从而节省内存。

（2）速度快。

应用：挖矿哈希（验证时速度快）挖矿算法的条件：难于解决，易于验证。

比如，POW，H(x||Nonce)小于 D(256 比特)。最大难度为 2 的 256 次方，现在的难度为 2 的 55 次方。

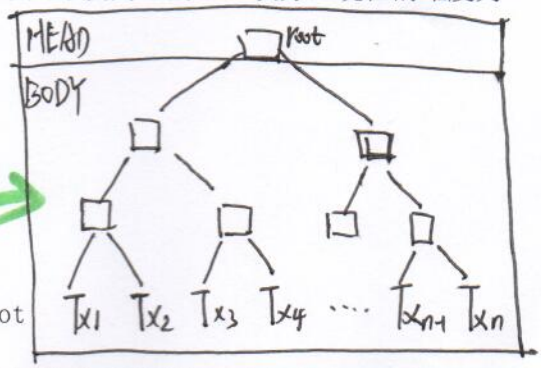
（3）对于任意两个 x 和 x1，H(x)不等于 H(x1)。

应用：连接问题

（4）数据完整性

应用：SPV 简单支付验证

Merkle Root，不需要下载整个区块体。



二、为什么采用哈希

三、哈希的应用

1. Tx(交易 Transaction)的哈希和 Merkle Root

2. 地址生成

3. 挖矿哈希(上一区块的哈希值、时间戳、交易的根哈希等) hash(x||Nonce)<target

4. 连接

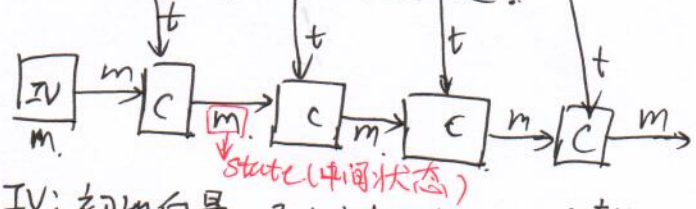
四、哈希的实现

```
#include "tomcrypt.h"
int main (int count, char **arguments){
        hash_state md;
        unsigned char *in = "Hello,world",out[16];
        //8jinzhi? //
        md5_init(&md);
        md5_process(&md,in,strlen(in));
        md5_done(&md,out);
        for (int i =0;  i<32; i++){
        //shuchu wei 4 jinzhi suoyi xuyao 32 ge zijie.? ?
                printf("%x",out[i]);
        }
        printf("\n");
        return 0;
}
```

M: This is | the message | 00|100000

SHA-256:

每T字段大个字节. 不足补零.



IV: 初始向量, Initial Vector. 长度为 (m+t)个0的字符串.

C: 压缩函数 compress. $(1)^{m+t} \to (1)^m$ t次.

Hash: 十六进制. 一个字节表示4个比特.

所以长为256比特的串. 输出字节长度为 256/4 = 64.

SHA-3:

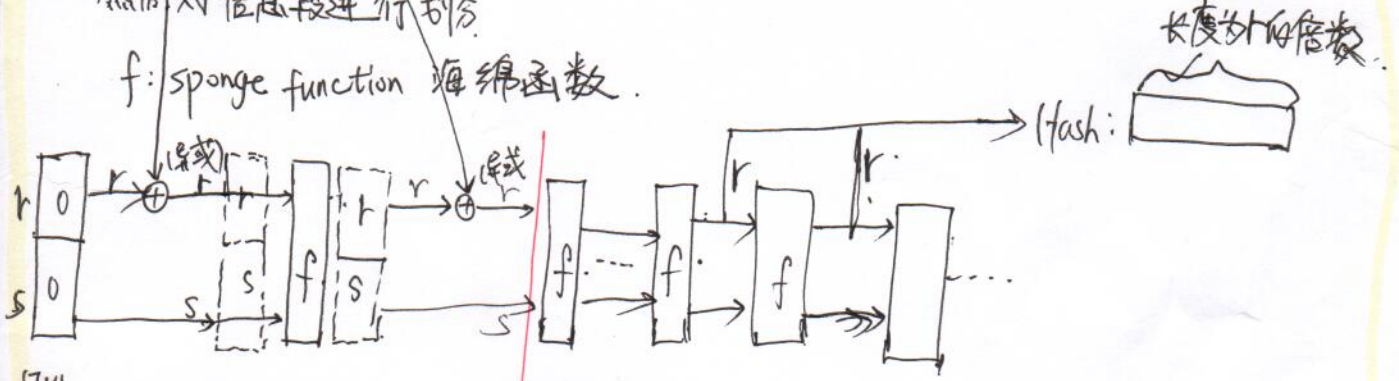
M: This is | the Message 00|100...000

首先给定初始向量 IV(Initial Vector), 一般固定为0

然后对信息段进行划分.

f: sponge function 海绵函数.

长度为r的倍数.



(IV)

(任意长输入 ⟶ 任意长输出)

absorbing 吸收阶段 | squeeting 压榨阶段.

2017.10.25.

目标：每个人生产自己的私钥，给每个人发送货币。

验证：① 格式：

② 输入：— 此输入是否存在
— 此输入的签章是否有效。

③ 输出：

④ 交易平衡：— 输入是否多于或者等于输出的总额。

⑤ 第一个交易：

⑥ 头部：— 交易的摘要是否正确。
— 劳动证明是否正确。

第二个及以上交易。

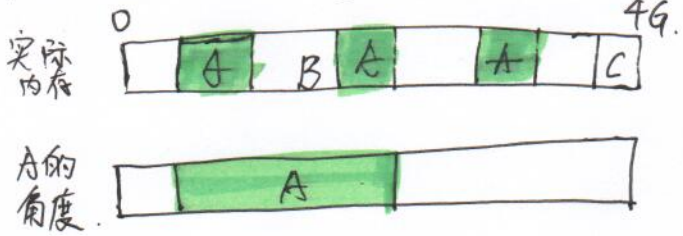
Note：必须先验证其他交易再验证第一个交易。

挖矿存在的意义：① 防止一个人在短时间内创造大量虚假的区块。
② 维持竞争性。

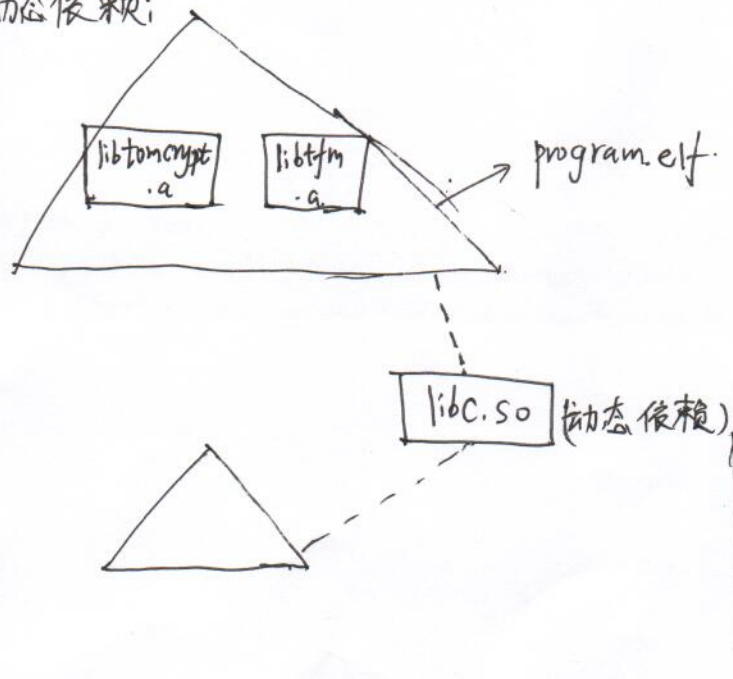
ecc_export / ecc_import：生字节，不能读。

base 64：将生字节转换为 ASCII。

虚拟内存 Virtual Memory



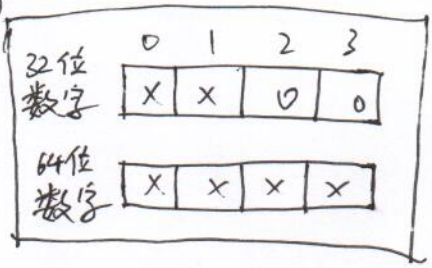
动态依赖：



优点：

① 一个文件可被多个程序依赖

② 仓库升级只需要替换一个文件。



作业：①创作自己完整的区块。
创币奖给自己。

②创第二个区块，以第一个为前任。
把交易1：把钱发给任何一个人。

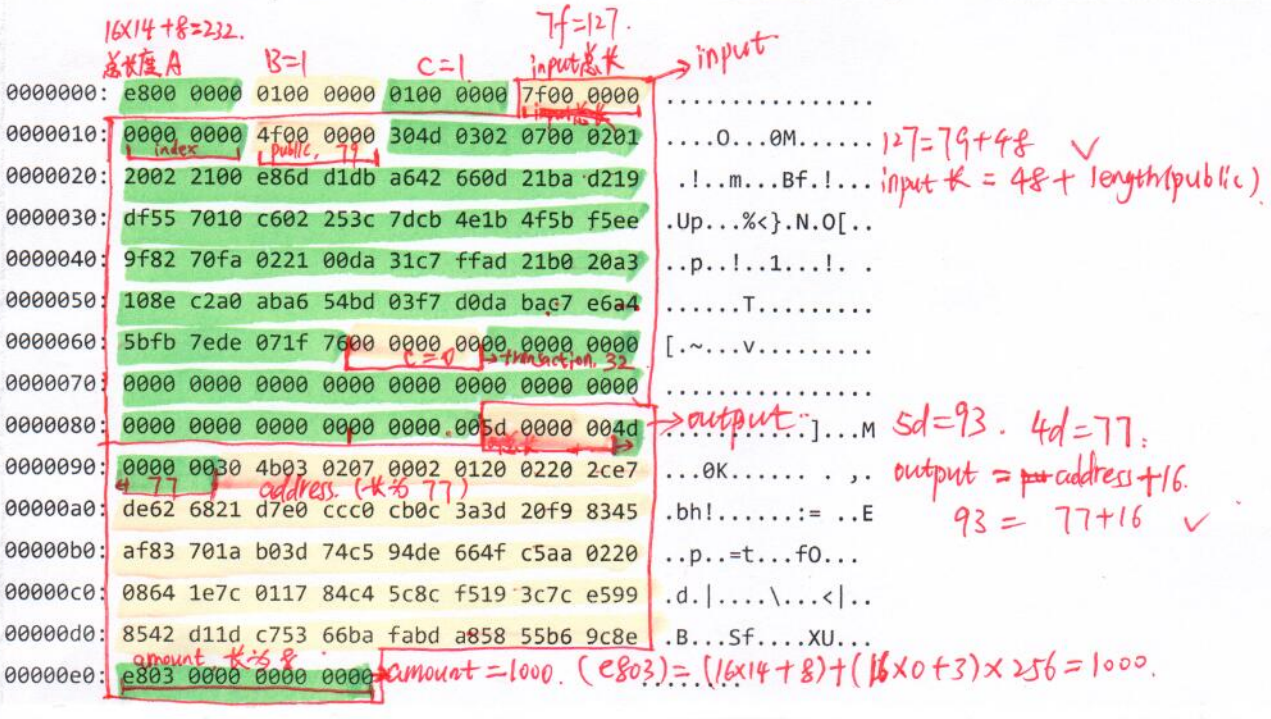
Note：private key 和 public key 是以 base64 存储的，
需要再转回生字节才能被 libtom crypt 查看。

如果把 base64 存在文件里，文本编修器会自动加入一个换行字符，
有可能会导致函数崩溃。

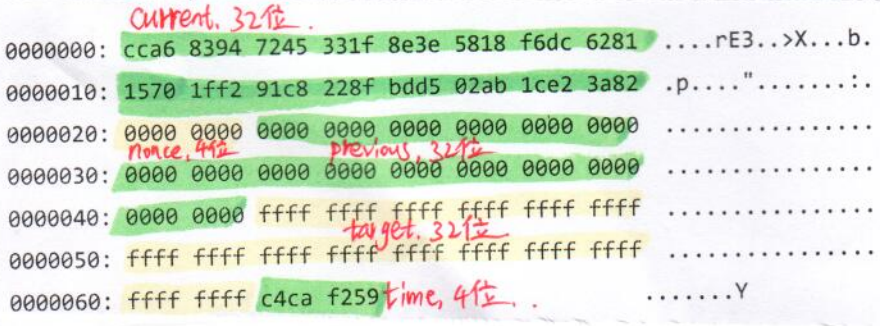
所以，用上节课写的 get_file 函数来读
如果最后一个字符是 10.(换行)，则应该删去。

或者：不把 private key 和 public key 存在文件里，
直接用 char *public = "" 存。

△：transaction 分析：



△：head 分析



目的：将 private 和 public 保存为 base64_encode 形式。

```c
#define TFM_DESC
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "tfm.h"
#include "tomcrypt.h"
unsigned char *get_private(unsigned int *length)
{
   prng_state random;
   ecc_key key;
   unsigned char *buffer;
   buffer = malloc(*length = 1000);
   ecc_make_key(&random, find_prng("sprng"), 32, &key);
   ecc_export(buffer, length, PK_PRIVATE, &key);
   return buffer;
}
unsigned char *get_public(unsigned int *length, unsigned char *private)
{
   ecc_key key;
   unsigned char *buffer;
   ecc_import(private, *length, &key);
   buffer = malloc(*length = 1000);
   ecc_export(buffer, length, PK_PUBLIC, &key);
   return buffer;
}
void main()
{
   unsigned char *private, *public, *buffer;
   unsigned int length_private, length_public, length_buffer;
   ltc_mp = tfm_desc;
   register_prng(&sprng_desc);
   register_hash(&sha256_desc);
   private = get_private(&length_private);
   length_public = length_private;
   public = get_public(&length_public, private);

   buffer = malloc(1000);
   memset(buffer, 0, 1000);
   base64_encode(private, length_private, buffer, &length_buffer);
   printf("%s%c%c", buffer, 10, 10);
   memset(buffer, 0, 1000);
   base64_encode(public, length_public, buffer, &length_buffer);
   printf("%s%c", buffer, 10);
}
```

将生字节转换为 ASCII

//C分配内存时，不会默认置为0.

printf("%d%c%d%c", length_buffer, 10, length_private, 10); 152, 113

printf("%d%c%d%c", length_buffer, 10, length_public, 10); 104, 78

```
unsigned char *get_output(unsigned int *total)
{
        unsigned char *private,*public,*complete,*y;
        unsigned int length,*x,z;
        private = get_private(&length);
        public = get_public(&length,private);          //A,B,address,amount  //4+4+B+8
        complete = malloc(*total = 16+length);
        x = malloc(4);  //A
        *x = 16 + length;
        memcpy(complete,x,4);
        x = malloc(4);  //B
        *x = length;
        memcpy(complete+4,x,4);
        y = malloc(length);  //address, 暂时用public表示address.
        memcpy(y,public,length);
        memcpy(complete+8,y,length);
        y = malloc(8);
        z = 1000;//用4字节的数 两个 (32位) ,表示一个8字节的数 (64位)
        memcpy(y,&z,4);
        memset(y+4,0,4);//后4个字节为0.
        memcpy(complete+8+length,y,8);
        return complete;
}
unsigned char *get_transaction(unsigned int *total)
{
        //A B C input output  //B number of input C number of output //A total length
        unsigned char *input,*output,*complete,*y;
        unsigned int length_input,length_output,*x;
        input = get_input(&length_input);
        output = get_output(&length_output);
        complete = malloc(*total = 12+length_input+length_output);
        x = malloc(4);  //A
        *x = 12 + length_input + length_output;
        memcpy(complete,x,4);
        x = malloc(4);        *x = 1;  //B. 只有一个输入.
        memcpy(complete+4,x,4);
        x = malloc(4);        *x = 1;  //C 只有一个输出.
        memcpy(complete+8,x,4);
        y =  malloc(length_input);  //input.
        memcpy(y,input,length_input);
        memcpy(complete+12,y,length_input);
        y = malloc(length_output);  //output.
        memcpy(y,output,length_output);
        memcpy(complete+12+length_input,y,length_output);
        return complete;
}
```

```
unsigned char *get_head(unsigned char *root)
{
        unsigned char *complete, *y;
        unsigned int *x;
        complete = malloc(104);//current nonce previous target time
        y = malloc(32);
        memcpy(y,root,32);
        memcpy(complete, y ,32);
        x = malloc(4);
        *x = 0;
        memcpy(complete+32,x,4);
        y = malloc(32);
        memset(y,0,32);
        memcpy(complete+36,y,32);
        y = malloc(32);
        memset(y,255,32);
        memcpy(complete+68,y,32);
        x = malloc(4);
        *x = time(0);
        memcpy(complete+100,x,4);
        return complete;
}
void main()
{
    char *head, *input, *output,*transaction,*root;
    int length_input,length_output,length_transaction,length_root;
    prng_state random;
    ecc_key key;
    ltc_mp = tfm_desc;
    register_prng(&sprng_desc);
    register_hash(&sha256_desc);
//      input = get_input(&length_input);
//      output = get_output(&length_output);
    transaction = get_transaction(&length_transaction);
//      printf("%d%c",length_input,10);//printf("%d%c",length_output,10);
//      printf("%d%c",length_transaction,10);
//      write(1,transaction,length_transaction);
    root = malloc(length_root = 32);
    hash_memory(find_hash("sha256"),transaction,length_transaction,root,&length_root);
    head = get_head(root);
    write (1,head,104);
}
```

head长度固定，所以不需要保存length.

root: Merkle树的根，transaction的Hash值.

// current.

// nonce.

// previous

// target 越大，难度越小

// target = 255 时，难度最小.

// time.

// 此时，再生成的input和output与transaction函数中的

// 长度不一定相同，不一定满足 length_transaction = 12 + output + input.

// 写出，可用 ./program.elf > transaction 写入文件

// 用 wc transaction 查长度 用 xxd transaction 查看内容.

// transaction的sha256值为 Merkle Root.

2017.10.28

一、什么是 ECDSA?

1.Elliptic Curve Digital Signature Algorithm

2.ECC 与 ECDSA
ECC：椭圆曲线密码，用于加密解密。pk 用于加密 encrypt，sk 用于解密 decrypt。
ECDSA：椭圆曲线数字签名算法。用于签名和验证。sk 用于签名 signature，pk 用于验证 verify。

3.签名的发展
Elgamal：DSA，不够安全
RSA： RSA_DSA，足够安全，但不够效率。
ECC： ECDSA，平衡安全与效率。
Lattice（格）：暂时还不够成熟。

4.定义：
K={（E,p,q,A,m,B）,B=mA}
比特币采用的曲线：secp256k1。y^2=(x^3+7)mod p
P 是一个非常大的素数，p=2^256-2^32-2^9-2^8-2^7-2^6-2^4-1。
满足（4a^3+27b!=0）mod p 的曲线才安全。
Lasses 定理。
A=（G），由 G 生成的。A 的集合里有 q 个元素。
q：私钥空间。
B： sk = m，pk = B.

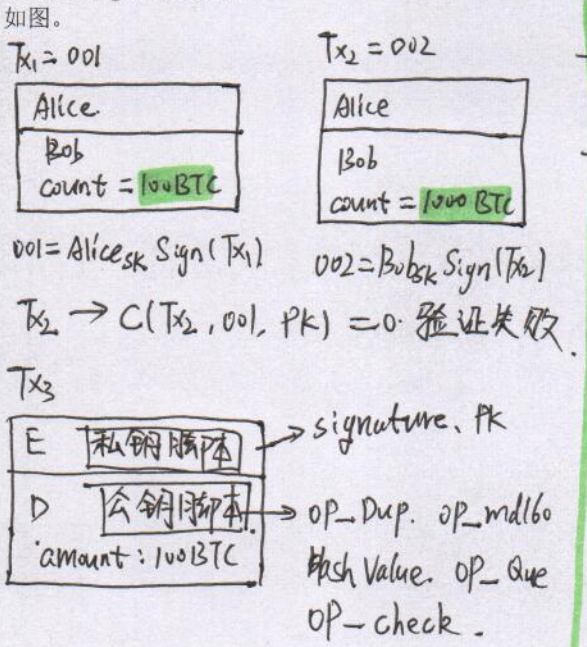
5.安全性
公钥的长度 len（B）=256bit，其安全性达到 128 位。安全性越高，效率越低。
生日悖论：Hash（x），256 位，需要计算 2 的 128 次方次的计算，才有大于 50%的概率找到碰撞。

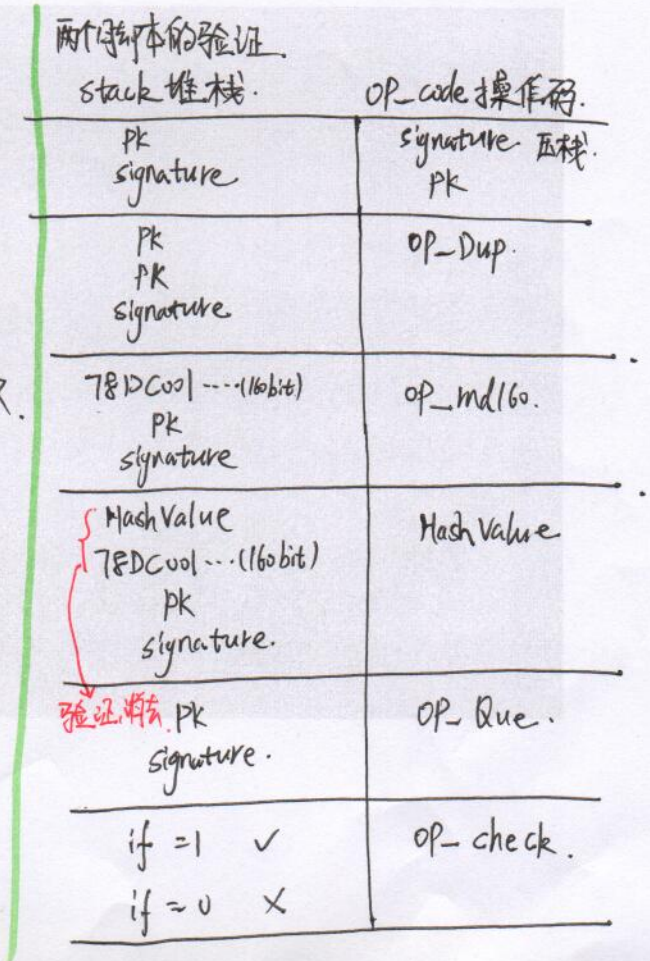
二、为什么采用 ECDSA?

1.高效
2.安全，同等安全性，ECDSA 的效率最高。
3.确权

三、ECDSA 在 BC 中的应用?

1.生成私钥 sk（ECC）
2.签名 sign（ECDSA）
如图。

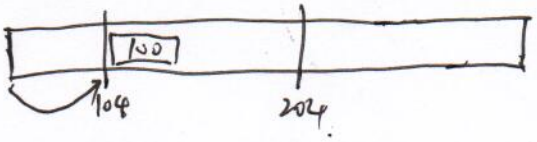




四、ECDSA 的编程实现?

2017.12.29.

指针跳转：X+104+(*)+(*)+…



X=X+104

X=X+(*X) Δ：某个交易的长度，可通过读交易前四个字节获得。

X=X+(*X)　　获得。

⋮

X+104+[*(X+104)]+[*[(X+104)+*(X+104)]]

身份：

区块 block 的身份：block → sha256(head)

交易 transaction 的身份：transaction → sha256(transaction)

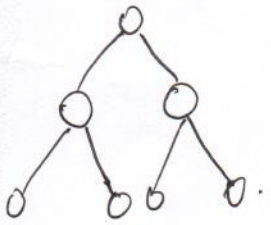
身体 body 的身份：body → Markle ~~Root~~ Tree.

头部 head 的身份：与 block 的身份相同。

输入 input 的身份：

输出 output 的身份：

Markle Tree：



若为奇数：则把最后一个 X_{2n} 复制一个，为偶数个。

若为偶数：则 两两两个合在一起 进行 SHA256.

逐层递归。

get_markle_root(x) = sha256(x)

只有一个交易，Markle 根 就是对这个交易的哈希。

get_markle_root(X_0, X_1, … X_{2n}) = get_Merkle_root(X_0, X_1, … X_{2n}, X_{2n})

get_Merkle_root(X_0, X_1, … X_{2n-1}) = get_Merkle_root(~~X_0, X_1~~ [SHA256(X_0+X_1), SHA256(X_2+X_3)…]

递归 function(x)

```
if (count(x)==1), return;
if (count(x)%2), x.push[x, (count(x)-1)];
while (i < count(x)). SHA256[X(i)+X(i+1)]; i=i+2;
```

alias nano = " nano -w -x -E -T3 -I -5 "

缩进量为3.

-5 → 向下滚动时，每次滚动一行

-I → 不使用系统管理者的设置.

-E → 用空格代替缩进

-x → 关闭说明

-w → 允许换行

* 等号前后不可有空格

Merkle Root: Merkle Tree 最上面的节点

get_Merkle_Root( ){ } 两种方式实现

get_lines_file($file) { }

作业1: 引入 digests.txt 文件，写 Merkle_Root，得出结果应该与老师的相同

"7a33512c337107073020766abc1c ...."

INPUT := A, INDEX, B, PUBLIC, C, SIGNATURE, TRANSACTION

A = INPUT

B = INPUT + 8

C = INPUT + 12 + *B.

SIGNATURE = input. + 16 + *B.

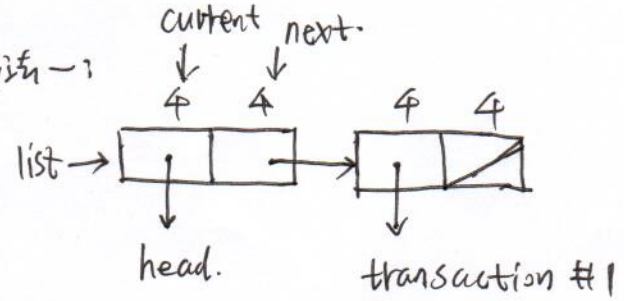
memcpy (destination, signature, *C)

OUTPUT := A, B, address, amount

TRANSACTION := A, B, C, INPUT*, output*

HEAD x = CURRENT, NONCE, previous, target, TIME

方法一:



方法二:

struct { current, next }

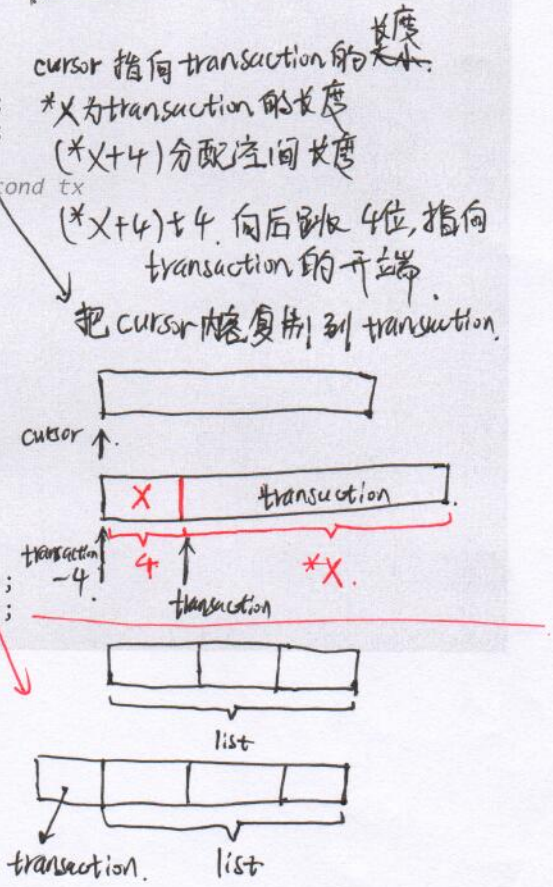
① Pascal string. ② linked list.

③ sharing a block into transactions

```php
<?php

function get_lines_file($file)//读取文件
{
   $result = array();
   $contents = file_get_contents($file);
   $contents = explode(chr(10),$contents); //用换行符分割.
   foreach($contents as $item){
      $item = str_replace(chr(13),'',$item); 去掉chr(13)
      if (strlen($item) > 0){
         array_push($result,$item);
      }
   }
   return $result;
}

/*$digests = array();
foreach (range(1,100) as $number)
   $digests[] = hash('sha256',rand());//100个随机卷文
//var_dump($digests);
//var_dump(get_Merkle_root($digests));

$c = '';
foreach($digests as $b)
   $c .= $b.chr(10); //加换行符号;
echo $c;*/

$x = get_lines_file('digest.txt');
var_dump($x);

function get_Merkle_root($digests)
{
   if (count($digests)==1){
      echo sprintf('case #1:%d%c',count($digests),10);
      return $digests[0];
   }

   if (count($digests) %2){//奇数.
      echo sprintf('case #2:%d%c',count($digests),10);
      $item = $digests[count($digests)-1];
      array_push($digests,$item);//push.把最后一个item复制.
      return get_Merkle_root($digests);
   }

   echo sprintf('case #3: %d%c',count($digests),10);
   for ($i = 0,$s = count($digests) / 2;$i < $s; $i++){
      $a = array_shift($digests);//从左侧取(pip是从右侧取)
      $b = array_shift($digests);
      array_push($digests,hash('sha256',$a.$b));//从右侧插入.
   }

   return get_Merkle_root($digests);

   /*$reduced = array();
   $i = 0;
   while($i < count($digests)){//新创建一个数组reduced存放.
      $merged = sprintf('%s%s',$digests[i],$digests[i+1]);
      array_push($reduced,hash('sha256',$merged));
      $i += 2;
   }
   return get_Merkle_root($reduced);*/

}
```

把所有运行结果存入 digest.txt 中
. php read.php > digest.txt 命令.

```c
#include "stdio.h"
#include "stdlib.h"
#include "string.h"

char *get_file(char *path , unsigned int *length){

        FILE *file;
        char *contents;

        file = fopen(path,"r+");
        fseek(file,0,SEEK_END);
        *length = ftell(file);
        fseek(file,0,SEEK_SET);

        contents = malloc(*length);
        fread(contents,1,*length,file);

        fclose(file);
        return contents;

}

unsigned int slice (unsigned char *cursor, unsigned int length)
{       //切割文件并存放到list中
        unsigned char *limit,*transaction;
        unsigned int *X;
        void *list, *link;

        limit = cursor + length; //底限(最尾部指向)
        cursor += 104;

        while(cursor<limit){
                X = (unsigned int *)cursor;
                transaction = malloc(*X +4)+4;
                memcpy(transaction,cursor,*x);
                memcpy(transaction-4,X,4);
                cursor += *X;//jump to the second tx

                link = malloc(8);
                memcpy(link,transaction,4);
                memcpy(link+4,list,4);

                list = link;
        }
}

int main(int count, char **arguments)
{
        unsigned char *block;
        unsigned int length;
        block = get_file(arguments[1],&length);
        printf("%d%c",slice(block, length),10);
}
```



```c
#include "stdio.h"
#include "stdlib.h"
#include "string.h"

int count(void *cursor)
{
        int count;
        void *Y;
        count = 0;

        while(cursor !=0)
        {
                count++;
                Y = cursor + 4;
                cursor = (void *)*Y;
        }
}

void *build_pair(void *left, void *right) //合二为一..
{
        void *pair;
        pair = malloc(8);
        memcpy(pair,&left,4);
        memcpy(pair+4,&right,4);
        return pair;
}


int scan(char *source) //返回大小.
{
        int *size;
        size = (int *)source;
        return *size;
}

char measure(char *source)
{
        int *length;
        length = (int *)(source-4);
        return *length;
}

char *build(char *source, int length) //带有长度的字符串.
{
        char *destination;
        destination = malloc(length+4)+4;
        memcpy(destination,source,length);
        memcpy(destination-4,&length,4);

        return destination;
}
char *get_file(char *path , unsigned int *length){//读文件!

        FILE *file;
        char *contents;

        file = fopen(path,"r+");
        fseek(file,0,SEEK_END);
        *length = ftell(file);
        fseek(file,0,SEEK_SET);

        contents = malloc(*length);
        fread(contents,1,*length,file);

        fclose(file);
        return contents;
```

```
}

void *slice (unsigned char *cursor, unsigned int length)
{
        char *limit;
        void *list;

        limit = cursor +length;
        list = 0;
        list = build_pair(build(cursor,104),list);
        cursor += 104;
        while(cursor <limit){
                char *x = build(cursor,scan(cursor));
                printf("valid pointer:%d%c",x,10);
                list = build_pair(x, list);
                cursor += scan(cursor);
        }
        return list;
}
uint64_t get_output_amount(unsigned char *output)
{
        int *B;
        uint64_t amount;

        B = (int *)(output+4);

        memcpy(&amount,output+8+ *B,8);
        return amount;
}


int verify(char *transaction)
{
        int *A,*B,*C;
        char *cursor;
        int count;

        A = (int *) transaction;
        B = (int *)(transaction +4);
        C = (int *)(transaction + 8);

        uint64_t input_total,output_total;
        cursor = transaction +12;
        count = *B;
        /*
        while(count --){
                input_total += get_input_total(cursor);
                cursor += scan(cursor);

        }
        */

        while(count--)
                cursor += scan(cursor);

        count = *C;

        while(count --){
                output_total += get_output_amount(cursor);
                cursor += scan(cursor);
        }

        printf("number of outputs:%d%c", *C,10);
        printf("volume of outputs:%d%c",output_total,10);

        return 0 ;
```

```
}
int main(int count, char **arguments)
{
        char *block;
        int length;
        void *list,*transaction,*t;

        char *a;

        block = get_file(arguments[1],&length);//读出block内容.
        list = slice(block,length);//分割文件并存到list中

        printf("%d%c",list,10);;
        list +=4;//前4位是长度.向后跳4位是transaction内容.

        list = (void *) *list;
        printf("%d%c",list,10);
        list = (void *) *list;
        printf("%d%c",list,10);

        //verify(list+4);
}
int main(int count, char **arguments)
{
    unsigned char *block, *data;
    long length;
    void *list;
    char *a;

    block = get_file(arguments[1], &length);
    list = slice(block, length);

    while ((int)list != 0 ) {
        data = (unsigned char *) *(int *) list;
        printf("value: %d%c", (int)data, 10);
        list = (void *) *(int *)(list + 4);
        printf("list: %d%c", (int)list, 10);
    }

    // verify(list + 4);
}
```