# 精通比特币二加密相关

**说明：有些技术内容是从网上搜索得到。**

# 源码的安装编译

尽量在LINUX上安装。如果要看钱包源码并运行，应该在有界面的操作系统上。有一篇安装文档。

更具体的看相关安装流程和实际演示。

不过有一些细节需要注意：

1、不用配置启动也可以启动

2、有一命令已经过时，比如书中的 getinfo,已经推荐使用getblockchaininfo

# 地址和密钥

**1、比特币地址种类**：

1）由公钥产生。

一般情况下，比特币地址由一个公钥生成并对应于这个公钥。

2）支付对象，比如脚本

这样做的好处其实就把地址抽象出来，应用的面会更广泛。

**2、加密体系：**

加密本质上就是不让非相关的人看到具体的信息内容。可分为对称和分对称加密算法。

1. 对称加密：

采用单钥密码系统的加密方法，同一个密钥可以加密也可以解密的。

其加密常用的算法主要有：DES、IDEA、RC2、RC4、SKIPJACK、RC5、AES算法等。

它们的优势在于简单易用，特别对大数据块的加密，有非常好的优势，可是由于加解密使用一套密码，所以密钥必须发到各个相关单位。这就容易在分发和保管的过程中出现密码泄露。一旦密码泄露了，所有的信息就都泄露了。

举个例子，二战时的日本的密码和德国的密码都被盟国给搞定了。所以密钥的管理十分困难，风险也很高。

1. 非对称加密：

采用公开密钥系统的加密方法，加密和解密的密钥不同，但是惟一配对。主要有RSA、Elgamal、背包算法、Rabin、D-H、ECC（椭圆曲线加密算法）等。

它主要是对小块数据或者少量数据加密，因为私钥一般不用公开出去，只把公钥发出去，所以安全系数很高。

另外一个重要的应用是数字签名，利用私钥生成可信的签名，利用公钥在验证。保障不可抵赖性，这在金融体系里有非常重要的作用。

1. 数字证书：

是一个经证书授权中心数字签名的包含公开密钥拥有者信息以及公开密钥的文件。最简单的证书包含一个公开密钥、名称以及证书授权中心的数字签名。

它以数字证书为核心的加密技术(加密传输、数字签名、数字信封等安全技术)可以对网络上传输的信息进行加密和解密、数字签名和签名验证，确保网上传递信息的机密性、完整性及交易的不可抵赖性。使用了数字证书，即使您发送的信息在网上被他人截获，甚至您丢失了个人的账户、密码等信息，仍可以保证您的账户、资金安全。

1. 证书认证中心：

所谓CA（Certificate Authority）认证中心，它是采用PKI（Public Key Infrastructure）公开密钥基础架构技术，专门提供网络身份认证服务，负责签发和管理数字证书，且具有权威性和公正性的第三方信任机构。

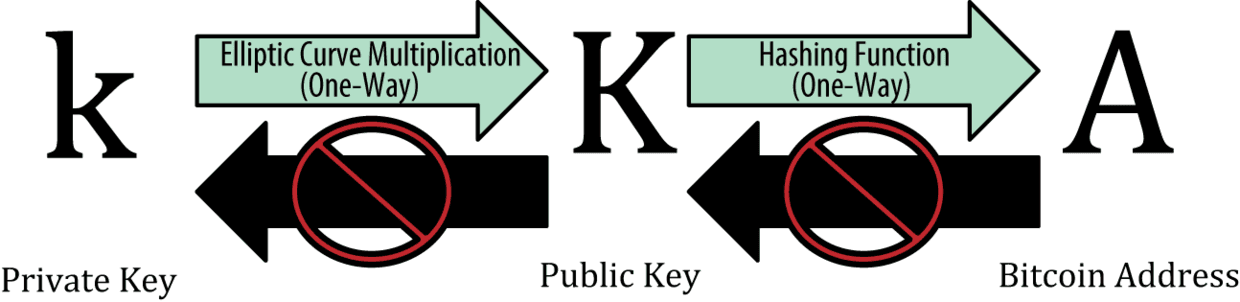
1. 加密体系：

公钥基础设施(public key infrastructure;PKI)：由公开密钥密码技术、数字证书、证书认证中心和关于公开密钥的安全策略等基本成分共同组成，管理密钥和证书的系统或平台。

**3、公钥和私钥**

在前面说过，一个钱包在生成的时候儿一般会生成N个密钥对，其中就包括N（private,public）对。

私钥（k）是一个数字，通常是随机选出的。有了私钥，我们就可以使用椭圆曲线乘法这个单向加密函数产生一个公钥（K）。有了公钥（K），我们就可以使 用一个单向加密哈希函数生成比特币地址（A）。



**私钥的产生**：私钥可以是1和n-1之间的任何数字，其中n是一个常数（n=1.158 \* 10^77，略小于2^256），并被定义为由比特币所使用的椭圆曲线的阶（见椭圆曲线密码学解释）。要生成这样的一个私钥，我们随机选择一个256位的数字，并检查它是否小于n-1。从编程的角度来看，一般是通过在一个密码学安全的随机源中取出一长串随机字节，对其使用SHA256哈希算法进行运算，这样就可以方便地产生一个256位的数字。如果运算结果小于n-1，我们就有了一个合适的私钥。否则，我们就用另一个随机数再重复一次。

需要注意的是不要自己写程序生成一个私钥，而要使用相关密码学生成工具即密码学安全的伪随机数生成器（CSPRNG），并且需要有一个来自具有足够熵值的源的种子。使用随机数发生器的程序库时，需仔细研读其文档，以确保它是加密安全的。正确实施CSPRNG是密钥安全性的关键所在。

以下是一个随机生成的私钥（k），以十六进制格式表示（256位的二进制数，以64位十六进制数显示，每个十六进制数占4位）：

1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD

看一下比特币源码中私钥的生成过程：

CPubKey CWallet::GenerateNewKey()

{…….

CKey secret;

//使用加密的伪随机数（CSPRNG）生成私钥

**secret.MakeNewKey(fCompressed);**

//由私钥生成公钥

CPubKey pubkey = secret.GetPubKey();

assert(secret.VerifyPubKey(pubkey));

………

}

void CKey::MakeNewKey(bool fCompressedIn) {

**RandAddSeedPerfmon();/**/增加随机种子

do {

//openssl产生随机数

**GetRandBytes(vch, sizeof(vch));**

} while (!Check(vch));

fValid = true;

fCompressed = fCompressedIn;

}

void RandAddSeed()

{

// Seed with CPU performance counter

int64\_t nCounter = GetPerformanceCounter();

RAND\_add(&nCounter, sizeof(nCounter), 1.5);

memory\_cleanse((void\*)&nCounter, sizeof(nCounter));

}

void RandAddSeedPerfmon()

{

RandAddSeed();

}

void GetRandBytes(unsigned char\* buf, int num)

{

if (RAND\_bytes(buf, num) != 1) {

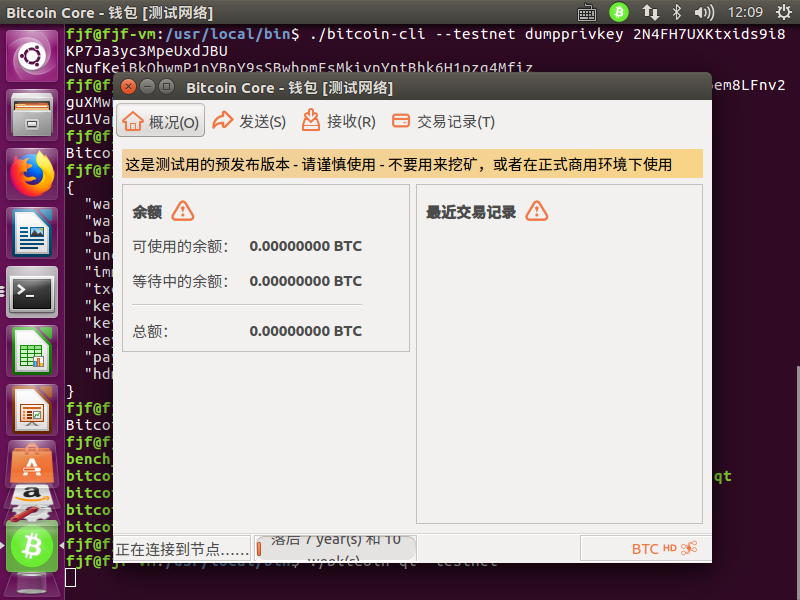
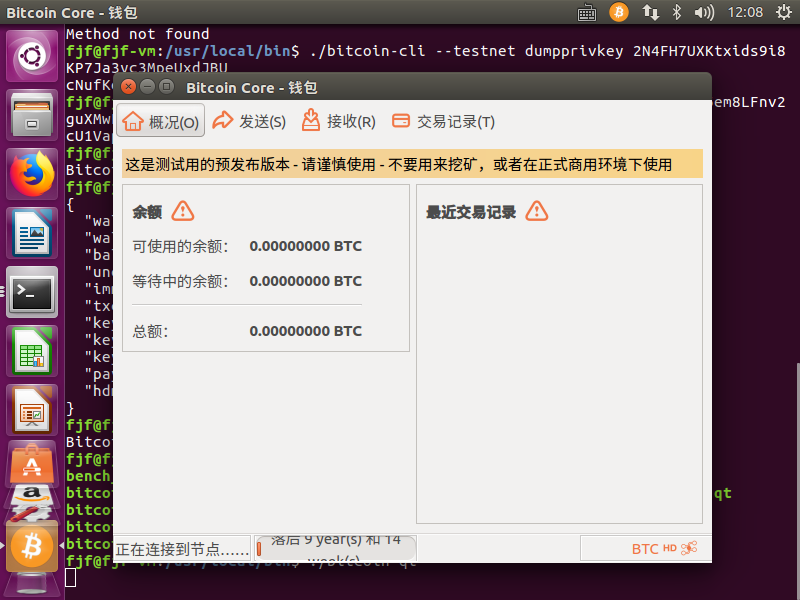
LogPrintf("%s: OpenSSL RAND\_bytes() failed with error: %s\n", \_\_func\_\_, ERR\_error\_string(ERR\_get\_error(), NULL));

assert(false);

}

}

使用比特币的客户端生成钱包的密钥对，如下图：

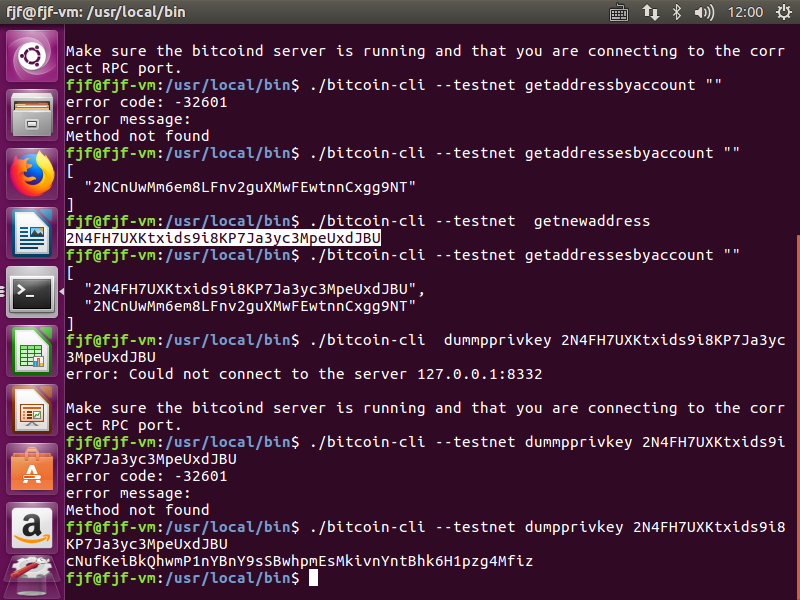


生成密钥：

下图是用：

/usr/local/bin$ ./bitcoin-cli --testnet getnewaddress

2NCnUwMm6em8LFnv2guXMwFEwtnnCxgg9NT



前面说过暴力破解的概率，这里说一说它的难度。首先举个例子：

假设地球上使用比特币的人数达到100亿，也就是10的10次方，则你猜中一个比特币私钥的概率是：。

好，我们再举两个现实中的例子你就明白有多难了。

"双色球"一等奖的中奖概率是8×10ˉ⁸。——摘自《中国福利彩票的中奖概率》

从全世界范围来看（假设全世界人口为70亿），一年中每个人遭遇雷击的概率是175万分之一（5×10ˉ⁷）。——摘自中国气象局-《遭遇雷击的概率有多高》

**公钥的生成**：通过椭圆曲线的乘法可以私钥计算得到公钥，即：

K = k\*G

小k是私钥，G是被称为生成点的常数点，而大K是所得公钥。

这个已经被数学家证明不可逆或者说可逆的成本是无法忍受的。

比特币公钥的生成：

CPubKey pubkey = secret.GetPubKey();

CPubKey CKey::GetPubKey() const {

……

CPubKey result;

int clen = 65;

int ret = secp256k1\_ec\_pubkey\_create(secp256k1\_context, (unsigned char\*)result.begin(), &clen, begin(), fCompressed);

……..

}

在上面的创建函数中会调用下面这个函数，实现创建公钥：

static int secp256k1\_eckey\_pubkey\_serialize(secp256k1\_ge\_t \*elem, unsigned char \*pub, int \*size, int compressed) {

if (secp256k1\_ge\_is\_infinity(elem)) {

return 0;

}

secp256k1\_fe\_normalize\_var(&elem->x);

secp256k1\_fe\_normalize\_var(&elem->y);

secp256k1\_fe\_get\_b32(&pub[1], &elem->x);

if (compressed) {

\*size = 33;

pub[0] = 0x02 | (secp256k1\_fe\_is\_odd(&elem->y) ? 0x01 : 0x00);

} else {

\*size = 65;

pub[0] = 0x04;

secp256k1\_fe\_get\_b32(&pub[33], &elem->y);

}

return 1;

}

**4、比特币地址**

比特币地址是一个由数字和字母组成的字符串，由公钥经过单向的加密哈希算法得到。**由公钥**生成的比特币地址开头一定是1.

计算过程：以公钥 K 为输入，计算其SHA256哈希值，并以此结果计算RIPEMD160 哈希值，得到一个长度为160位（20字节）的数字：

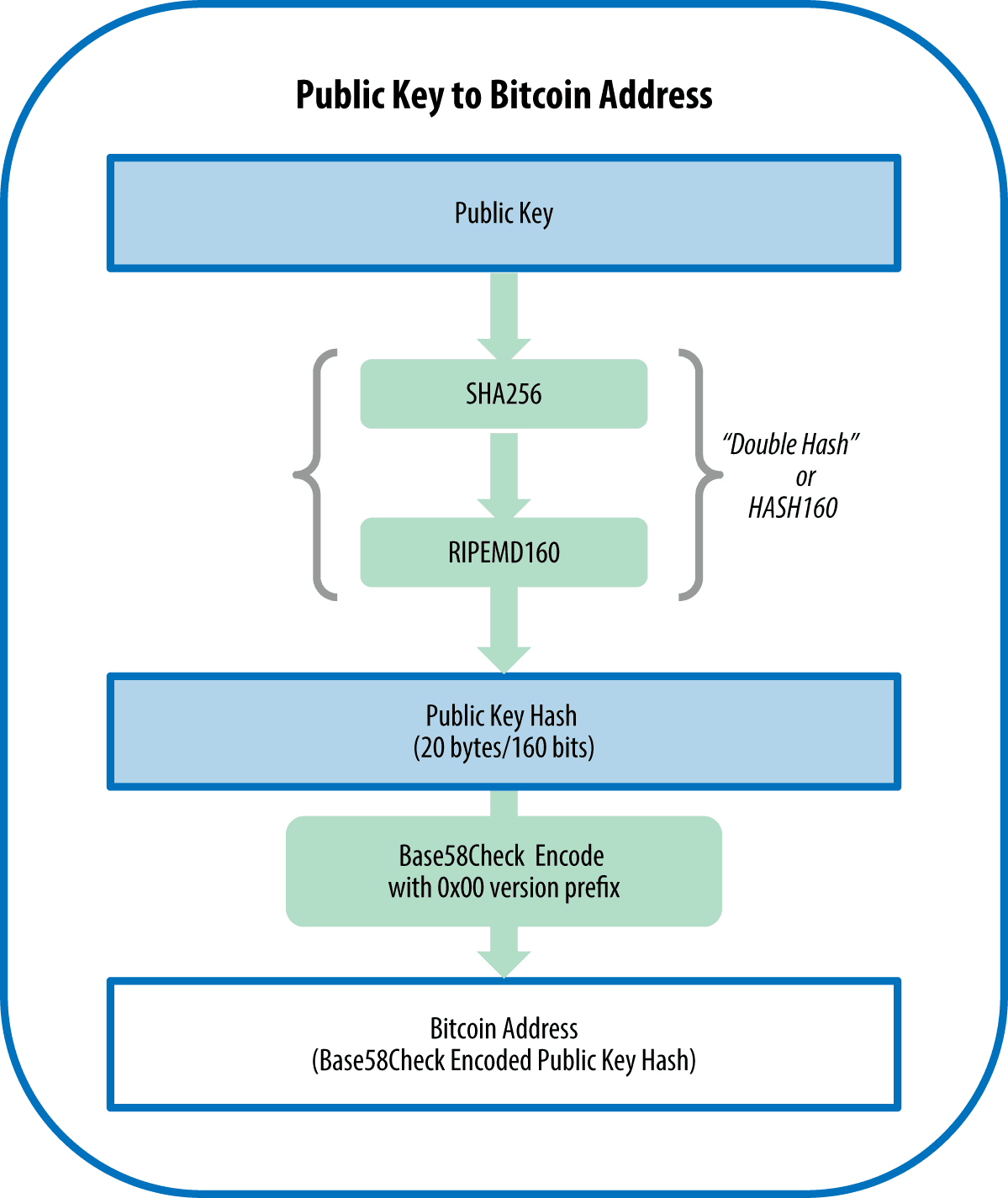
A = RIPEMD160(SHA256(K))

公式中，K是公钥，A是生成的比特币地址。

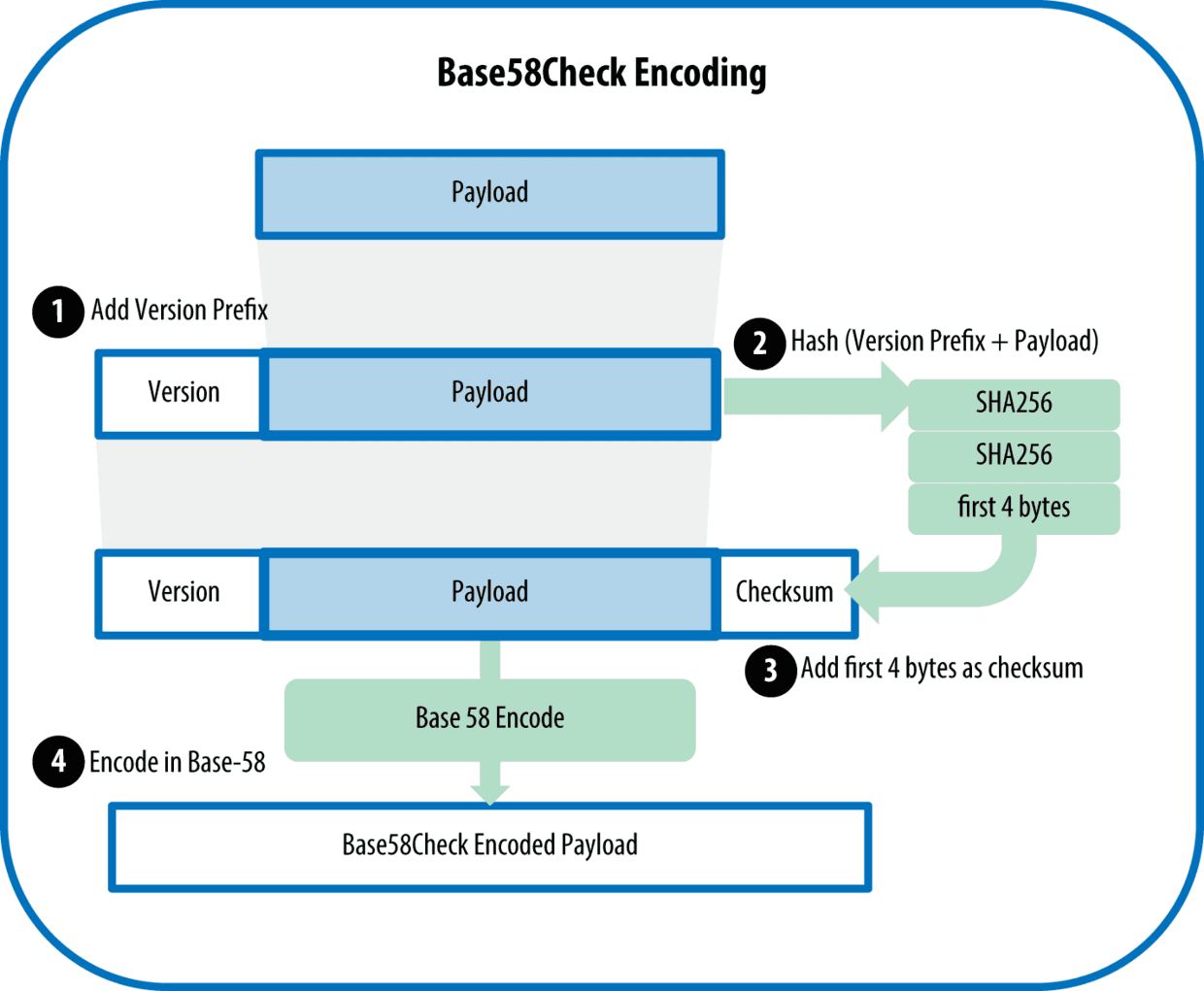
**提示比特币地址与公钥不同。比特币地址是由公钥经过单向的哈希函数生成的。**

（具体的Base58，16，64不赘述）

**椭圆曲线这里有一个动画。**



转换编码Base58Check：



例程中生成公钥过程：

bc::wallet::ec\_private secret(

decoded, bc::wallet::ec\_private::mainnet\_p2kh);

// Get public key.

bc::wallet::ec\_public public\_key(secret);

例程里其实是已经生成了一个私钥，再用其来生成公钥，在构造

数中最后一个参数默认为True。

ec\_private::ec\_private(const ec\_secret& secret, uint16\_t version, bool compress)

: valid\_(true), compress\_(compress), version\_(version), secret\_(secret){}

然后它用比特币的要求结果由三部分组成：前缀（0x00）、数据和校验和(4 bytes)：

unencoded\_address.reserve(25);

// Version byte, 0 is normal BTC address (P2PKH).

unencoded\_address.push\_back(0);

// Hash data

bc::extend\_data(unencoded\_address, hash);

// Checksum is computed by hashing data, and adding 4 bytes from hash.

bc::append\_checksum(unencoded\_address);

再用下面的方法生成标准的BASE58Check的地址：

const std::string address = bc::encode\_base58(unencoded\_address);

书中调用的是libbitcoin的函数（base\_58.cpp），如下：

std::string EncodeBase58(const unsigned char\* pbegin, const unsigned char\* pend)

{

// Skip & count leading zeroes.

int zeroes = 0;

while (pbegin != pend && \*pbegin == 0) {

pbegin++;

zeroes++;

}

// Allocate enough space in big-endian base58 representation.

std::vector<unsigned char> b58((pend - pbegin) \* 138 / 100 + 1); // log(256) / log(58), rounded up.

// Process the bytes.

while (pbegin != pend) {

int carry = \*pbegin;

// Apply "b58 = b58 \* 256 + ch".

for (std::vector<unsigned char>::reverse\_iterator it = b58.rbegin(); it != b58.rend(); it++) {

carry += 256 \* (\*it);

\*it = carry % 58;

carry /= 58;

}

assert(carry == 0);

pbegin++;

}

// Skip leading zeroes in base58 result.

std::vector<unsigned char>::iterator it = b58.begin();

while (it != b58.end() && \*it == 0)

it++;

// Translate the result into a string.

std::string str;

str.reserve(zeroes + (b58.end() - it));

str.assign(zeroes, '1');

while (it != b58.end())

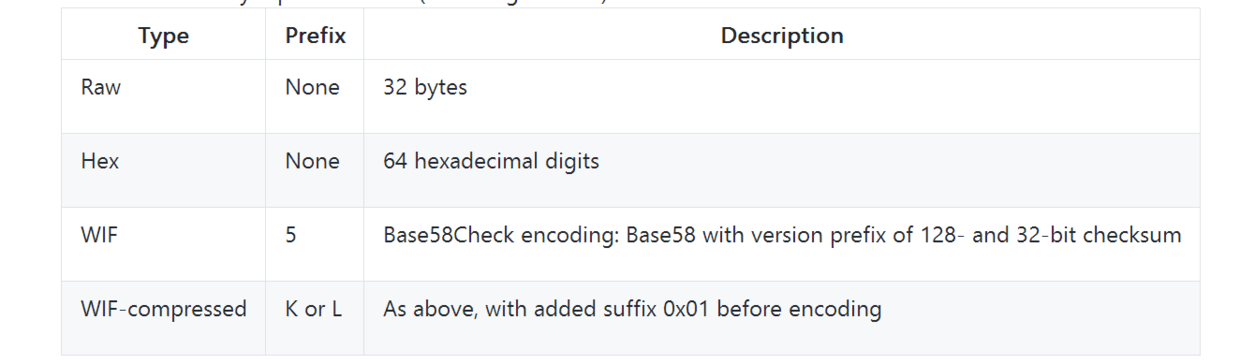
str += pszBase58[\*(it++)];

return str;

}

**5、密钥的格式**

**私钥有三种表现形式**：



看一个二维码【WIF格式】：



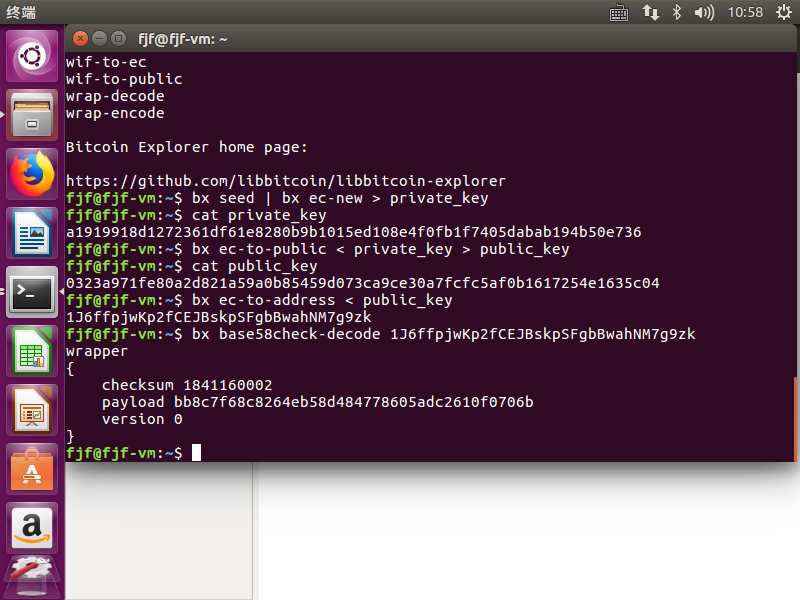
压缩私钥其实并没有减少数据占用量，反而多了一个字节01.新钱包目前都是这种压缩格式。它的目的主要是对应压缩的公钥。

**公钥的格式**：

公钥主要有压缩和非压缩两种格式，目的主要是解决公钥在交易过程中太长占用空间的问题。未压缩的公钥以04开头，压缩的以02或者03开头。表示正负两种可能，在图上表示横坐标的上方和下方。

注意：pybitcointools这个加已经不再更新，维护地址上说明如果你不怕担心什么后果，可以从历史分支上下载。

故基于其的使用这里略过。



# 高级密钥和地址

**加密私钥**：

BIP0038提出了一个通用标准，使用一个口令加密私钥并使用Base58Check对加密的私钥进行编码，这样加密的私钥就可以安全地保存在备份介质里，安全地在钱包间传输，保持密钥在任何可能被暴露情况下的安全性。

WIF(Wallet import Format)格式。

BIP0038加密方案是：输入一个比特币私钥，通常使用WIF编码过，base58chek字符串的前缀“5”。此外BIP0038加密方案需要一个长密码作为口令，通常由多个单词或一段复杂的数字字母字符串组成。BIP0038加密方案的结果是一个由 base58check编码过的加密私钥，前缀为6P。如果你看到一个6P开头的的密钥，这就意味着该密钥是被加密过，并需要一个口令来转换（解码）该密钥回到可被用在任何钱包WIF格式的私钥（前缀为5）。

**P2SH和多重签名地址**：

好多人会误认为前者是后者，其实不是。只不过前者多用于后者的情况。它一般以数字3开头。

顾名思义，这其实就是一个脚本生成的地址，所以它在新的标准里是不生成UTXO的。它的生成和比特币生成的方法是保持一致的。也是采用双哈希函数。

~$ echo dup hash160

[89abcdefcdbaabbaabbaabbaabbaabbaabbaabba]

equalverify checksig > script

fjf@fjf-vm:~$ bx script-encode < script | bx sha256 | bx ripemd160 | bx base58check-encode --version 5

35RhJmqCtpXLCcrndx837rqQsjx8c3fje1

多重签名的优势就是可以在获得一定数量签名后才可执行交易，最典型的应用就是众筹。

**靓号程序源码**：

std::string bitcoin\_address(const bc::ec\_secret& secret)

{

// Convert secret to payment address

bc::wallet::ec\_private private\_key(secret);

bc::wallet::payment\_address payaddr(private\_key);

// Return encoded form.

return payaddr.encoded();

}

bool match\_found(const std::string& address)

{

auto addr\_it = address.begin();

// Loop through the search string comparing it to the lower case

// character of the supplied address.

for (auto it = search.begin(); it != search.end(); ++it, ++addr\_it)

if (\*it != std::tolower(\*addr\_it))

return false;

// Reached end of search string, so address matches.

return true;

}

其实很简单，就是由随机数生成一个私钥，私钥转成地址，然后再匹配查找一下，如果不是就循环继续查找。

# 与第一版的比较

1. 将钱包单独独立成了一篇章节。
2. 移除了一些过时的工具比如SX，引入新的BX。
3. 新版本忽略了一些细节，更注重从整体上把握整个比特币。比如对一些代码和交易的文本都进行了省略。
4. 将一些具体的内容迁移到了相关章节，比如上面省略的部分，使得章节划分更加合理。