**比特币实验报告**

**一、任务**

**1.在比特币测试网上发送一个tx，并将tx数据解析到每一bit，最好自己编写脚本**

**2.报告此推断技术在以太坊ECDSA中的应用**

**二、实验步骤**

**（一）任务一**

**1、下载Centos7**

（1）下载阿里的镜像：https://mirrors.aliyun.com/centos/7/isos/x86\_64/

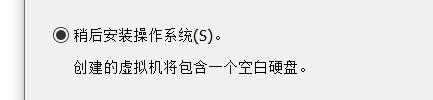
使用的是Minimal ISO 精简版：CentOS-7-x86\_64-Minimal-2009.iso

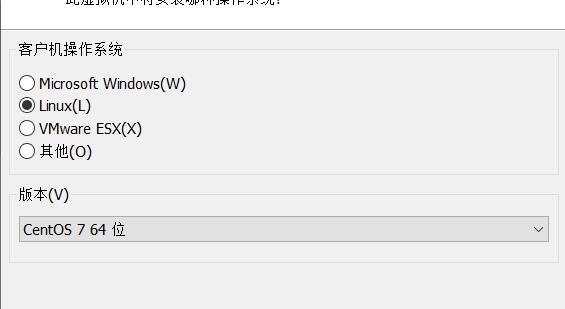
（2）点击虚拟机进行安装，点击创建虚拟机。



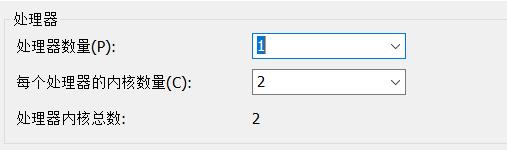
（3）选择典型安装。



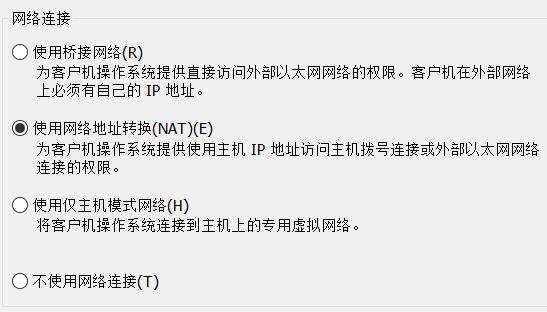




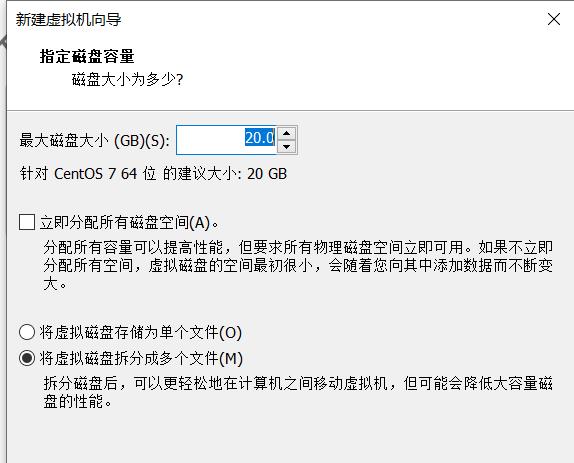
逻辑处理器数量:指线程数，即同时并行处理线程数量的能力。线程数=CPU颗数 \* 每个cpu内核数。



VMnet8网口对应的是NAT模式。



创建20G磁盘





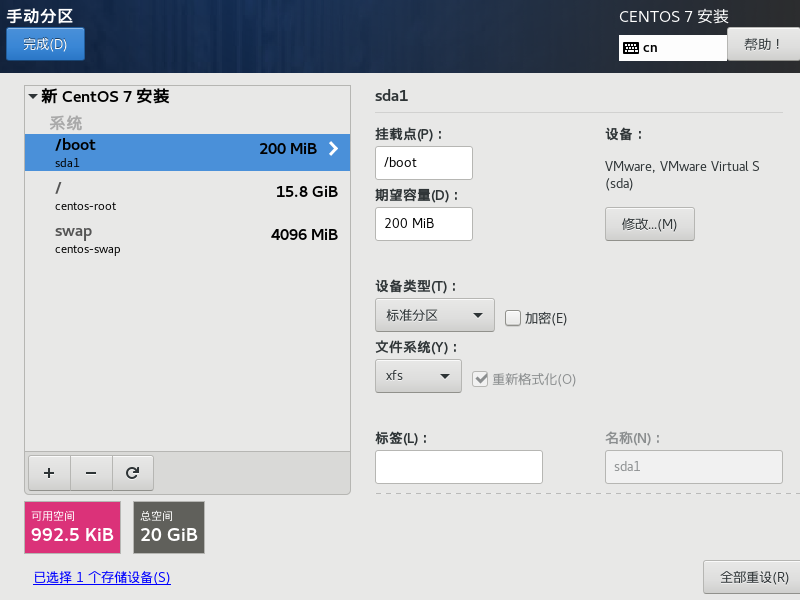
（4）开始安装选择第一个选项即可。

（5）选择安装过程中的语言



（6）自定义分区

分别创建 /boot区、swap交换分区、根分区/

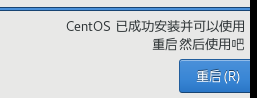


（7）网络和主机名设置



（8）开始安装

（9）设置root密码为root



**2.下载Docker**

（1）安装需要的安装包

sudo yum install -y yum-utils

（2）设置镜像的仓库

sudo yum-config-manager \

--add-repo \

<http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

（3）更新yum软件包索引

yum makecache fast

（4）安装docker相关 docker-ce社区版 ee企业版

sudo yum install docker-ce docker-ce-cli containerd.io

（5）启动docker

sudo systemctl start docker

**3.搭建比特币测试网**

### （1）下载比特币测试网络(bitcoin-testnet)的Docker镜像

docker pull freewil/bitcoin-testnet-box

### （2）运行Docker镜像

docker run -t -i -p 19001:19001 -p 19011:19011 freewil/bitcoin-testnet-box

输出如下类似：

@ubuntu:~$ sudo docker run -t -i -p 19001:19001 -p 19011:19011 freewil/bitcoin-testnet-box  
tester@647e5bee8412:~/bitcoin-testnet-box$

### （3）启动测试网络

①进入Docker运行环境后，输入下面的命令来启动两个比特币节点，从而组成比特币测试网络：

make start

输出如下类似:

tester@647e5bee8412:~/bitcoin-testnet-box$ make start  
bitcoind -datadir=1 -daemon  
Bitcoin server starting  
bitcoind -datadir=2 -daemon  
Bitcoin server starting

②启动成功后，将在本机模拟运行两个比特币测试钱包节点，组成一个私有范围的比特币测试网络。

输入下面的命令可以查看测试网络节点状态信息:

make getinfo

输出如下类似：

bitcoin-cli -datadir=1 getinfo //第一个钱包节点的信息  
{  
​ “version”: 120100,  
​ “protocolversion”: 70012,  
​ “walletversion”: 60000,  
​ “balance”: 0.00000000,  
​ “blocks”: 0,  
​ “timeoffset”: 0,  
​ “connections”: 1,  
​ “proxy”: “”,  
​ “difficulty”: 4.656542373906925e-10,  
​ “testnet”: false,  
​ “keypoololdest”: 1467253951,  
​ “keypoolsize”: 101,  
​ “paytxfee”: 0.00000000,  
​ “relayfee”: 0.00001000,  
​ “errors”: “”  
}  
bitcoin-cli -datadir=2 getinfo //第二个钱包节点的信息  
{  
​ “version”: 120100,  
​ “protocolversion”: 70012,  
​ “walletversion”: 60000,  
​ “balance”: 0.00000000,  
​ “blocks”: 0,  
​ “timeoffset”: 0,  
​ “connections”: 1,  
​ “proxy”: “”,  
​ “difficulty”: 4.656542373906925e-10,  
​ “testnet”: false,  
​ “keypoololdest”: 1467253951,  
​ “keypoolsize”: 101,  
​ “paytxfee”: 0.00000000,  
​ “relayfee”: 0.00001000,  
​ “errors”: “”  
}

③查看下两个节点的bitcoin.conf文件是如何配置的  
节点1的配置文件，路径：/home/tester/bitcoin-testnet-box/1/bitcoin.conf

# testnet-box functionality  
regtest=1  
dnsseed=0  
upnp=0  
# listen on different ports than default testnet  
port=19000  
rpcport=19001  
# always run a server, even with bitcoin-qt  
server=1  
# enable SSL for RPC server  
#rpcssl=1  
rpcallowip=0.0.0.0/0  
rpcuser=admin1  
rpcpassword=123

节点2的配置文件，路径：/home/tester/bitcoin-testnet-box/2/bitcoin.conf

内容类似节点1。

再顺便查看下节点1的对等节点的信息：

tester@0cc1d63a966c:~/bitcoin-testnet-box$ bitcoin-cli -datadir=1 getpeerinfo

[  
{  
​ "id": 0,  
​ "addr": "127.0.0.1:52766",  
​ "addrbind": "127.0.0.1:19000",  
​ "services": "000000000000040d",  
​ "relaytxes": true,  
​ "lastsend": 1531383259,  
​ "lastrecv": 1531383259,  
​ "bytessent": 429,  
​ "bytesrecv": 453,  
​ "conntime": 1531383259,  
​ "timeoffset": 0,  
​ "pingtime": 0.002959,  
​ "minping": 0.002959,  
​ "version": 70015,  
​ "subver": "/Satoshi:0.16.0/",  
​ "inbound": true,  
​ "addnode": false,  
​ "startingheight": 0,  
​ "banscore": 0,  
​ "synced\_headers": -1,  
​ "synced\_blocks": -1,  
​ "inflight": [  
​ ],  
​ "whitelisted": false,  
​ "bytessent\_per\_msg": {  
​ "feefilter": 32,  
​ "getheaders": 93,  
​ "ping": 32,  
​ "pong": 32,  
​ "sendcmpct": 66,  
​ "sendheaders": 24,  
​ "verack": 24,  
​ "version": 126  
​ },  
​ "bytesrecv\_per\_msg": {  
​ "feefilter": 32,  
​ "getaddr": 24,  
​ "getheaders": 93,  
​ "ping": 32,  
​ "pong": 32,  
​ "sendcmpct": 66,  
​ "sendheaders": 24,  
​ "verack": 24,  
​ "version": 126  
​ }  
}]

④查看下节点2的对等节点的信息：

tester@0cc1d63a966c:~/bitcoin-testnet-box$ bitcoin-cli -datadir=2 getpeerinfo

输出与节点1类型

### （4） 初始化和区块链数据

①使用getnewaddress命令分别为两个钱包生成一个地址(或者用命令 make address1 也可以)：

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ bitcoin-cli -datadir=1 getnewaddress

2NAUVNvRVKn2QT2yoKYo1LXBBHES1DiiAWp

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ bitcoin-cli -datadir=2 getnewaddress

2N5czXHSEFronnYvMHUhSinQW8jjp7UjRtu

这里生成的KEY,根据具体情况会发生变化

②查看地址对应的私钥：

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ bitcoin-cli -datadir=1 dumpprivkey 2NAUVNvRVKn2QT2yoKYo1LXBBHES1DiiAWp

cP5cEwanYgwJTUA6RG43DQ9p3ErpmqdPrJK7kX48zaMKkEfoL8s3

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ bitcoin-cli -datadir=2 dumpprivkey 2N5czXHSEFronnYvMHUhSinQW8jjp7UjRtu

cQ8UyP5CoJ2SbHXdJagxxHSjYnxd69kaCaSU7uMPq6iRxpEhpuYP

③生成一个区块：

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ make generate

输出类似如下：

bitcoin-cli -datadir=1 generate 1  
[  
"5db89a9818792f8665b6540d32c3834856cb6a07ed34aecb03128e96cbe3658a"]

继续生成9个区块：

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ make generate BLOCKS=9

bitcoin-cli -datadir=1 generate 9  
[  
"61bc358db1ddb82972b830a00cb1dd27a7e6f3f5b409d667ff90af7a72caeb75",  
"0eedb9ecb498df0e638013343ca97189ca768c54592ed5155a2ccbedf13f5309",  
"7f74386d70b295102c5e44e373232648f7c07e00918c0b6de1ad033c3c121b38",  
"7ca80b7bf1209d18e56b5c399992171c074c566e3109376b09877401b86a0b1d",  
"61f0b2f3f4d2d4d199d7cbbed9bba721e65de3e30f474042c0eb4ce4cfc1ad54",  
"2f97d16c46e5ccc46b795b9ee72f58720c494ae0f48d98d70a5b5a9e8b5be2ba",  
"26edd9fd85b3e75ba7e731596ac065afbc29f68d8a6dd16f928d1e898ae18fb6",  
"684fa2b10a662bffdd7e4e148e3a73c430cf7bbc14d0ed320200e008dd1ea625",  
"2b62353790c59d2e643c815d56952264052dd99876a61e0d46e81accd7661262"]

④查看钱包信息：

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ make getwalletinfo

输出类似如下：

bitcoin-cli -datadir=1 getwalletinfo  
{  
"walletname": "wallet.dat",  
"walletversion": 159900,  
"balance": 0.00000000,  
"unconfirmed\_balance": 0.00000000,  
"immature\_balance": 500.00000000,  
"txcount": 10,  
"keypoololdest": 1531366844,  
"keypoolsize": 999,  
"keypoolsize\_hd\_internal": 1000,  
"paytxfee": 0.00000000,  
"hdmasterkeyid": "0ac63ad922d529b4ff4be71a10f1751c3978b5f5"  
}  
bitcoin-cli -datadir=2 getwalletinfo  
{  
"walletname": "wallet.dat",  
"walletversion": 159900,  
"balance": 0.00000000,  
"unconfirmed\_balance": 0.00000000,  
"immature\_balance": 0.00000000,  
"txcount": 0,  
"keypoololdest": 1531366844,  
"keypoolsize": 999,  
"keypoolsize\_hd\_internal": 1000,  
"paytxfee": 0.00000000,  
"hdmasterkeyid": "78f6a0f992aae42ff3504477f150a65e86c1f287"  
}

结果显示钱包1的balance(余额)里根本没有任何比特币，但immature\_balance(未成熟的余额)里有相应的500个比特币，这是为什么呢？

结果显示，钱包1和钱包2都有20个区块了，没有什么问题，那么为什么钱包1的balance里没有比特币呢？

⑤继续创建200个区块：

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ make generate BLOCKS=200

再查询钱包余额：

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ make getwalletinfo

bitcoin-cli -datadir=1 getwalletinfo  
{  
"walletname": "wallet.dat",  
"walletversion": 159900,  
"balance": 6000.00000000,  
"unconfirmed\_balance": 0.00000000,  
"immature\_balance": 3225.00000000,  
"txcount": 220,  
"keypoololdest": 1531366844,  
"keypoolsize": 999,  
"keypoolsize\_hd\_internal": 1000,  
"paytxfee": 0.00000000,  
"hdmasterkeyid": "0ac63ad922d529b4ff4be71a10f1751c3978b5f5"  
}  
bitcoin-cli -datadir=2 getwalletinfo  
{  
"walletname": "wallet.dat",  
"walletversion": 159900,  
"balance": 0.00000000,  
"unconfirmed\_balance": 0.00000000,  
"immature\_balance": 0.00000000,  
"txcount": 0,  
"keypoololdest": 1531366844,  
"keypoolsize": 999,  
"keypoolsize\_hd\_internal": 1000,  
"paytxfee": 0.00000000,  
"hdmasterkeyid": "78f6a0f992aae42ff3504477f150a65e86c1f287"  
}

结果显示钱包1的balance(余额)里6000个比特币了。

⑥给钱包2转账，这里转10个比特币：

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ make sendfrom1 ADDRESS=2N5czXHSEFronnYvMHUhSinQW8jjp7UjRtu AMOUNT=10

bitcoin-cli -datadir=1 sendtoaddress 2N5czXHSEFronnYvMHUhSinQW8jjp7UjRtu 10

967d37c71c1f768ff75e8ee675cf0dc5899a953bb4954b3b0f8ad3f25fc2aa52

⑦查询钱包余额

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ make getwalletinfo

结果显示钱包2的10个比特币还未确认。

好办，继续生成10个区块，使得交易得到确认：

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ make generate BLOCKS=10

⑧再查看钱包余额：

tester@0f4ec9f7a91a:~/bitcoin-testnet-box$ make getwalletinfo

从结果可以看到钱包2的10个比特币到账了，说明交易得到了确认。

**（二）任务二**

密码学签名是[区块链](https://so.csdn.net/so/search?q=%E5%8C%BA%E5%9D%97%E9%93%BE&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_44282220/article/details/_blank)的关键技术之一，可以在不暴露私钥的前提下证明地址的所有权。该技术主要用来签署交易。

1.ECDSA 签名由两个数字（整数）组成：r 和 s。以太坊还引入了额外的变量 v（恢复标识符）。签名可以表示成 {r, s, v}。

在创建签名时，你要先准备好一条待签署的消息，和用来签署该消息的私钥（dₐ）。简化后的签名流程如下：

（1）对待签署消息进行哈希计算，得到哈希值（e）。

（2）生成一个安全的随机数k。

（3）将k乘以椭圆曲线的常量G，来计算椭圆曲线上的点（x₁, y₁）。

（4）计算r = x₁ mod n。如果r等于0，请返回步骤2 。

（5）计算s = k-1(e + rdₐ) mod n。如果s等于0，请返回步骤2。

（6）在以太坊上，通常使用 Keccak256("\x19Ethereum Signed Message:\n32" + Keccak256(message))来计算哈希值。这样可以确保该签名不能在以太坊之外使用。

由于k是随机值，我们每次得到的签名都不一样。如果 k 的随机程度不够高，或者随机值被泄漏，就有可能使用两个不同的签名计算出私钥【“fault attack”】。但是，如果你在 MyCrypto 内签署同一条消息，每次得到的输出值都相同，那么如何确保其安全性？这些确定性签名均采用 RFC 6979 标准。该标准描述了如何基于私钥和消息（或哈希值）来生成安全的k值。

3.为了验证消息，我们需要掌握原始消息、使用私钥签署消息的地址，以及 {r, s, v} 签名本身。版本号就是 MyCrypto 使用的某个版本号。旧版本的 MyCrypto 通常会加上消息的当前日期和时间，计算其哈希值，然后按照上述步骤签署该消息。后来又进行了更改，以符合 JSON-RPC 方法 personal\_sign 方法，因此需要指明版本号（“2”）。

简化后的公钥恢复流程如下：

（1）计算消息的哈希值（e）。

（2）计算椭圆曲线上的点 R = (x₁, y₁)，其中 x₁ 是 r（v = 27），或 r + n（v = 28）。

（3）计算u₁=-zr-1mod n和u₂=sr-1mod n。

（4）计算点Qₐ=(xₐ, yₐ)=u₁×G+u₂×R。

Qₐ是地址用来签名的私钥所对应的公钥。我们可以通过公钥计算出一个地址，并检查该地址是否与已提供地址相符。如果相符，则签名有效。

4.签署交易。就像消息一样，交易在发送前也需要签名。如果你使用 Ledger 和 Trezor 之类的硬件钱包，签名过程会在硬件内部发生。如果使用私钥（或 keysotre 文件、助记词），可以直接在 MyCrypto 上完成签名。签署交易所使用的方法与签署消息非常相似，只不过交易的编码方式略有不同。

要签署的交易先用 RLP 编码方式编码，包含了所有交易参数（nonce、gas price、gas limit、to、value、data）和签名（v, r, s）。

签过名的交易的第一组字节包含 RLP 编码后的交易参数，最后一组字节包含签名 {r, s, v}。我们可以通过以下方式对签名交易进行编码：

（1）交易参数：RLP(nonce, gasPrice, gasLimit, to, value, data, chainId, 0, 0)。

（2）使用 Keccak256 算法来计算经过RLP编码的未签署交易的哈希值。

（3）按照上文讲述的步骤，通过ECDSA算法，使用私钥签署哈希值。

（4）对已签名的交易进行编码：RLP(nonce, gasPrice, gasLimit, to, value, data, v, r, s)。

将经过 RLP 编码的交易数据解码后，我们又可以得到原始交易参数和签名。

请注意，链ID是被编码到签名的v参数中的，因此我们不会将链ID本身包含在最终的签名交易数据中。我们也不会提供任何发送方地址，因为地址可以通过签名恢复。这就是以太坊网络内部用来验证交易的方式。

### 5.对于区块链和去中心化来说，签名非常重要。签名不仅可以用来发送交易，还可以用来与去中心化交易所、多签合约和其它智能合约进行交互。目前还没有明确的消息签名标准，进一步采用 EIP 712 规范有助于生态系统改善用户体验，并为消息签名制定标准。