Lab 8: Bitcoin and Blockchain

1 Details

Aim: To provide a foundation in understanding in Bitcoin and Blockchain.

Demo of setting up Geth: https://youtu.be/Gl3Suylr-7E

2 Activities

L1.1 Using blockchain.info, find the details of the genesis block:

Date created:

Reward:

Number of transactions:

Size of block:

Which account received the mining reward for the genesis block (last four digits):

How many USD does the original miner have in the account they used for the first genesis record:

When did the genesis block creator stop trading?

L1.2 Using blockchain.info, determine the following

Total bitcoins in circulation:

Most recent hash block (last four hex digits):

Block reward per block:

Difficulty:

Average time between blocks:

Market capitalisation (USD):

24 hr price (USD):

24hr transactions (USD):

Hash rate:

Last successful miner:

Maximum block size:

Balance for 1GbVUSW5WJmRCpaCJ4hanUny77oDaWW4to:

L1.3 Download and create the Python file defined on this page:

https://asecuritysite.com/encryption/bit

Now run the Python file, and compare the results in L.1.2.

Total bitcoins in circulation:

Most recent hash block (last four hex digits):

Block reward per block:

Difficulty:

Average time between blocks:

Market capitalisation (USD):

24 hr price (USD):

24hr transactions (USD):

Hash rate:

Balance for 1GbVUSW5WJmRCpaCJ4hanUny77oDaWW4to:

C Ethereum

In this tutorial, we will run an Ethereum blockchain on a Windows host.

Demo: https://www.youtube.com/watch?v=Gl3Suylr-7E
Outline: https://asecuritysite.com/subjects/chapter91

On your desktop computer, download Geth, and install it.

https://geth.ethereum.org/downloads/

Open a terminal on your Windows desktop and run as an Administrator. Next go to "c:\program files\geth" folder.

We are going to create the blockchain in the **c:\eth6** folder. First create **three new accounts**:

```
C:\program files\geth> geth --datadir=c:\eth6 account new
```

Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase: **Qwerty1**
Repeat passphrase: **Qwerty1**
Address: {5cba4752a6fe25ffbd7710a67900d3517d7be4db}

Open custom.json, and copy and paste the following details for your genesis block, but **replace the hex IDs with the three accounts that you have created**:

```
{
    "config": {
        "chainId": 15,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
    "difficulty": "20000000",
    "gasLimit": "0x3d0900",
    "alloc": {
        "228041751ddb7365cc4bc75c4985d14d5db2432f": { "balance": "30000000" },
        "cdfc92d1b5dd1c9ee1c9e2368abc86a193ae35a5": { "balance": "40000000" },
        "c9c425ae15a0e66500ecf5b7a1c10c6ed35600b9": { "balance":
"0x400000000000000" }

    }
}
```

Next run **geth** and create the genesis block details:

```
c:\Program Files\Geth> geth --datadir=c:\eth6 init customg.json
INFO [06-26|21:42:43] Allocated cache and file handles         database=d:\\eth6
\\geth\\chaindata cache=16 handles=16
INFO [06-26|21:42:43] Writing custom genesis block
INFO [06-26|21:42:43] Successfully wrote genesis state         database=chaindat
a                 hash=10367b.67437b
INFO [06-26|21:42:43] Allocated cache and file handles         database=d:\\eth6
\\geth\\lightchaindata cache=16 handles=16
Fatal: Failed to write genesis block: database already contains an incompatible
genesis block (have 0c5f429f24f7078a, new 10367b56f68be716)
```

Examine the c:\eth6 folder.

What are the contents of this folder:

Next we will start our blockchain:

```
C:\Program Files\Geth> geth --datadir=c:\eth6
```

Next we will connect to the geth and create a new account:

```
C:\Program Files\Geth> geth attach
Welcome to the Geth JavaScript console!

instance: Geth/v1.6.6-stable-10a45cb5/windows-amd64/go1.8.3
coinbase: 0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9
at block: 0 (Thu, 01 Jan 1970 00:00:00 GMT)
 datadir: d:\eth6
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txp
ool:1.0 web3:1.0

> personal.newAccount("Qwerty1")
"0xce1373ddfa2232dc9ca82d98420be7a2e11962b5"

> web3.eth.accounts
["0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9", "0xbb4fcfac2efd3dbc35117dc979ce5c
43ca5c615b", "0xce1373ddfa2232dc9ca82d98420be7a2e11962b5"]
```

Take a note of your new account ID:

We can look at the initial balances in the accounts (use the hex values contained in the accounts). For the following view of all the account balances (replace the hex IDs with the ones on your system):

```
> eth.getBalance("0xce1373ddfa2232dc9ca82d98420be7a2e11962b5")
0
> eth.getBalance("0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9")
288230376151711744
```

What are the balances:

Next unlock the account with the most Ether:

```
>
personal.unlockAccount('0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9','Qwert
y')
true
```

Next we can transfer some currency from one account to another (transfer from the account with the most Ether into your account). For this, transfer Ether from the account with most funds to your newly created account, and then view the transaction:

```
> eth.sendTransaction({from:
'0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9', to:
'0xce1373ddfa2232dc9ca82d98420be7a2e11962b5',value:1000})
"0x4029e82ac13fd2a56078c2747f2ff55b42db12c8fa40dbde8c6350b128476243"
>
>
eth.getTransaction('0x4029e82ac13fd2a56078c2747f2ff55b42db12c8fa40db
de8c6350b128476243')
{
  blockHash: "0x0000000000000000000000000000000000000000000000000000000000000000
",
  blockNumber: null,
  from: "0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9",
  gas: 90000,
  gasPrice: 18000000000,
  hash: "0x4029e82ac13fd2a56078c2747f2ff55b42db12c8fa40dbde8c6350b128476243",
  input: "0x",
  nonce: 0,
  r: "0xedbbbe21778eab7a3b3f82198854e6354abff4348dc9668ec337a786749a4d3a",
  s: "0x27228d637ac06acf1ffdcd93ff5a2dbd59f23353d196b97ff2ee7e2a14527595",
  to: "0xce1373ddfa2232dc9ca82d98420be7a2e11962b5",
  transactionIndex: 0,
  v: "0x41",
  value: 1000
}
```

If we look at the balances there has not been any transfers:

```
> eth.getBalance("0xce1373ddfa2232dc9ca82d98420be7a2e11962b5")
0
> eth.getBalance("0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9")
288230376151711744
```

We can now start the miner and view the balances:

```
> miner.start()
null
> eth.getBalance("0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9")
288230376151711744
> eth.getBalance("0xce1373ddfa2232dc9ca82d98420be7a2e11962b5")
0
```

We can transfer again:

```
> eth.sendTransaction({from:
'0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9', to:
'0xce1373ddfa2232dc9ca82d98420be7a2e11962b5',value:100000})
"0x2e25093e25cbf511c2892cb38b45a5c9f6f9b2785774cd5830cf5bd978839165"
> eth.getBalance("0xce1373ddfa2232dc9ca82d98420be7a2e11962b5")
0
> eth.getBalance("0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9")
288230376151711744
```

The mining process adds some credits to the initial account:

```
> eth.getBalance("0xc9c425ae15a0e66500ecf5b7a1c10c6ed35600b9")
5288230376151711744

> eth.getBalance("0xce1373ddfa2232dc9ca82d98420be7a2e11962b5")
0
```

the mining process we see:

```
> eth.getBalance("0xce1373ddfa2232dc9ca82d98420be7a2e11962b5")
200000
```

If we look at the blockchain we see there are two records:

```
> eth.blockNumber
2
```

What are the balances in the accounts:

```

```

D Creating a contract

Now let's create a contract. First open up:

http://ethereum.github.io/browser-solidity

and paste the following code:

```
pragma solidity ^0.4.0;
contract test2{
   uint a ;
   function test2() {
       a = 1;
   }
   function val() returns(uint){
       return a;
   }
}
contract test3 is test2{
    uint b = a++;
    function show() returns(uint){
        return b;
    }
}
```

We can compile the code (use the 0.5.0 compiler) and get the Web3Deploy version (by clicking on Bytecode):


```
var test2Contract =
web3.eth.contract([{"constant":false,"inputs":[],"name":"val","outputs":[{
"name":"","type":"uint256"}],"payable":false,"stateMutability":"nonpayable
","type":"function"},{"inputs":[],"payable":false,"stateMutability":"nonpa
yable","type":"constructor"}]);
var test2 = test2Contract.new(
   {
     from: web3.eth.accounts[0],
     data:
'0x60606040523415600e57600080fd5b6001600081905550609f806100246000396000f30
0606060405260043610603f576000357c01000000000000000000000000000000000000000
0000000000000000000900463ffffffff1680633c6bb436146044575b600080fd5b3415604e5
7600080fd5b6054606a565b60405180828152602001915050604051809103 90f35b6000805
49050905600a165627a7a7230582062175dc4e9fcfe956cc06a92ee74103af0feae371ddeb
5bb46c65f3f74140a0f0029',
     gas: '4700000'
   }, function (e, contract){
```

```
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
         console.log('Contract mined! address: ' + contract.address + '
transactionHash: ' + contract.transactionHash);
    }
 })
```

Now we copy from **Web Deploy** and place in a JavaScript file (sayhello.js), and then load it:

```
>loadScript('sayhello2.js')
```

and next define the account to run the script (replace with one of your IDs):

```
> web3.eth.defaultAccount =
'0x821eacc2a570c1aeb9b5aa64b5b915d4c1e1f3ee'
```

We can now start our miners:

```
> miner.start()
null
> null [object Object]
Contract mined! address: 0x8d487f4a719b5a1cf47c61cc83e757b8d269f877 transactionH
ash: 0xf4bb0fa6ddc1d9e1921a55d576d68acf5b715d00cd89cc7268ece3653c50de50
null [object Object]
Contract mined! address: 0xf3872dc9ced78283ad3a511e970891807dd38590 transactionH
ash: 0xab90aa5169f4ebfcbc139874208cabb29416feb3f12c296c93466d7d8090f805
null [object Object]
Contract mined! address: 0x7a74b5da4168f0a06a752301a3711c8991acaf88 transactionH
ash: 0x6ce2a63c59d124d5ecd4681a368243ba7de8aeacc735d41583f834789cba0b16
```

Finally we can view:

```
> test_sol_test2
{
  abi: [{
      constant: false,
      inputs: [],
      name: "val",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      inputs: [],
      payable: false,
      type: "constructor"
  }],
  address: "0x7a74b5da4168f0a06a752301a3711c8991acaf88",
  transactionHash: "0x6ce2a63c59d124d5ecd4681a368243ba7de8aeacc735d41583f834789c
ba0b16",
```

```
  allEvents: function(),
  val: function()
}
> test_sol_test3
{
  abi: [{
      constant: false,
      inputs: [],
      name: "val",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      constant: false,
      inputs: [],
      name: "show",
      outputs: [{...}],
      payable: false,
      type: "function"
  }],
  address: "0xbd570c2f87b8af945146177377276901fd82b12d",
  transactionHash: "0xc028384b4d8ea0e283c9cd3a6a747ab3efff859bb591d55f710ca20b09
665808",
  allEvents: function(),
  show: function(),
  val: function()
}
```

And then test:

```
> test_sol_test2.val()
"0xd69b536cd4055a45e209f3274d9b9370f33c88b474c0dca294b665efa2ac5d2d"
> test_sol_test3.val()
"0x4a5fa248e8f6c2223082518106c3e784d54e4ff70793c9d4f65c9ef931cd667c"
```

E A bit of maths

Now we will create a contract to do a bit of maths. Let's say we want to calculate the square root of a value:

```
pragma solidity ^0.4.0;

contract mymath {
   function sqrt(uint x) constant returns (uint y) {
    uint z = (x + 1) / 2;
    y = x;
    while (z < y) {
        y = z;
```

```
        z = (x / z + z) / 2;
    }
}
}
```

When we create the JavaScript for the compiled version, and we load and run we get:

```
> personal.unlockAccount('0xc7552f45deb093cafb47286a0bc9415845ca3735','Qwerty')
true
> loadScript('mycontract.js')
null [object Object]
true
Contract mined! address: 0xc706a04b759a32dbec85702dd3864584e737aa77 transactionH
ash: 0xece670dcb578a78dec4d2338755ecade084a517310daacf37fd46fe336341563
null [object Object]
Contract mined! address: 0xfafb5f4d0db2c545592ac9134292162b03088295 transactionH
ash: 0x46204af57db69df078e1ae637b50fa76d8415ee1c1e3bd7e1c2990f328dc85ce
null [object Object]
Contract mined! address: 0x83e0bbb8abe2f0976fde9cf5db05333de067b0df transactionH
ash: 0xabea9606989bcc1bf93513213d298c84d47c7e8e1b397eaf536ebffb793d9304

> test_sol_mymath.sqrt(9)
3
> test_sol_mymath.sqrt(12)
3
> test_sol_mymath.sqrt(81)
9
```