Lab 8: Bitcoin and Blockchain

1 Details

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

At the end of this lab, remember to stop your Blockchain (Control-C from the console that is running Geth), and shut down your VM. You may also want to use "rm -r mynapier" in order to delete your 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 your Ubuntu host:

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

Outline: https://asecuritysite.com/subjects/chapter91

On your Ubuntu computer, install Geth:

```
sudo apt-get install software-properties-common
sudo add-apt-repository -y ppa:ethereum/Ethereum
sudu apt-get update
sudo apt-get install ethereum
```

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

```
napier@napier-virtual-machine:~$ sudo geth --datadir=mynapier account new
WARN [03-20|22:24:56.282] Sanitizing cache to Go's GC limits       provided=1024 updated=666
INFO [03-20|22:24:56.284] Maximum peer count                       ETH=25 LES=0 total=25
Passphrase:  Qwerty1
Address: {39a18a459b2475925e3014679707e4970a6a836d}
napier@napier-virtual-machine:~$ sudo geth --datadir=mynapier account new
WARN [03-20|22:25:12.291] Sanitizing cache to Go's GC limits       provided=1024 updated=666
INFO [03-20|22:25:12.293] Maximum peer count                       ETH=25 LES=0 total=25
Passphrase:  Qwerty1
Address: {3a5b61aeb069dc7df1b8f1b58e883118ea8bef3f}
napier@napier-virtual-machine:~$ sudo geth --datadir=mynapier account new
WARN [03-20|22:25:46.518] Sanitizing cache to Go's GC limits       provided=1024 updated=666
INFO [03-20|22:25:46.521] Maximum peer count                       ETH=25 LES=0 total=25
Passphrase: Qwerty1
Address: {a2cda8c68259cc314bd12b67873ce7380b3dc496}
```

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,
        "eip150Block": 0,
        "eip158Block": 0
    },
    "difficulty": "1",
    "gasLimit": "0x3d0900",
    "alloc": {
        "39a18a459b2475925e3014679707e4970a6a836d": { "balance": "30000000" },
        "3a5b61aeb069dc7df1b8f1b58e883118ea8bef3f": { "balance": "40000000" },
        "a2cda8c68259cc314bd12b67873ce7380b3dc496": { "balance": "0x400000000000000" }
    }
}
```

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

```
napier@napier-virtual-machine:~$ geth --nousb --datadir=mynapier init custom.json
WARN [03-20|20:53:53.824] Sanitizing cache to Go's GC limits       provided=1024 updated=666
INFO [03-20|20:53:53.832] Maximum peer count                       ETH=25 LES=0 total=25
INFO [03-20|20:53:53.834] Allocated cache and file handles
database=/home/napier/mynapier/geth/chaindata cache=16 handles=16
INFO [03-20|20:53:53.868] Writing custom genesis block
INFO [03-20|20:53:53.869] Persisted trie from memory database      nodes=4 size=580.00B
time=289.527µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [03-20|20:53:53.870] Successfully wrote genesis state         database=chaindata
hash=6061cd…6972ac
INFO [03-20|20:53:53.870] Allocated cache and file handles
database=/home/napier/mynapier/geth/lightchaindata cache=16 handles=16
INFO [03-20|20:53:53.915] Writing custom genesis block
INFO [03-20|20:53:53.916] Persisted trie from memory database      nodes=4 size=580.00B
time=161.337µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [03-20|20:53:53.916] Successfully wrote genesis state         database=lightchaindata
hash=6061cd…6972ac
```

We have now created wallets and the genesis block in the mynapier folder. Examine the **mynapier** folder.

What are the contents of this folder:

How are the keys stored:

Note: You may have to use "chmod 777 mynapier" to allow access to the folder.

Next we will start our blockchain:

```
napier@napier-virtual-machine:~$ geth --datadir=mynapier --networkid=15
WARN [03-20|20:54:31.161] Sanitizing cache to Go's GC limits       provided=1024 updated=666
INFO [03-20|20:54:31.168] Maximum peer count                       ETH=25 LES=0 total=25
INFO [03-20|20:54:31.195] Starting peer-to-peer node               instance=Geth/v1.8.23-
stable-c9427004/linux-amd64/go1.10.4
INFO [03-20|20:54:31.195] Allocated cache and file handles
database=/home/napier/mynapier/geth/chaindata cache=333 handles=524288
INFO [03-20|20:54:31.241] Initialised chain configuration          config="{ChainID: 15
Homestead: 0 DAO: <nil> DAOSupport: false EIP150: <nil> EIP155: 0 EIP158: 0 Byzantium: <nil>
Constantinople: <nil>  ConstantinopleFix: <nil> Engine: unknown}"
INFO [03-20|20:54:31.241] Disk storage enabled for ethash caches
dir=/home/napier/mynapier/geth/ethash count=3
INFO [03-20|20:54:31.241] Disk storage enabled for ethash DAGs     dir=/home/napier/.ethash
count=2
INFO [03-20|20:54:31.242] Initialising Ethereum protocol           versions="[63 62]"
network=1
INFO [03-20|20:54:31.497] Loaded most recent local header
```

Next open up another console, and we will connect to geth and create a new account:

```
napier@napier-virtual-machine:~$ geth attach /home/napier/mynapier/geth.ipc
WARN [03-20|20:54:56.172] Sanitizing cache to Go's GC limits       provided=1024
updated=666
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.23-stable-c9427004/linux-amd64/go1.10.4
coinbase: 0xa09243e009a8e6b7e13edaf876ebb138656d9b9d
at block: 0 (Thu, 01 Jan 1970 01:00:00 BST)
 datadir: /home/napier/mynapier
 modules:  admin:1.0  debug:1.0  eth:1.0  ethash:1.0  miner:1.0  net:1.0  personal:1.0
rpc:1.0 txpool:1.0 web3:1.0

> personal.newAccount("Qwerty1")
" 0xd739a090f842be4f178c6727a68324d6c7608d71"
> web3.eth.accounts
["0x39a18a459b2475925e3014679707e4970a6a836d",
"0x3a5b61aeb069dc7df1b8f1b58e883118ea8bef3f”,                                    ",
"0x617fc4cd56a938f28b02417088f587c1fb4e7ae4",
"0xd739a090f842be4f178c6727a68324d6c7608d71"]
```

We can see we now have four account (three created from the genesis block, and which have some cryptocurrency, and a new one). Take a note of your new account ID (just first four hex characters):

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("0xd739a090f842be4f178c6727a68324d6c7608d71")
0
> eth.getBalance("0x617fc4cd56a938f28b02417088f587c1fb4e7ae4")
30000000
```

What are the balances in each account:

Next unlock the account with the most Ether:

```
> personal.unlockAccount("0x617fc4cd56a938f28b02417088f587c1fb4e7ae4")
Unlock account 0xd739a090f842be4f178c6727a68324d6c7608d71
Passphrase: Qwerty1
true
```

Note you can also use:

personal.unlockAccount("0x617fc4cd56a938f28b02417088f587c1fb4e7ae4","**Qwerty1**"**)**

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

```
> eth.sendTransaction({from: '0x617fc4cd5602417088f587c1fb4e7ae4a938f28b', to:'
0xd739a090f842be4f178c6727a68324d6c7608d71,value:1000})
"0x7b37132db152ab7382eb3e580195c1f1b961fee3e1ffbc64bf7a033336b9e2af"
> eth.getTransaction('0x7b37132db152ab7382eb3e580195c1f1b961fee3e1ffbc64bf7a033336b9e2af')
{
  blockHash: "0x0000000000000000000000000000000000000000000000000000000000000000",
  blockNumber: null,
  from: "0x617fc4cd56a938f28b02417088f587c1fb4e7ae4",
  gas: 90000,
  gasPrice: 1000000000,
  hash: "0x7b37132db152ab7382eb3e580195c1f1b961fee3e1ffbc64bf7a033336b9e2af",
  input: "0x",
  nonce: 0,
  r: "0xc9c1cece4aff8143c09be07dfcff600f657bb561e5d8034dbec692ec5554894e",
  s: "0x721f63984441c4eea870383ed8784d3475f32b740e84ad07ba0e795d5d2aead5",
  to: "0xd4499406c13f0c8601927cfaecb325c5ae2cac6a",
  transactionIndex: 0,
  v: "0x41",
  value: 1000
}
```

What are *r*, *s*, *gas*, and *gasprice*:

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

```
> eth.getBalance("0x617fc4cd5602417088f587c1fb4e7ae4a938f28b ")
288230376151711744
> eth.getBalance("0xd739a090f842be4f178c6727a68324d6c7608d71")
0
```

This is because the miners have not started yet. We can now start the miner and view the balances:

```
> miner.start()
null
> eth.getBalance("0x617fc4cd5602417088f587c1fb4e7ae4a938f28b")
288230376151711744
> eth.getBalance("0xd739a090f842be4f178c6727a68324d6c7608d71")
0
```

We can transfer again:

```
> eth.sendTransaction({from: 0x617fc4cd5602417088f587c1fb4e7ae4a938f28b, to:
'0xce1373ddfa2232dc9ca82d98420be7a2e11962b5',value:100000})
"0x2e25093e25cbf511c2892cb38b45a5c9f6f9b2785774cd5830cf5bd978839165"
> eth.getBalance("0xce1373ddfa2232dc9ca82d98420be7a2e11962b5")
0
> eth.getBalance("0x617fc4cd5602417088f587c1fb4e7ae4a938f28b")
288230376151711744
```

The mining process adds some credits to the initial account:

```
> eth.getBalance("0x617fc4cd5602417088f587c1fb4e7ae4a938f28b")
5288230376151711744

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

After the mining process we see:

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

If we look at the blockchain we see there are two blocks have been created:

```
> eth.blockNumber
2
```

What are the balances in the accounts:

```
What are the balances on the accounts?

How many blocks have been mined?
```

D Creating a contract

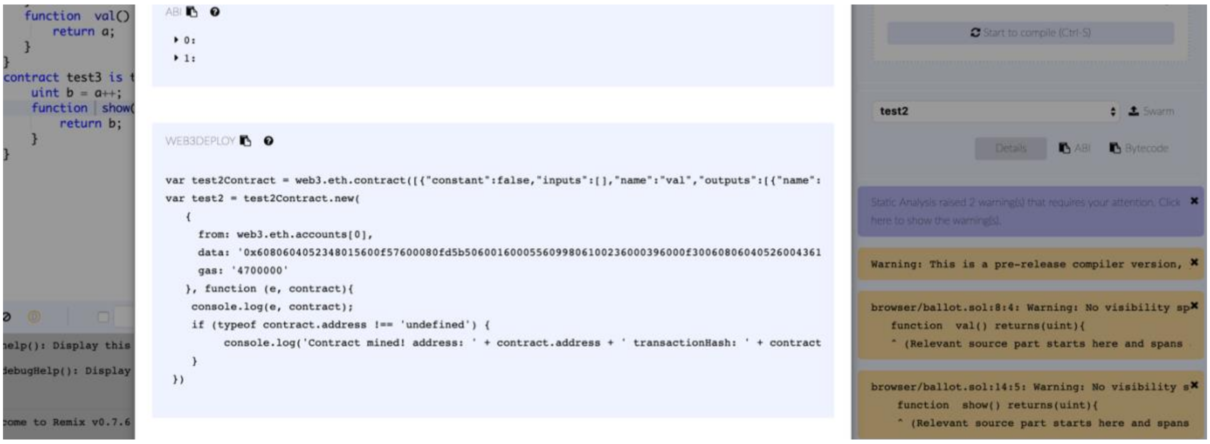
Now let's create a contract. First open up:

http://remix.ethereum.org/

and paste the following code:

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

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



```
  function  val()
      return a;
  }
}
contract test3 is t
   uint b = a++;
   function show(
      return b;
   }
}
```

```
help(): Display this
debugHelp(): Display
come to Remix v0.7.6
```

ABI

```
▸ 0:
▸ 1:
```

WEB3DEPLOY

```
var test2Contract = web3.eth.contract([{"constant":false,"inputs":[],"name":"val","outputs":[{"name":
var test2 = test2Contract.new(
   {
     from: web3.eth.accounts[0],
     data: '0x6080604052348015600f57600080fd5b5060016000556099806100236000396000f30060806040526004361
     gas: '4700000'
   }, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
         console.log('Contract mined! address: ' + contract.address + ' transactionHash: ' + contract
    }
 })
```

Static Analysis raised 2 warning(s) that requires your attention. Click here to show the warning(s).

```
Warning: This is a pre-release compiler version,
```

```
browser/ballot.sol:8:4: Warning: No visibility sp
   function  val() returns(uint){
   ^ (Relevant source part starts here and spans
```

```
browser/ballot.sol:14:5: Warning: No visibility s
    function  show() returns(uint){
    ^ (Relevant source part starts here and spans
```

A sample is:

```
var test2Contract =
web3.eth.contract([{"constant":false,"inputs":[],"name":"val","outputs":[{"name":"","type":"u
int256"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"inputs":[],"pay
able":false,"stateMutability":"nonpayable","type":"constructor"}]);
var test2 = test2Contract.new(
   {
     from: web3.eth.accounts[0],
     data:
'0x60606040523415600e57600080fd5b6001600081905550609f806100246000396000f300606060405260043610
603f576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff1680633
c6bb436146044575b600080fd5b3415604e57600080fd5b6054606a565b6040518082815260200191505060405180
910390f35b600080549050905600a165627a7a7230582062175dc4e9fcfe956cc06a92ee74103af0feae371ddeb5b
b46c65f3f74140a0f0029',
     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 onto our blockchain:

```
> 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:

```
> 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()
}
> 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:

```
> test2.val()
"1"
> test3.val()
"2"
```

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) public view 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
```

Now, we will install the Solidity compiler on your Ubuntu host:

```
sudo add-apt-repository ppa:ethereum/ethereum
sudo apt-get install solc
```

Create your Solidity program on Ubuntu (1.sol), and then compile it to a binary format:

```
napier@napier-virtual-machine:~$ solc --bin 1.sol
1.sol:4:4: Warning: Function state mutability can be restricted to pure
   function sqrt(uint x) public view returns (uint y) {
   ^ (Relevant source part starts here and spans across multiple lines).


======= 1.sol:mymath =======
Binary:
608060405234801561001057600080fd5b5060de8061001f6000396000f3fe6080604052348015600f57600080fd5b5060043610602857600
0
          3560e01c8063677342ce14602d575b600080fd5b60566004803603602081101560415760008 0fd5b8101908080359060200190929
          190505050606c565b60405180828152602001915050604051809103 90f35b6000806002600184018115156 07d57fe5b04905082915
```

```
05b8181101560ac5780915060028182858115156609b57fe5b040181151560a557fe5b0490506084565b5091905056fea165627a7a7
23058207aa8e5d6d41b8fd664f343f3f2bb080b89ab85d06bd9ded50b8eb65a747362dd0029
```

What can you observe from the code produced? How could you deploy this?

Now compile with the ABI option. What information does the output give?

How does this relate to the code your previously created from the Web site?

Commands

```
sudo apt-get install software-properties-common
sudo add-apt-repository -y ppa:ethereum/ethereum
sudo apt-get install ethereum

sudo geth --datadir=mynapier init custom.json
sudo geth --datadir=mynapier account new
sudo geth --datadir=mynapier --networkid=15

sudo geth --datadir=mynapier account new
personal.newAccount("Qwerty1")
web3.eth.accounts
personal.unlockAccount("0x39a18a459b2475925e3014679707e4970a6a836d", “Qwerty1”)
seth.sendTransaction({from: "0x39a18a459b2475925e3014679707e4970a6a836d ", to:
"0xa2cda8c68259cc314bd12b67873ce7380b3dc496",value:1000000})
eth.getBalance("0x39a18a459b2475925e3014679707e4970a6a836d")
eth.getBalance("0xa2cda8c68259cc314bd12b67873ce7380b3dc496")
eth.blockNumber
web3.eth.defaultAccount = '0x821eacc2a570c1aeb9b5aa64b5b915d4c1e1f3ee'
loadScript('mycontract.js')

sudo add-apt-repository ppa:ethereum/ethereum
sudo apt-get install solc

solc --bin 1.sol
```