Preparation for Blockchain Programming

Heung-No Lee
2021/03/22

Purpose of this lecture:

1. Introduction to GIST LV Testnet
   A. Download BIT-ECC (DeSecure Blockchain)
2. Installation of Virtual Machine and LINUX into Windows
   A. Anaconda/Python/C++ Installation
   B. Installation of BIT-ECC Core into your local machine
   C. Testing of BIT-ECC APIs
   D. BIT-ECC Programming
   E. Learning the LVnet

We will use this LVnet throughout the whole course to learn the Bitcoin protocol and APIs via homeworks and projects.

I. DECENTRALIZED SECURE (DESECURE) BLOCKCHAIN

DeSecure Blockchain is made by INFONET(Director Heung-No Lee). It is a blockchain aiming to achieve decentralization and security. See its intro here.

Decentralization is sought by anti-ASIC consensus algorithm. ASIC-chips are the culprit of centralization of the mining market. A few mining corporations dominate the mining market. Anti-ASIC property is achieved by utilization of error-correction codes in proof-of-work. We have used an error-correction code, called low-density parity-check codes, to augment the Secure Hash Function, and have made a new PoW called ECCPoW. The key new property is to make the ever changing crypto-puzzles; as a result, an ASIC chips cannot be made.

Using ECCPoW, security is maintained. It is to be noted that the proof-of-work system is the most secure consensus mechanism, compared to recently proposed attempts in which proof-of-work is given up for seeking scalability and transaction speeds. Proof-of-stakes and Byzantine fault tolerance based consensus systems are wrong approaches as global cryptocurrencies. They are not only less secure but run into other problems.

Currently, there are two DeSecure blockchain projects:

1. BIT-ECC: It is a new Bitcoin whose consensus part is replaced with ECCPoW.
2. ETH-ECC: It is a new Ethereum whose consensus part is replaced with ECCPoW.

These two packages, BIT-ECC and ETH-ECC, are downloadable from our github repository: https://github.com/cryptoecc/. For papers, you can visit our Publications page in https://infonet.gist.ac.kr.

The DeSecure Blockchain github below shows several other supporting directories such as

1. BIT-ECC: https://github.com/cryptoecc/bitcoin_ECC This is an official Bitcoin C++ implementation of BIT-ECC protocol. One can use this protocol to start a new BIT-ECC blockchain network.
2. Bitcoin_ECC_BlockGenerationProgram: This is for a BIT-ECC block generation algorithm (mining algorithm). One can run this BlockGen algorithm to join our current GIST LV Testnet project.

DeSecure Blockchain

Repositories 10 | Packages | People | Projects

Find a repository... | Type | Language | Sort

bitcoin_ECC

Forked from bitcoin/bitcoin

Bitcoin Core integration/staging tree

C++ MIT 28,076 1 0 0 Updated 9 days ago

ETH-ECC

Official Go implementation of the ECCPoW Ethereum protocol of INFONET

Go LGPL-3.0 2 0 0 11 Updated 11 days ago

Bitcoin_ECC_BlockGenerationProgram

Forked from mint-young/Bitcoin_ECC_MiningProgram

C++ 1 0 0 0 Updated 18 days ago

btc-rpc-explorer

Forked from jhonggg/btc-rpc-explorer

Simple, database-free, self-hosted Bitcoin blockchain explorer, via RPC. Built with Node.js, express, bootstrap-v4.

JavaScript MIT 519 0 0 0 Updated on 25 Jan

wallet

Forked from bitpay/wallet

Bitpay Wallet (formerly Copay) is a secure Bitcoin and other crypto currencies wallet platform for both desktop and mobile devices.

TypeScript MIT 1,632 0 0 0 Updated on 15 Jan

Depreciated_go-ethereum_ECC

Forked from Onther-Tech/go-ethereum

Please move to new repository ->

Go LGPL-3.0 10,631 1 0 0 Updated on 9 Jun 2020

ECCPOW-LDPC

C++ 4 0 0 0 Updated on 20 Feb 2020

introduction

Introduction about DeSecure

0 0 0 0 Updated on 10 Feb 2020

ethereum_ECC

Python MIT 0 0 0 0 Updated on 21 Jun 2019

ECCPoW

This is GIST project

C++ 0 0 0 0 Updated on 11 May 2019

Top languages

C++ Go JavaScript Python TypeScript

People

This organization has no public members. You must be a member to see who's a part of this organization.

3. Explorer Program: btc-rpc-explorer This is an explorer algorithm for BIT-ECC made from a Bitcoin explorer. Using this program, one can browse blocks and view block details.
4. ECCPoW: **cryptoecc/ECCPOW-LDPC** This one discusses the details of ECCPoW.
5. Wallets: https://github.com/cryptoecc/wallet In order to generate a transaction, one needs to have a wallet. This repository shows a bitpay wallet. To make this wallet working for BIT-ECC and ETH-ECC networks, one need to modify the source codes. Thus, a wallet application for Android or ios phones is not ready yet. A PC version is available for the GIST LV BIT-ECC network.
6. ETH-ECC: https://github.com/cryptoecc/ETH-ECC This is an official GO implementation of the ECCPoW Ethereum protocol of INFONET. One can start a new ETH-ECC blockchain using this protocol. See our paper titled as Ethereum ECCPoW, at https://arxiv.org/abs/2101.10729.

II. GIST LV TESTNET (LVNET)

LiberVance is a tech venture launched by Prof. Heung-No Lee in January 2020. The name of the venture comes from two words, **Liber**ty and ad**Vance**. The aim of LiberVance is to introduce the lab technologies developed by my research team.

LV has started testing the BIT-ECC protocol extensively. The explorer for the TESTNET http://explorer.libervance.com:8080/ shows that the block height is 34,890 on 2021/03/21, 17:48:50, the universal time.

The TESTNET is for the BIT-ECC protocol which can be downloaded from https://github.com/cryptoecc/bitcoin_ECC. There are many different versions. The most recent one we are working on is ecc-0.1.2 version.

We will use the LVnet to learn Bitcoin protocols and Bitcoin blockchains.

1. BIT-ECC is different from Bitcoin only in the consensus part. Thus, there are some changes in the block header, including Network ID, Difficulty, Port number, and the format of hash values. The rest, the networking part and the wallet part, are exactly the same. All the APIs from Bitcoin are the same for BIT-ECC; thus we can use this testnet to learn Bitcoin APIs.
2. We can modify the Bitcoin APIs aiming to further improve the BIT-ECC.

There you can find 9 documents one can use to start learning the LVnet. We can use them to learn the BIT-ECC protocol and the LVnet. The LVnet currently up at running is based on the BIT-ECC protocol version ecc-0.1.2.

1. BIT-ECC: Environment Set Up, Installation, and Testing: The aim of this document is to instruct the readers how to set up the virtual environment, run and test the BIT-ECC [Bitcoin Error Correction Code PoW Blockchain] on local machine.
   A. Installation of VMware Workstation: We will install LINUX operating system on top of the Windows OS which is the main OS of your local machine. LINUX is very convenient for developers.
   B. You need to download BIT-ECC protocol and install it into your local LINUX machine.
   C. In your local machine, you can test your private network. You can run a BIT-ECC and run a BIT-ECC server.
   D. You can use this BIT-ECC server to see basic functions of BIT-ECC server such as sending transactions, generating blocks, and verifying the inclusion of sent transaction in a published block.
   E. To start your own new BIT-ECC blockchain, you need to make up a genesis block, set up network ID, port number, etc.
   F. We can utilize the currently running GIST LV TESTNET (LVnet) for learning the Bitcoin protocols in this class.
2. Generate the Genesis Block of the BIT-ECC: The aim of this document is to instruct the readers how to make the BIT-ECC genesis block. A genesis block is needed to start your own blockchain. We will have to modify some parameters values that are related to genesis block. 'chainparams.cpp' file is used to modify values at the '~/bitcoin_ECC/src' folder. The current version ecc-0.1.2 version contains this modification (need to check). After the modification is made, the blockchain core needs to be compiled and executed again. You can leave a comment on the genesis

block. For example, Huisoo one of the programmers at INFONET left a message on the genesis block; see if you can spot it.

3. BIT-ECC: Block Generation Program. This is an instruction to run a block generation program (mining) into your local machine. It requires the correct version of the BIT-ECC installed into your local machine. Coins will be given as reward for each block generation. You need to a BIT-ECC address and get the minted coins to this address. You can create a new BIT-ECC address using the standard procedure of Bitcoin address creation.
4. BIT-ECC: Check transactions using Bitcoin-core wallet. You can use this to create your new BIT-ECC address (the same as the Bitcoin address mechanism), you're your address to receive coins, and send coins via a new transaction.
5. BIT-ECC: Explorer. This file shows how the BIT-ECC Explorer was set up. You can use this file to set up a new BIT-ECC explorer. See this link as well.
6. How to add fixed seed nodes: This file discusses how to add IP addresses to `contrib/seeds/nodes.main.txt` file located in bitcoin-ECC directory. Fixed seed nodes are useful providing continuous full node service to the entire BIT-ECC network. Other nodes can come and go while these nodes are always there.

BIT-ECC: Environment Setup, Installation and Testing

Wahidur Rahman
28th December 2020
INFONET (Prof. Heung-No Lee), GIST
https://github.com/infonetGIST
https://infonet.gist.ac.kr/

Purpose: The aim of this document is to instruct the readers how to set up the virtual environment, run and test the BIT-ECC [Bitcoin Error Correction Code PoW Blockchain] on local machine.

This document has the following contents.

Contents:

1. Environment Setup

In this chapter, we will explain how to install VMware Workstation and how-to setup Ubuntu operating system on the virtual machine. The setup has been prepared for The Windows 10 environment.

1.1 Installation of VMware Workstation:

Download the latest version of a VMware Workstation [The current version is 16.0, downloaded on December 20th, 2020] from the below link and follow the step-by-step instruction to install it on your local machine.

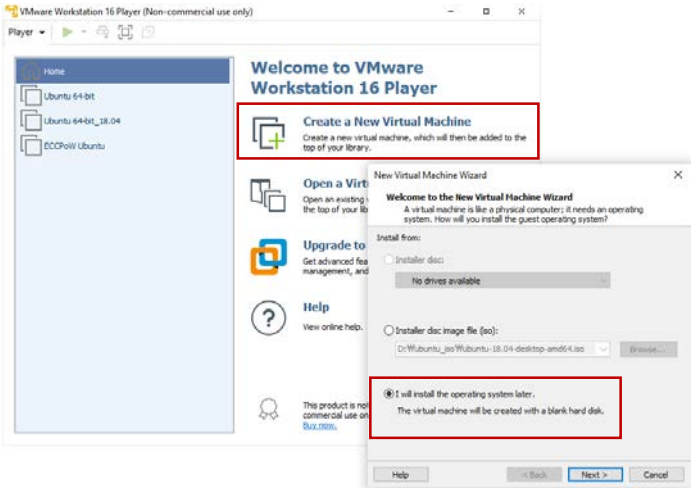
VMware Workstation is available both for Windows and Linux operating systems.
https://www.vmware.com/kr/products/workstation-player/workstation-player-evaluation.html

1.2 Download Ubuntu ISO Image file:

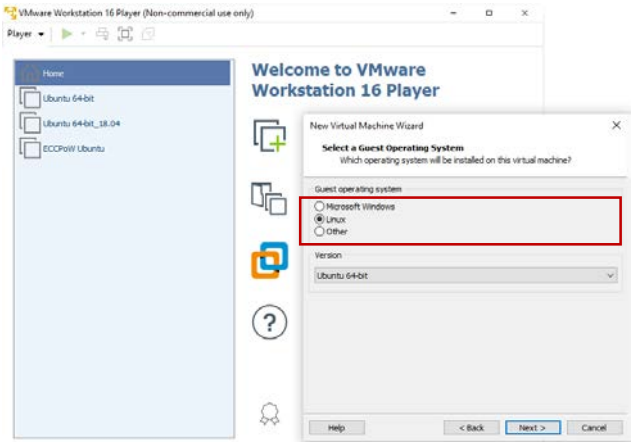
Download the Ubuntu 18.04 LTS iso image file from the below link:
https://releases.ubuntu.com/18.04/

1.3 Setup the Linux operating system on VMware Workstation

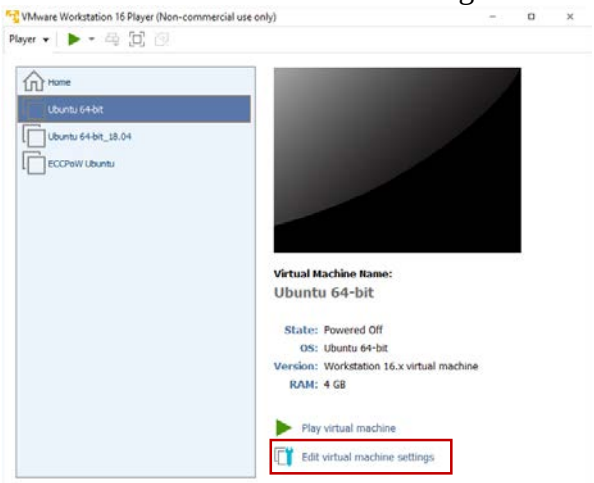
Select the ‘Create a New Virtual Machine’ tab, then go next. Put a check mark on the bud ‘I will install the operating system later.’ This will create a virtual machine with a blank hard disk.



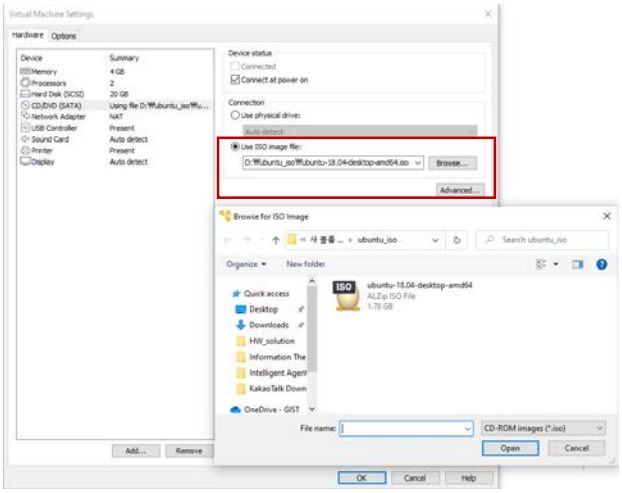
On the next wizard, select the Linux as a guest operating system and keep the version Ubuntu 64-bit.



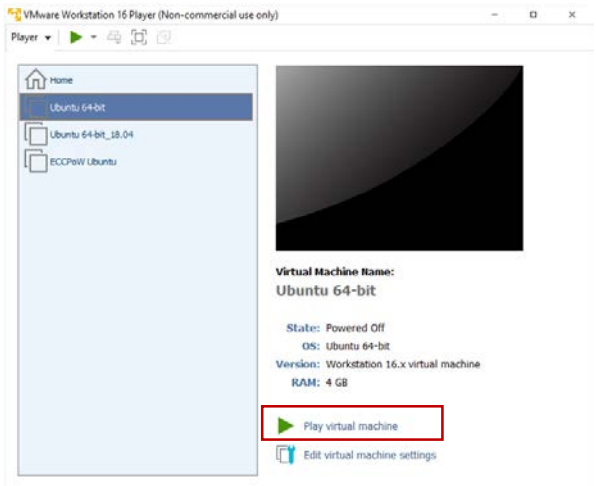
If not required, then keep the installation location as default. Select the disk space as per your requirement. Now select the ‘Edit virtual machine settings.’



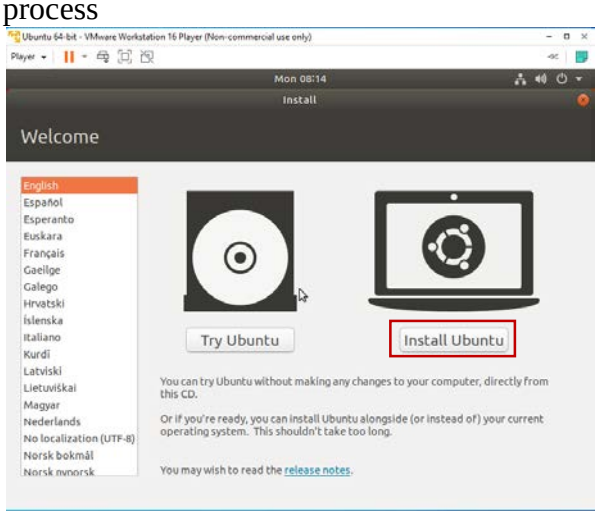
Change the memory and processors as per your requirements. On the CD/DVD (SATA) tab go for the connection tab and use ISO image file as a source of the operating system



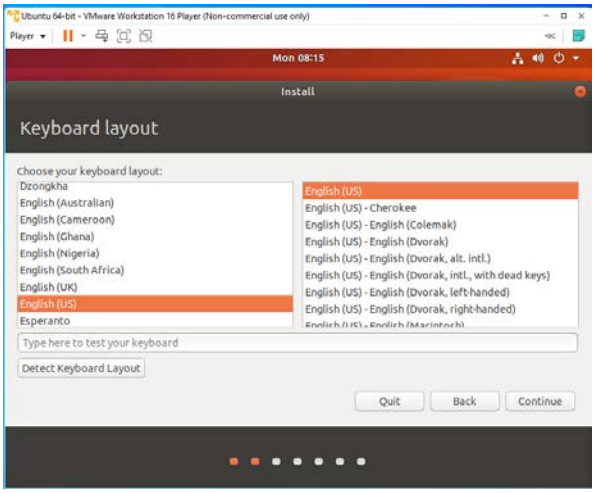
To start the installation process, click on the tab ‘Play virtual machine’ and follow the instructions.



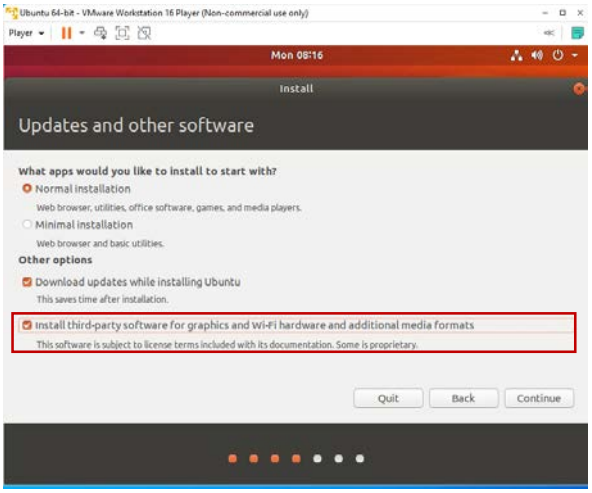
For Ubuntu Setup on virtual machine first select your desire language and select ‘Install Ubuntu’ to start the process



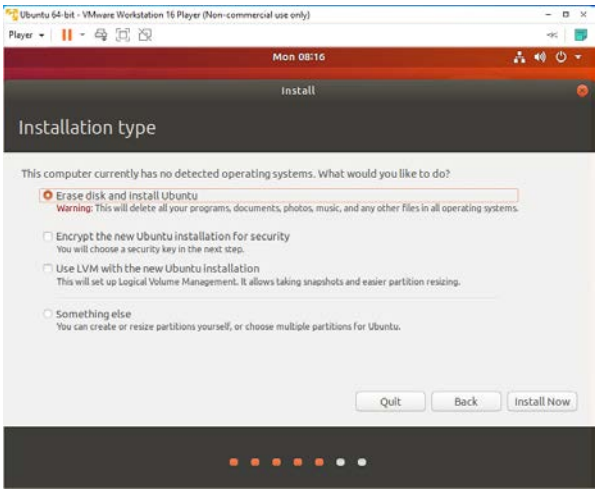
Select your preferable keyboard layout.



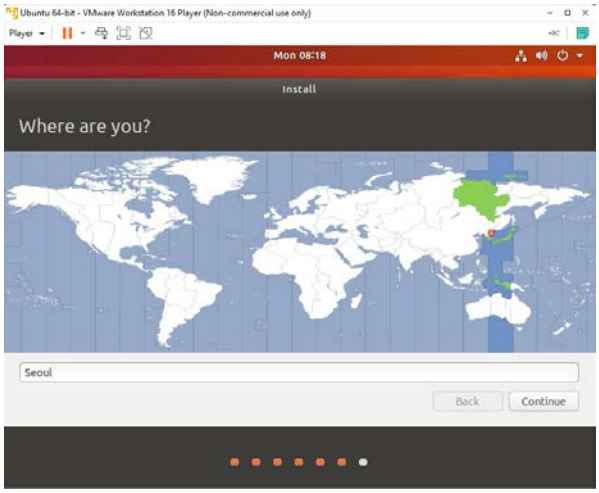
For update and other software check the ‘install third party software’ and keep other options as default



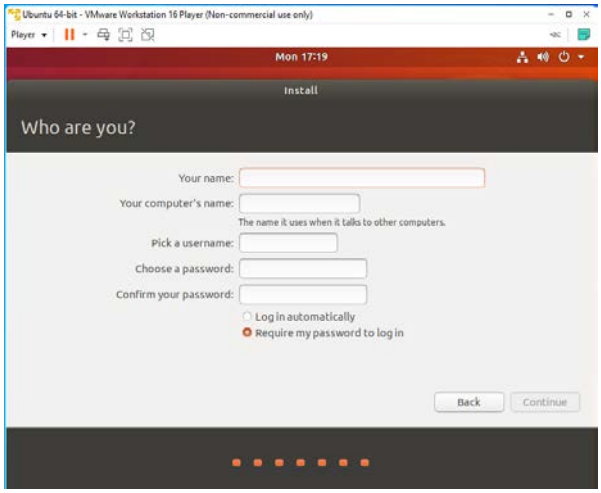
Select ‘Erase disk and install Ubuntu’ and click ‘Install Now’.



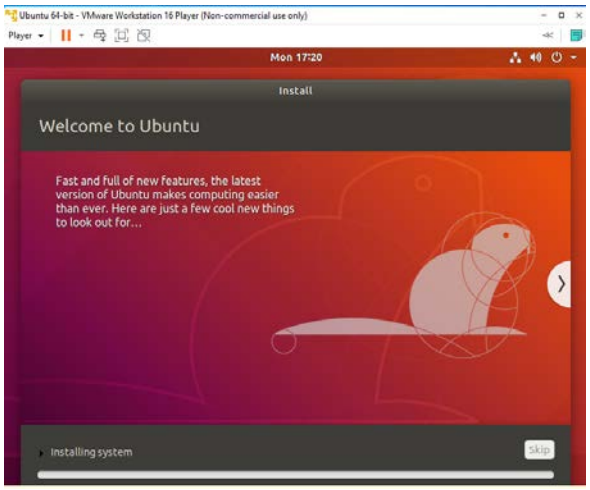
Next select your desire time zone and click on the ‘Continue’ tab.



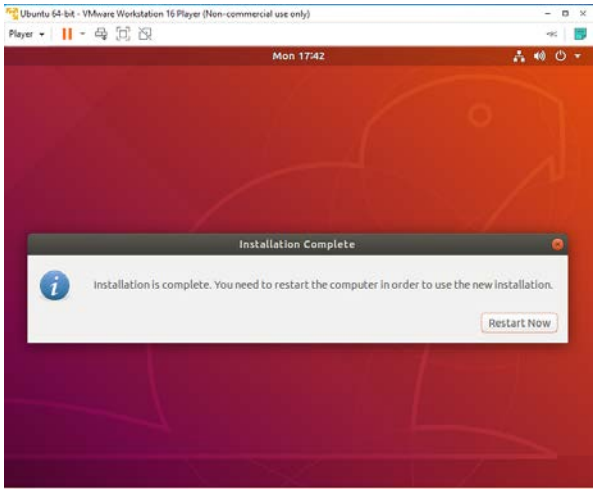
Select the machine name and password.



Now installation process will start, and it will take few minutes to complete the whole installation process.



Click ‘Restart Now’ to restart the machine for initiating the new process.



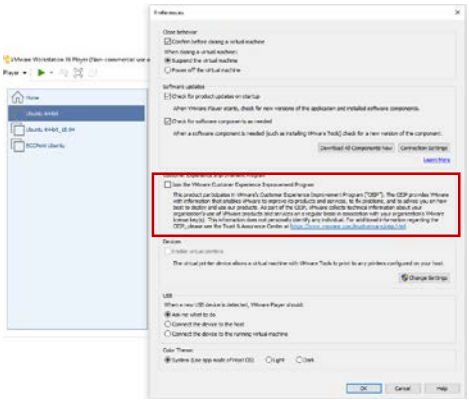
1.4 Two basic commands to fix the screen size.

Open a new terminal and run the below commands.

1. `$ sudo apt-get install open-vm-tools`.
2. `$ sudo apt-get install open-vm-tools-desktop`.

1.5 Enable Copy and Paste Feature in VMware Workstation

Player > File > Performances > Software update > Download all components now.



2. Installation of BIT-ECC

In this chapter, we will describe how to download, install, and run the BIT-ECC on your local machine. Beside that we will install some necessary packages that are prerequisite for this set up.

2.1 Installation of build tool:

```
1. $ sudo apt-get install build-essential libtool autotools-
   dev automake pkg-config bsdmainutils
2. $ sudo apt-get install libqrencode-dev autoconf openssl
   libssl-dev libevent-dev libminiupnpc-dev
```

2.2 Installation of boost library:

```
1. $ sudo apt-get install libboost-all-dev
```

2.3 Installation of Berkeley DB:

```
1. $ sudo apt-get install software-properties-common
```

```
2. $ sudo add-apt-repository ppa:bitcoin/bitcoin
3. $ sudo apt-get update
4. $ sudo apt-get install libdb4.8-dev libdb4.8++-dev
```

2.4 Installation of qt-wallet:

```
1. $ sudo apt-get install libqt5gui5 libqt5core5a libqt5dbus5
   qttools5-dev qttools5-dev-tools libprotobuf-dev protobuf-
   compiler libqrencode-dev
```

2.5 Installation of git:

```
1. $ sudo apt install git
```

2.6 Download the most recent version of BIT-ECC blockchain core:

```
1. $ git clone https://github.com/cryptoecc/bitcoin_ECC.git
2. $ cd bitcoin_ECC
3. $ git checkout ecc-0.1.2
```

2.7 Build the source code:

```
1. $ cd bitcoin_ECC
2. $ ./autogen.sh
3. $ ./configure
4. $ make
5. $ sudo make install
```

2.8 Execute the BIT-ECC core:

```
1. $ bitcoind -txindex –daemon
```

Please see the next chapter.

3. Test Private Network

Open a new terminal under /bitcoin_ECC directory and do the following process to test the private network.

3.1 Execute BIT-ECC/ Run Bitcoin server:

```
1. $ bitcoind -txindex –daemon
```

```
wahid@wahid:~$ cd bitcoin_ECC
wahid@wahid:~/bitcoin_ECC$ bitcoind -txindex -daemon
Bitcoin server starting
```

3.2 Get Block count:

1. `$ bitcoin-cli getblockcount`

```
wahid@wahid:~/bitcoin_ECC$ bitcoin-cli getblockcount
8495
```

Note: This block count will vary time to time. It is always an increasing number.

3.3 Generate an account/ Generate public address:

1. `$ bitcoin-cli getnewaddress`

```
wahid@wahid:~/bitcoin_ECC$ bitcoin-cli getnewaddress
3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw
```

3.4 Generate 10 new blocks:

1. `$ bitcoin-cli generatetoaddress 10 3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw`

```
wahid@wahid:~/bitcoin_ECC$ bitcoin-cli generatetoaddress 10 3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw
[
  "63fe74965bed18c4dc4ec729b17cabc655f22155c806e0e28b700953995e88dc",
  "2e45d535bc534e5f8d63b65a0e8c33582ab4dd4a4ed4680083e50e1e8e661deb",
  "380080fddac154fb6ff7777aa962bd2c672a9bbb21fb1d41f09e92e6e6b118de",
  "9f753058701ac00d175e6a728901b5a25fe370ec306eb1814dba93098f5fdcea",
  "745aeb57670c41b38fec22b5c11d4834e193fb40350d29720405f98d54f54103",
  "a20888294db16eef1774ae630780abf46ca0c43ff270d49c458be8e169da0a9d",
  "bc494c6673c19df32b8b71419db662ff62b51aead7e88a058638e708bcac536e",
  "470cd2b7a7e067ee9020ea86f7eb805e6bba2879d137067cbb757c176447ad2a",
  "f6449d3df5099b7e9c82e8c032962267bfa15719e2ccaa402c2c44562090550f",
  "fd4fbb26957029e1e30c64c10cf320198bf9ad4ad1ca379640af9b4c4d371a2c"
]
```

3.5 Check blockchain information:

1. `$ bitcoin-cli getblockchaininfo`
2. `$ bitcoin-cli getbalance`

3.6 Send Transaction:

Executing a transaction activity is the combination of the below four activities.

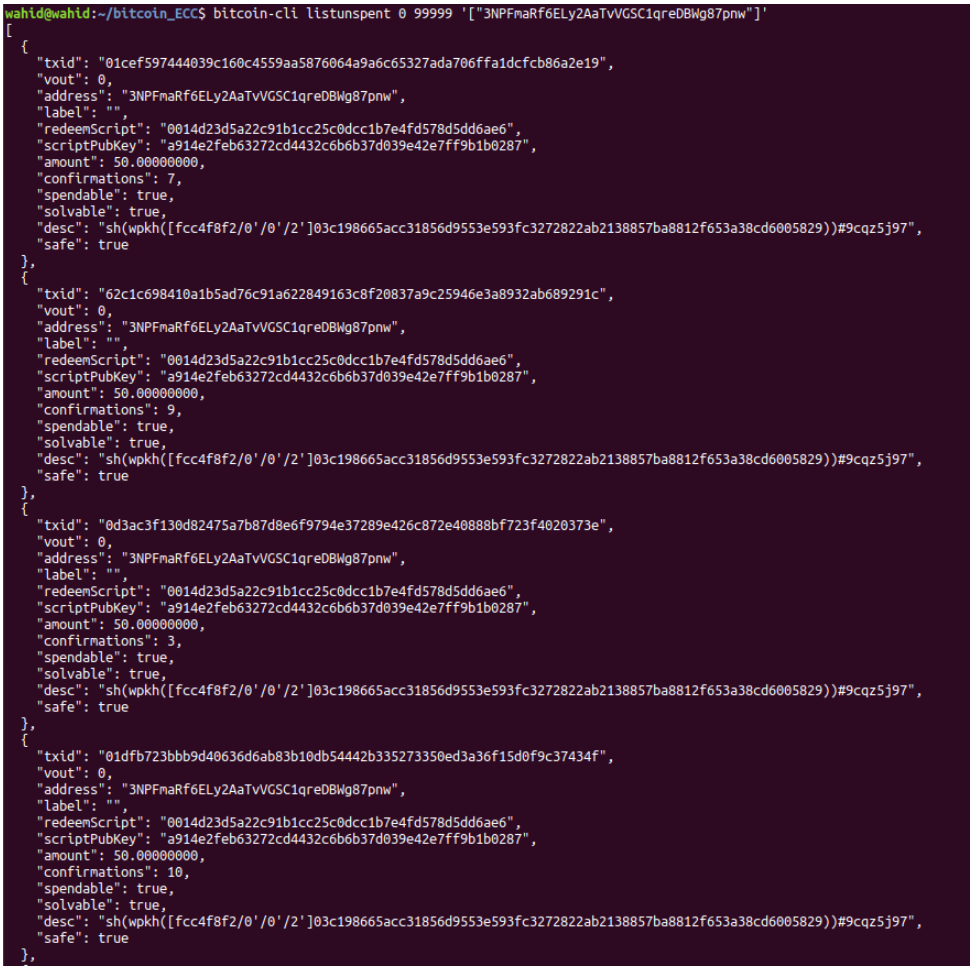
1. listunspent
2. createrawtransaction
3. signrawtransaction
4. sendrawtransaction

Here,

Alice public address [Payer]: 3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw
Bob public address [Payee]: 3GZ3dAMprHiEk9DUF6m8U8BCaupqiMhNDL
Transfer amount: 0.025 BTC-BTC

3.6.1 Listunspent**:** Show confirmed outputs (unspent) in Alice address:

1. `$ bitcoin-cli listunspent 0 99999 '["3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw"]'`



3.7 Create Raw Transaction:

The input field includes:

1. *Txid: 01cef597444039c160c4559aa5876064a9a6c65327ada706ffa1dcfcb86a2e19*
2. *Recipient pub address: 3GZ3dAMprHiEk9DUF6m8U8BCaupqiMhNDL*
3. *Transfer amount: 0.025*
4. *Sender pub address: 3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw*
5. *Amount change: 49.9745*

1. `$ bitcoin-cli createrawtransaction '[{"txid":"01cef597444039c160c4559aa5876064a9a6c65327ada706ffa1dcfcb86a2e19","vout":0}]'`

```
'{"3GZ3dAMprHiEk9DUF6m8U8BCaupqiMhNDL":0.025,
"3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw":49.9745}'
```

```
wahid@wahid:~/bitcoin_ECC$ bitcoin-cli createrawtransaction '[{"txid":"01ce
f597444039c160c4559aa5876064a9a6c65327ada706ffa1dcfcb86a2e19","vout":0}]' '
{"3GZ3dAMprHiEk9DUF6m8U8BCaupqiMhNDL ":0.025, "3NPFmaRf6ELy2AaTvVGSC1qreDBW
g87pnw":49.9745}'
0200000001192e6ab8fcdca1ff06a7ad2753c6a6a9646087a59a55c460c139404497f5ce010
000000000ffffffff02a02526000000000017a914a307fa7979dbb9611079b9a0d18fbde250
f3d12d871009df290100000017a914e2feb63272cd4432c6b6b37d039e42e7ff9b1b0287000
00000
```

3.8 Sign Raw Transaction:

3.8.1 Check the private key:

```
1. $ bitcoin-cli dumpprivkey
   3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw
```

```
wahid@wahid:~/bitcoin_ECC$ bitcoin-cli dumpprivkey 3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw
L3VUeRCzU6iRDgF92pkv5p27VBCgsqPuq5fY1JtNe6u1nbBpAFSR
```

3.8.2 Sign the transaction in the serialized transaction format using private key:

```
1. $ bitcoin-cli signrawtransactionwithkey
   "0200000001192e6ab8fcdca1ff06a7ad2753c6a6a9646087a59a55c4
   60c139404497f5ce010000000000ffffffff02a02526000000000017a
   914a307fa7979dbb9611079b9a0d18fbde250f3d12d871009df290100
   000017a914e2feb63272cd4432c6b6b37d039e42e7ff9b1b028700000
   000"
   '["L3VUeRCzU6iRDgF92pkv5p27VBCgsqPuq5fY1JtNe6u1nbBpAFSR"]
   '
```

```
wahid@wahid:~/bitcoin_ECC$ bitcoin-cli signrawtransactionwithkey "020000001192e6ab8fcdca1ff06a7ad2753c6a6a9646087a59a55c460c139404497f5ce010000000000ffffffff02a02526000000000017a914a307fa7979dbb9611079b9a0d18fbde250f3d12d871009df290100000017a914e2feb63272cd4432c6b6b37d039e42e7ff9b1b028700000000" '["L3VUeRCzU6iRDgF92pkv5p27VBCgsqPuq5fY1JtNe6u1nbBpAFSR"]'
{
  "hex": "02000000000101192e6ab8fcdca1ff06a7ad2753c6a6a9646087a59a55c460c139404497f5ce010000000017160014d23d5a22c91b1cc25c0dcc1b7e4fd578d5dd6ae6ffffffff02a02526000000000017a914a307fa7979dbb9611079b9a0d18fbde250f3d12d871009df290100000017a914e2feb63272cd4432c6b6b37d039e42e7ff9b1b02870247304402205430c05d66ad4570aa187127989544912214a76b0b65bbbbe546c5d8ec72d24470220157b63e6fec7c7e6e13ff0a3ce37462c925aa54baafaa5cfc92f2a420420b123012103c198665acc31856d9553e593fc3272822ab2138857ba8812f653a38cd600582900000000",
  "complete": true
}
```

3.9 Send Raw Transaction:

```
1. $ bitcoin-cli sendrawtransaction
   02000000000101192e6ab8fcdca1ff06a7ad2753c6a6a9646087a59a5
   5c460c139404497f5ce010000000017160014d23d5a22c91b1cc25c0d
```

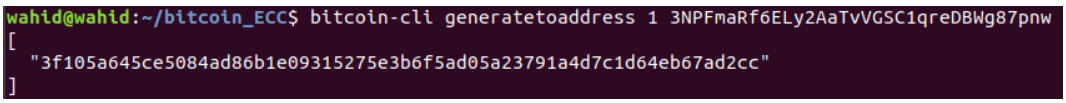
```
cc1b7e4fd578d5dd6ae6ffffffff02a025260000000000017a914a307f
a7979dbb9611079b9a0d18fbde250f3d12d871009df290100000017a9
14e2feb63272cd4432c6b6b37d039e42e7ff9b1b028702473044022050
43c05d66ad4570aa187127989544912214a76b0b65bbbbe546c5d8ec7
2d24470220157b63e6fec7c7e6e13ff0a3ce37462c925aa54baafaa5c
fc92f2a420420b123012103c198665acc31856d9553e593fc3272822a
b2138857ba8812f653a38cd600582900000000
```

3.9.1 Verification of the issued transaction in a block.

1. Generate one block
2. Get block information
3. Get transaction information

3.9.1.1 Generate one block: Generate a new block with sender public address.

1. `$ bitcoin-cli generatetoaddress 1 3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw`



3.9.1.2 Generate block information: For the block information run the below command.

1. `$ bitcoin-cli getblock 3f105a645ce5084ad86b1e09315275e3b6f5ad05a23791a4d7c1d64eb67ad2cc`

3.9.1.3 Get transaction information: For the transaction detail run the below command.

1. `$ bitcoin-cli gettransaction f01791cd253e1bef474498905d4f40f127a28e1cf94dee8239ce0e04cfb91dff`

```
wahid@wahid:~/bitcoin_ECC$ bitcoin-cli gettransaction f01791cd253e1bef474498905d4f40f127a28e1cf94dee8239ce
0e04cfb91dff
{
  "amount": 0.00000000,
  "fee": -0.00050000,
  "confirmations": 1,
  "blockhash": "3f105a645ce5084ad86b1e09315275e3b6f5ad05a23791a4d7c1d64eb67ad2cc",
  "blockindex": 1,
  "blocktime": 1608614156,
  "txid": "f01791cd253e1bef474498905d4f40f127a28e1cf94dee8239ce0e04cfb91dff",
  "walletconflicts": [
  ],
  "time": 1608613252,
  "timereceived": 1608613252,
  "bip125-replaceable": "no",
  "details": [
    {
      "address": "3GZ3dAMprHiEk9DUF6m8U8BCaupqiMhNDL",
      "category": "send",
      "amount": -0.02500000,
      "label": "",
      "vout": 0,
      "fee": -0.00050000,
      "abandoned": false
    },
    {
      "address": "3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw",
      "category": "send",
      "amount": -49.97450000,
      "label": "",
      "vout": 1,
      "fee": -0.00050000,
      "abandoned": false
    },
    {
      "address": "3GZ3dAMprHiEk9DUF6m8U8BCaupqiMhNDL",
      "category": "receive",
      "amount": 0.02500000,
      "label": "",
      "vout": 0
    },
    {
      "address": "3NPFmaRf6ELy2AaTvVGSC1qreDBWg87pnw",
      "category": "receive",
      "amount": 49.97450000,
      "label": "",
      "vout": 1
    }
  ],
  "hex": "02000000000101192e6ab8fcdca1ff06a7ad2753c6a6a9646087a59a55c460c139404497f5ce010000000017160014d2
3d5a22c91b1cc25c0dcc1b7e4fd578d5dd6ae6ffffffff02a02526000000000017a914a307fa7979dbb9611079b9a0d18fbde250f3
d12d871009df290100000017a914e2feb63272cd4432c6b6b37d039e42e7ff9b1b028702473044022054 3c05d66ad4570aa1871279
89544912214a76b0b65bbbbe546c5d8ec72d24470220157b63e6fec7c7e6e13ff0a3ce37462c925aa54baafaa5cfc92f2a420420b1
23012103c198665acc31856d9553e593fc3272822ab2138857ba8812f653a38cd600582900000000"
}
```

BIT-ECC:
Generate Genesis Block

Ha-young Park, Wahidur, Huisu Kim
8th January 2021
INFONET (Prof. Heung-No Lee), GIST
https://github.com/infonetGIST
https://infonet.gist.ac.kr/

Purpose: The aim of this document is to instruct the readers how to generate BIT-ECC genesis block of at local machine.

Contents:

1. Change values of genesis block

We will modify some parameters values that are related to genesis block. 'chainparams.cpp' file is used to modify values at '~/bitcoin_ECC/src' folder. In the 'chainparams.cpp', there are two override functions and three classes for each network. The override function is 'CreateGenesisBlock' and the classes of networks are mainnet, testnet(v3) and regression test.

1.1 Modify the 'CreateGenesisBlock' function:

Modify the value of ‘txNew.vout[0].nValue’ parameter as 50. It means that the reward point for generating a new block will be 50 BTC.

```
static CBlock CreateGenesisBlock(const char* pszTimestamp, const CScript& genesisOutputScript, uint32_t nTime, uint32_t nNonce, uint32_t nBits, int32_t
nVersion, const CAmount& genesisReward)
{
    CMutableTransaction txNew;
    txNew.nVersion = 1;
    txNew.vin.resize(1);
    txNew.vout.resize(1);
    txNew.vin[0].scriptSig = CScript() << 486604799 << CScriptNum(4) << std::vector<unsigned char>((const unsigned char*)pszTimestamp, (const unsigned
char*)pszTimestamp + strlen(pszTimestamp));
    txNew.vout[0].nValue = 50;
    txNew.vout[0].scriptPubKey = genesisOutputScript;

    CBlock genesis;
    genesis.nTime    = nTime;
    genesis.nBits    = nBits;
    genesis.nNonce   = nNonce;
    genesis.nVersion = nVersion;
    genesis.vtx.push_back(MakeTransactionRef(std::move(txNew)));
    genesis.hashPrevBlock.SetNull();
    genesis.hashMerkleRoot = BlockMerkleRoot(genesis);
    return genesis;
}
```

Modify the value of 'pszTimestamp' parameter. Write any sentence you want or just modify the date as current date.

```
static CBlock CreateGenesisBlock(uint32_t nTime, uint32_t nNonce, uint32_t nBits, int32_t nVersion, const CAmount& genesisReward)
{
    const char* pszTimestamp = "The Times 08/Jan/2021 New hope for Covid patients";
    const CScript genesisOutputScript = CScript() <<
ParseHex("04678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5f") <<
OP_CHECKSIG;
    return CreateGenesisBlock(pszTimestamp, genesisOutputScript, nTime, nNonce, nBits, nVersion, genesisReward);
}
```

1.2 Modify the parameters of 'CreateGenesisBlock' function:

Each class has 5 parameters- ‘nTime’, ‘nNonce’, ‘nBits’, ‘nVersion’ and ‘genesisReward’. Change the first and the second one [‘nTime’, ‘nNonce’]. For UNIX time please follow the mentioned link [https://www.unixtimestamp.com/] and put any random integer number for the ‘nNonce’ value.

```
int init_level = 1;
genesis = CreateGenesisBlock(1558627231, 399, init_level, 1, 50 * COIN);
```

```
int init_level = 1;
genesis = CreateGenesisBlock(1610072113, 1012, init_level, 1, 50 * COIN);
//printf("\nmainnet with level = %d\n",init_level);
//printf("set is constructed from %d to %d with step 2\n",ldpc_level_table[init_level].from, ldpc_level_table[init_level].to);
//printf("n : %d\t wc : %d\t wr : %d\n", ldpc_level_table[init_level].n, ldpc_level_table[init_level].wc, ldpc_level_table[init_level].wr);
```

Repeat this process for other networks. If 'init_level' is different with mainnet, change it same as mainnet.

[testnet(v3)]

```
int init_level = 1;
genesis = CreateGenesisBlock(1610072113, 1012, init_level, 1, 50 * COIN);
//printf("\ntestnet with level = %d\n",init_level);
//printf("set is constructed from %d to %d with step 2\n",ldpc_level_table[init_level].from, ldpc_level_table[init_level].to);
//printf("n : %d\t wc : %d\t wr : %d\n", ldpc_level_table[init_level].n, ldpc_level_table[init_level].wc, ldpc_level_table[init_level].wr);
```

[regression test]

```
int init_level = 1;
genesis = CreateGenesisBlock(1610072113, 1012, init_level, 1, 50 * COIN);
//printf("\nregtest with level = %d\n",init_level);
//printf("set is constructed from %d to %d with step 2\n",ldpc_level_table[init_level].from, ldpc_level_table[init_level].to);
//printf("n : %d\t wc : %d\t wr : %d\n", ldpc_level_table[init_level].n, ldpc_level_table[init_level].wc, ldpc_level_table[init_level].wr);
```

2. Get hash values of a genesis block and Merkle-root

We will enter the 'bitcoin core' to get new hash values of a genesis block and Merkle-root.

2.1 Modify hash values to pass assert () function:

Replace the source code for each class with the new code as written inside the box below.

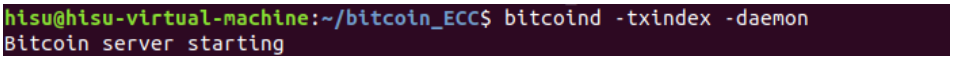
```
consensus.hashGenesisBlock = genesis.GetHash();
assert(consensus.hashGenesisBlock == uint256S("01a92cfc6a9949ae39ae5c58918138414beedad0e5a4f921612bc182bef280d2"));
assert(genesis.hashMerkleRoot == uint256S("0873377ff3078f39a4f1470e7e15d1234c4be9980b41724e714fa953ba04824b"));
```

```
consensus.hashGenesisBlock = genesis.GetHash();
assert(consensus.hashGenesisBlock == genesis.GetHash();
```

```
assert(genesis.hashMerkleRoot ==
BlockMerkleRoot(genesis));
```

Save 'chainparams.cpp' file, compile it and execute the bitcoin core using the following commands.

```
1) $ cd bitcoin_ECC
2) $ ./autogen.sh
3) $ ./configure
4) $ make
5) $ sudo make install
6) $ bitcoind -txindex -daemon
```



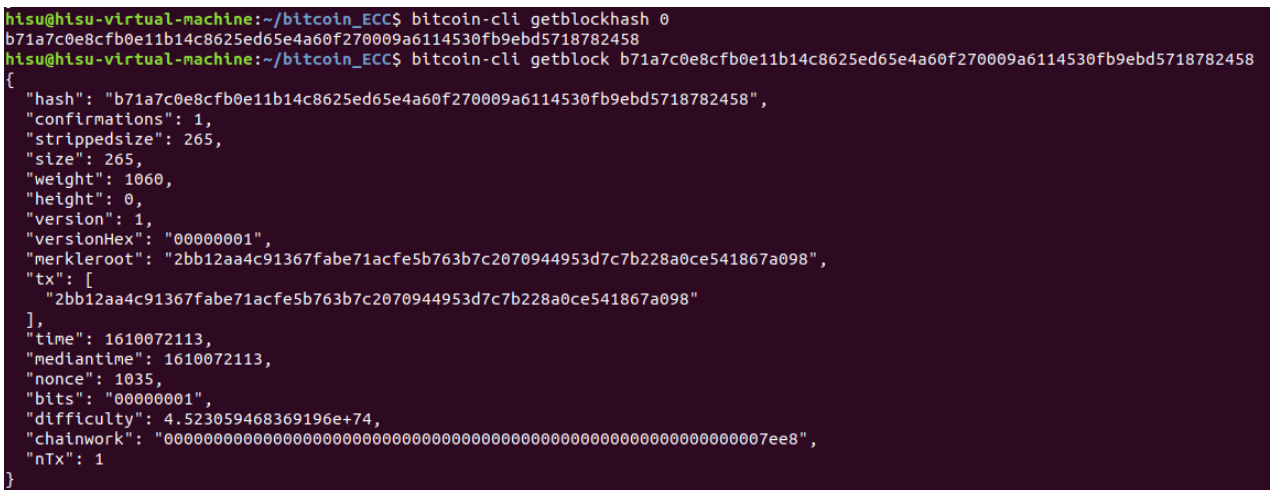
2.2 Get hash values:

Get hash value of the genesis block which block height would be 0.

```
1) $ bitcoin-cli getblockhash 0
```

Use the hash value of the genesis block to get Merkle-root hash value.

```
1) $ bitcoin-cli getblock
b71a7c0e8cfb0e11b14c8625ed65e4a60f270009a6114530fb9edb571
8782458
```



Save both the hash values [genesis block and Merkle-root].

2.3 Replace parts with hash values:

```
consensus.hashGenesisBlock = genesis.GetHash();
assert(consensus.hashGenesisBlock == genesis.GetHash());
assert(genesis.hashMerkleRoot == BlockMerkleRoot(genesis));
```

Replace the source code for each class with the new hash values as written inside the box.

```
consensus.hashGenesisBlock = genesis.GetHash();
assert(consensus.hashGenesisBlock ==
uint256S(‘‘b71a7c0e8cfb0e11b14c8625ed65e4a60f270009a61145
30fb9edb5718782458’’);
assert(genesis.hashMerkleRoot ==
uint256S(‘‘2bb12aa4c91367fabe71acfe5b763b7c2070944953d7c7
b228a0ce541867a098’’));
```

3. Execute blockchain core

3.1 Compile and execute bitcoin server:

Compile the code again using same procedures as mentioned in 2.1

3.2 Check the number of blocks in the blockchain:

First command line returns the number of blocks and Second one returns the information of the blockchain we accessed.

```
1) $ bitcoin-cli getblockcount
2) $ bitcoin-cli getblockchaininfo
```

```
hisu@hisu-virtual-machine:~/bitcoin_ECC$ bitcoind -txindex -daemon
Bitcoin server starting
hisu@hisu-virtual-machine:~/bitcoin_ECC$ bitcoin-cli getblockcount
0
```

```
hisu@hisu-virtual-machine:~/bitcoin_ECC$ bitcoin-cli getblockchaininfo
{
  "chain": "main",
  "blocks": 0,
  "headers": 0,
  "bestblockhash": "b71a7c0e8cfb0e11b14c8625ed65e4a60f270009a6114530fb9ebd5718782458",
  "difficulty": 4.523059468369196e+74,
  "mediantime": 1610072113,
  "verificationprogress": 1,
  "initialblockdownload": false,
  "chainwork": "0000000000000000000000000000000000000000000000000000000007ee8",
  "size_on_disk": 273,
  "pruned": false,
  "softforks": [
    {
      "id": "bip34",
      "version": 2,
      "reject": {
        "status": false
      }
    },
    {
      "id": "bip66"
```

BIT-ECC:
Block Generation Program

Ha-young Park
4th March 2021
INFONET(Prof. Heung-No Lee), GIST

https://github.com/infonetGIST

https://infonet.gist.ac.kr/

Purpose: The aim of this document is to illustrate the block generation algorithms (mining algorithm) for BIT-ECC core, download and run them in local machine.

Contents

1. Prerequisites

1.1 ‘BIT-ECC’ must be installed.
1.2 ‘tmux’ must be installed. ‘tmux’ is a tool for managing pseudoterminals by unit of session and window.

```
$ sudo apt install tmux
```

2. Installation

2.1 Download source code files using the below command but you don’t need to compile the files.

```
$ git clone
https://github.com/cryptoecc/Bitcoin_ECC_BlockGenera
tionProgram
```

3. To run block generation program

3.1 Before starting mining, Check network is synchronized.
3.2 Use this command to check the number of blocks.

```
$ bitcoin-cli getblockcount
```

3.3 When the network is synchronized, Go to ‘Bitcoin_ECC_BlockGenerationProgram’ directory and Run tmux.

3.4 Run ./block_generation_program.
Enter your BIT-ECC public address to get reward coins.

```
(base) gist@gist:~/Bitcoin_ECC_BlockGenerationProgram$ ./block_generation_program
Error: Cannot obtain a lock on data directory /home/gist/.bitcoin. Bitcoin Core is probabl
y already running.
Wait for loading blocks.
Enter the address to get reward coins
3M2F75EruTojXUiaFeUMNHKN8YiAzLxSEt
Start mining.
If you want to stop mining, then enter ctrl + z.
[
  "93e28040c9dfe640d8e06f109157e17e09be1d930b776e4488fe4a4c3a886358"
]
[
  "407f1069c10bc586238e2cc1eb6bd12a16eed26683469cb234c8e02138d31afe"
]
[
  "f939957cc94ac3a2f75bf5b0efe073fd2eb6a689d4321ef5eac042bb602bc622"
]
```

3.5 After that, simply turn off the terminal window.