BLOCKCHAIN IMPLEMENTATION USING RASPBERRY PI

Project Report
Submitted to

By

JAIJITH B C	CHANDANA B	SHAIK JAMEER KHAJA
181040011	181040014	181040001
EWT	EWT	EWT

Under the guidance of
Mr. Mohan Kumar J
School of Information Sciences,
Lower Ground 2, Academic Block 5,
MIT Campus, Manipal

TABLE OF CONTENTS

ABSTRACT

Blockchain is a digital, decentralized (distributed) ledger that keeps a record of all transactions that take place across a peer-to-peer network. It is an interlinked and continuously expanding list of records stored securely across a number of interconnected systems.

Since the birth of the Internet, there have been efforts to create virtual currencies, but those efforts failed due to the 'double spend' problem, namely the risk that a digital asset such as a currency can be spent twice. The current solution to eliminate the double spend problem is through the introduction of 'intermediaries of trust' such as banks. But the application of blockchain technology makes it possible to solve the fundamental problem of double spending without the need for such intermediaries of trust, thereby facilitating the transfer of assets such as virtual currencies over the Internet securely. This concept can be extended to non-currency related areas and that's the promise of blockchain technology.

CHAPTER 1

INTRODUCTION

Blockchain is arguably one of the most significant and disruptive technologies that came into existence since the inception of the Internet. It's the core technology behind Bitcoin and other crypto-currencies that drew a lot of attention in the last few years.

As its core, a blockchain is a distributed database that allows direct transactions between two parties without the need of a central authority. This simple yet powerful concept has great implications for various institutions such as banks, governments and marketplaces, just to name a few. Any business or organization that relies on a centralized database as a core competitive advantage can potentially be disrupted by blockchain technology.

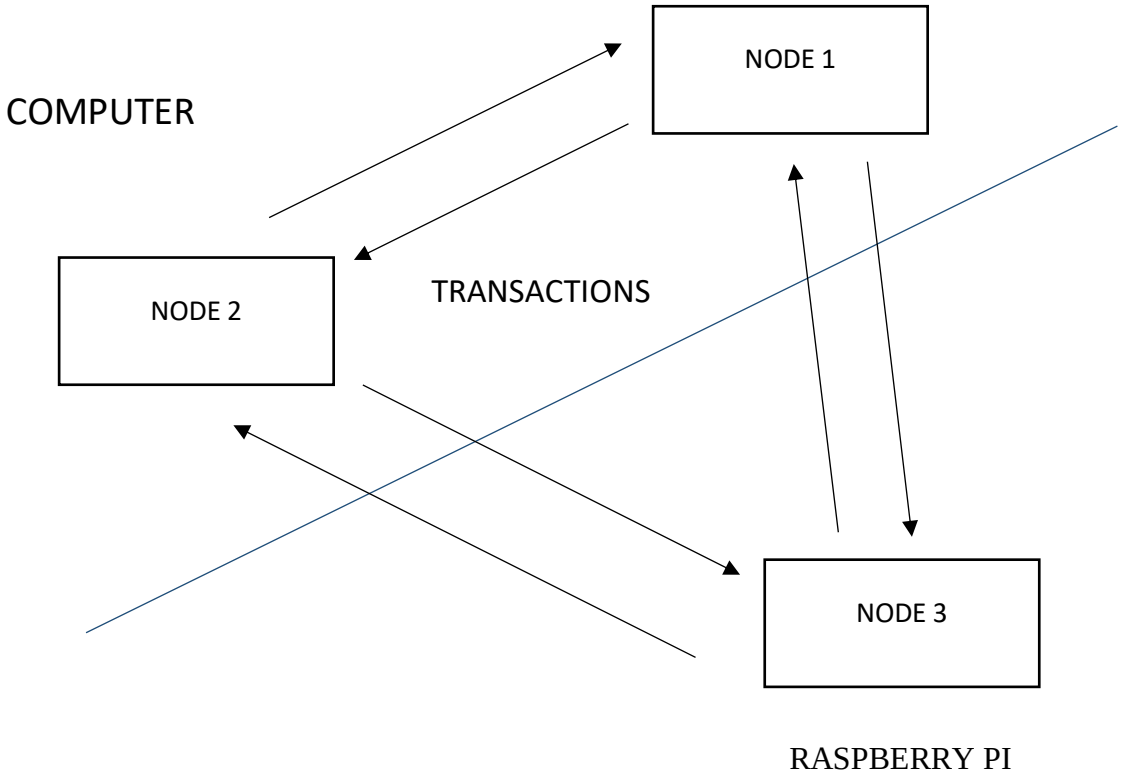


Fig: Block diagram showing transactions between various nodes

Blockchain is a digital, decentralised (distributed) ledger that keeps a record of all transactions that take place across a peer-to-peer network. It is an interlinked and continuously expanding list of records stored securely across a number of interconnected systems [4]. This makes blockchain technology resilient since the network has no single point of vulnerability. Additionally, each 'block' is uniquely connected to the previous blocks via a digital signature which means that making a change to a record without disturbing the previous records in the chain is not possible, thus rendering the information tamper-proof. The key innovation in blockchain technology is that it allows its participant to transfer assets across the Internet without the need for a centralised third party. Blockchain technology was developed as the underlying technology behind the cryptocurrency called bitcoin. The aftermath of the 2008 subprime crisis reduced trust in the existing financial system.

This is when a person or a group of people called Satoshi Nakamoto wrote a white paper containing the 'bitcoin protocol' which used a distributed ledger and consensus building to compute algorithms. The bitcoin protocol was written to disintermediate traditional financial intermediaries as a means of facilitating direct P2P transactions. Since the birth of the Internet, there have been efforts to create virtual currencies, but those efforts failed due to the 'double spend' problem, namely the risk that a digital asset such as a currency can be spent twice. The current solution to eliminate the double spend problem is through the introduction of 'intermediaries of trust' such as banks. But the application of blockchain technology makes it possible to solve the fundamental problem of double spending without the need for such intermediaries of trust, thereby facilitating the transfer of assets such as virtual currencies over the Internet securely. This concept can be extended to non-currency related areas and that's the promise of blockchain technology.

CHAPTER 2

TYPES OF BLOCKCHAIN

Blockchain emerged from the marriage of two concepts:
1. Asymmetrical cryptography, which allows the use of a paired public and private key system.
2. Distributed IT architecture (especially P2P). Asymmetrical cryptography enables users who do not know each other to exchange encrypted information. The system is based on a public key that can be made available to all, and allows encrypted data to be sent to a third party. The third party accesses the encrypted data via a paired private key. The public key is similar to a bank account number, which can be provided to anyone. The private key, which remains secret, acts as the password to the same bank account. A distributed system is a series of independent computers (nodes) that connect to a network and can communicate with each other. It is similar to the Internet, which also has no central node. Downtime for one server does not affect the other users. The blockchain network is a P2P distributed system. Information is shared among the different users.

The blockchain is open-ended and operates in a decentralised, ongoing manner thanks to the activity of its users who can store information, and to consensus algorithms (notably "proof-of-work" and "proof-of-stake") which certify the information per block (unit). Users running these algorithms are known as miners. When a block has been validated, it is added to the blockchain and shared with the network. Blocks are connected to each other in such a way that if users wish to change one block, the entire blockchain must also be changed. On the Bitcoin blockchain, network security is guaranteed by the availability of massive computer power. These two pillars (asymmetrical cryptography and distributed IT architecture) make it possible to create a secure environment that establishes a new basis for trust and allows for new ways of exchanging data, new types of transactions and
new forms of contracts

There mainly three types of Blockchains that have emerged after Bitcoin introduced Blockchain to the world.

1. Public Blockchain
2. Private Blockchain
3. Consortium or Federated Blockchain

There are some more complicated types also such as public-permissioned blockchain, private-permissioned blockchain etc but I will keep it simple for this discussion[2].

Now Let’s discuss all the three one by one.

2.1. Public Blockchain

A public blockchain as its name suggests is the blockchain of the public, meaning a kind of blockchain which is-***'for the people, by the people and of the people'***

Here no one is in charge and anyone can participate in reading/writing/auditing the blockchain. Another thing is that these types of blockchain are open and transparent hence anyone can review anything at a given point of time on a public blockchain.

But a natural question that comes to our mind is that when no one is in charge here then how the decisions are taken on these types of the blockchain. So the answer is that decision making happens by various decentralized consensus mechanisms such as proof of work (POW) and proof of stake (POS) etc.

- **Example:** Bitcoin, Litecoin etc

On Bitcoin and Litecoin blockchain networks anyone can do the following things that make it truly public blockchain.

—>Anyone can run BTC/LTC full node and start mining.

—>Anyone can make transactions on BTC/LTC chain.

—>Anyone can review/audit the blockchain in a Blockchain explorer.

2.2. Private Blockchain

Private blockchain as its name suggests is a private property of an individual or an organization.

Unlike public blockchain here there is an in charge who looks after of important things such as read/write or whom to selectively give access to read or vice versa.

Here the consensus is achieved on the whims of the central in-charge who can give mining rights to anyone or not give at all.

That's what makes it centralized again where various rights are exercised and vested in a central trusted party but yet it is cryptographical secured from the company's point of view and more cost-effective for them.

But it is still debatable if such a private thing can be called a 'Blockchain' because it fundamentally defeats the whole purpose of blockchain that Bitcoin introduced to us.

- **Example:** Bankchain

In such types of blockchain:

—>Anyone can't run a full node and start mining.

—>Anyone can't make transactions on the chain.

—>Anyone can't review/audit the blockchain in a Blockchain explorer.

2.3. Consortium or Federated Blockchain

This type of blockchain tries to remove the sole autonomy which gets vested in just one entity by using private blockchains.

So here instead of one in charge, you have more than one in charge. Basically, you have a group of companies or representative individuals coming together and making decisions for the best benefit of the whole network. Such groups are also called consortiums or a federation that's why the name consortium or federated blockchain.

For example, let suppose you have a consortium of world's top 20 financial institutes out which you have decided in the code that if a transaction or a block or decision is voted/verified by more than 15 institutes then only it should get added to the blockchain.

So it is a way of achieving thing much faster and you also have more than one single point of failures which in a way protects the whole ecosystem against a single point of failure.

- **Example:** r3, **EWF**

In such type blockchain:

—>Members of the consortium can run a full node and start mining.

—>Members of the consortium can make transactions/decisions on the chain.

—>Members of the consortium can review/audit the blockchain in a Blockchain explorer.

Why We Need Them

Public Blockchain	Private Blockchain	Consortium or Federated Blockchain
Anyone can run BTC/LTC full node	Anyone can't run a full node	Selected members of the consortium can run a full node
Anyone can make transactions	Anyone can't make transactions	Selected members of the consortium can make transactions
Anyone can review/audit the blockchain	Anyone can't review/audit the blockchain	Selected members of the consortium can review/audit the blockchain

We require more types of blockchain because keeping such blockchains solves problems such as:-

1. One no longer need to rely upon huge servers.
2. They are cost effective and fast.
3. They reduce the need for more trusted parties because you can implement smart contracts instead of them.
4. Gives options for rights and access management while leveraging the same blockchain technology and reaping its benefits.
5. Reduces redundant work.
6. Distributed consensus between interested parties becomes fast even though you are geographically segregated.

Because of all these, I think different types of blockchain will be used for different type of industries as and when required.

Where we require privacy and control, private & consortium blockchain will be a good option and where we require openness, as well as censorship resistance public blockchains, are a must need.

And that's why different people are discussing different use cases of the blockchain tech across various industry verticals.

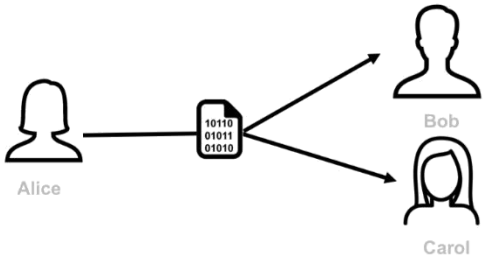
Nevertheless, time will only tell how far each type of these blockchains go and who will be the winner.

CHAPTER 3

DOUBLE SPENDING

3.1. What is Double-Spending?

Suppose that Alice wants to pay Bob 1$. If Alice and Bob use physical cash, then Alice will no longer have the 1$ after the transaction is executed. If Alice and Bob use digital money, then the problem gets more complicated. Digital money is in digital form and can be easily duplicated. If Alice sends a digital file worth 1$ to Bob by email for example, Bob cannot know for sure if Alice has deleted her copy of the file. If Alice still has the 1$ digital file, then she can choose to send the same file to Carol. This problem is called double-spending.



Double-Spending Problem: If Alice sends money in digital format to Bob, Bob cannot know for sure if Alice has deleted her copy of the file and she can choose to send the same file to Carol.

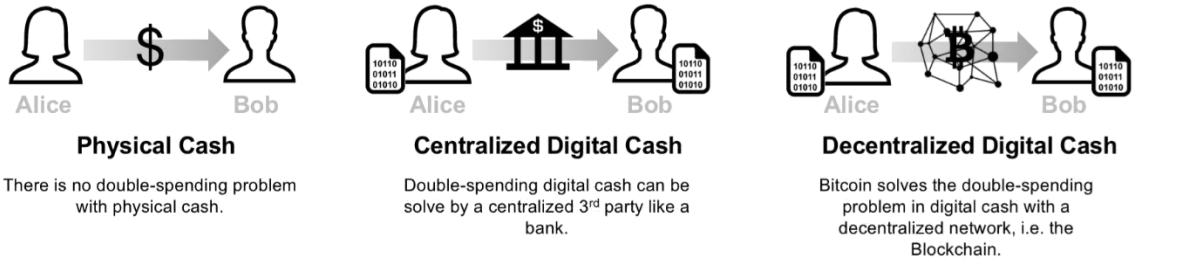
One way of solving the double-spending problem is to have a trusted third party (a bank for example) between Alice, Bob and all other participants in the network. This third party is responsible for managing a centralized ledger that keeps track of and validates all the transactions in the network. The drawback of this solution is that for the system to function, it requires trust in a centralized third party.

3.2. Bitcoin: A Decentralized Solution for the Double-Spending Problem

To solve the double-spending problem, Satoshi proposed a public ledger, i.e., Bitcoin's blockchain to keep track of all transactions in the network. Bitcoin's blockchain has the following characteristics:

- Distributed: The ledger is replicated across a number of computers, rather than being stored on a central server. Any computer with an internet connection can download a full copy of the blockchain.

- Cryptographic: Cryptography is used to make sure that the sender owns the bitcoin that she's trying to send, and to decide how the transactions are added to the blockchain.
- Immutable: The blockchain can be changed in append only fashion. In other words, transactions can only be added to the blockchain but cannot be deleted or modified.
- Uses Proof of Work (PoW): A special type of participants in the network called miners compete on searching for the solution to a cryptographic puzzle that will allow them to add a block of transactions to Bitcoin's blockchain. This process is called Proof of Work and it allows the system to be secure (more on this later).



Sending bitcoin money goes as follows:

- Step 1 (one-time effort): Create a bitcoin wallet. For a person to send or receive bitcoins, she needs to create a bitcoin wallet. A bitcoin wallet stores 2 pieces of information: A private key and a public key. The private key is a secret number that allows the owner to send bitcoin to another user, or spend bitcoins on services that accept them as payment method. The public key is a number that is needed to receive bitcoins. The public key is also referred to as bitcoin address (not entirely true, but for simplicity we will assume that the public key and the bitcoin address are the same). Note that the wallet doesn't store the bitcoins themselves. Information about bitcoins balances are stored on the Bitcoin's blockchain.
- Step 2: Create a bitcoin transaction. If Alice wants to send 1 BTC to Bob, Alice needs to connect to her bitcoin wallet using her private key, and create a transaction that contains the amount of bitcoins she wants to send and the address where she wants to send them (in this case Bob's public address).
- Step 3: Broadcast the transaction to Bitcoin's network. Once Alice creates the bitcoin transaction, she needs to broadcast this transaction to the entire Bitcoin's network.
- Step 4: Confirm the transaction. A miner listening to Bitcoin's network authenticates the transaction using Alice's public key, confirms that Alice has enough bitcoins in her wallet (in this case at least 1 BTC), and adds a new record to Bitcoin's Blockchain containing the details of the transaction.
- Step 5: Broadcast the blockchain change to all miners. Once the transaction is confirmed, the miner should broadcast the blockchain change to all miners to make sure that their copies of the blockchain are all in sync.

CHAPTER 4

A TECHNICAL DEEP DIVE ON BLOCKCHAIN

The goal of this section is to go deeper into the technical building blocks that power the blockchain. We will cover public key cryptography, hashing functions, mining and security of the blockchain.

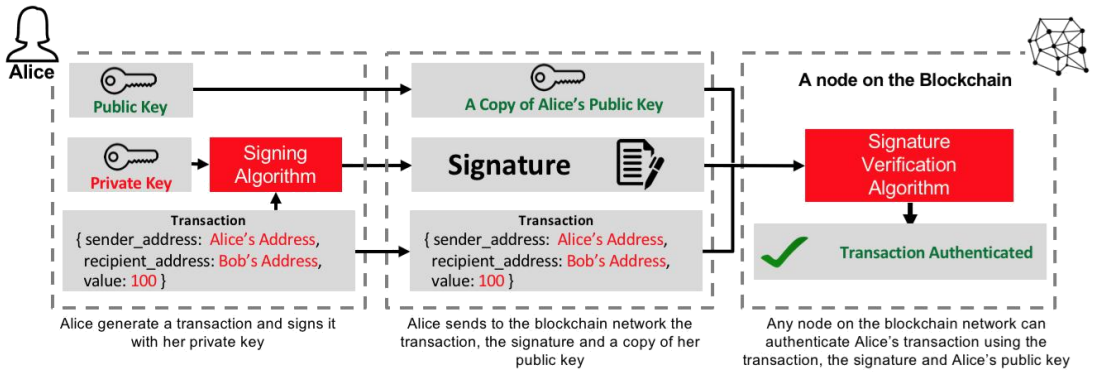
4.1. Public Key Cryptography

Public-key cryptography, or asymmetrical cryptography, is any cryptographic system that uses pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner. This accomplishes two functions: authentication, where the public key verifies a holder of the paired private key sent the message, and encryption, where only the paired private key holder can decrypt the message encrypted with the public key. [1]

RSA and Elliptic Curve Digital Signature (ECDSA) are the most popular public-key cryptography algorithms.

In the case of Bitcoin, ECDSA algorithm is used to generate Bitcoin wallets. Bitcoin uses a variety of keys and addresses, but for the sake of simplicity, we will assume in this blog post that each Bitcoin wallet has one pair of private/public keys and that a Bitcoin address is the wallet's public key.

To send or receive BTCs, a user starts by generating a wallet which contains a pair of private and public keys. If Alice wants to send Bob some BTCs, she creates a transaction in which she enters both her and Bob's public keys, and the amount of BTCs she wants to send. She then sign the transaction using her private key. A computer on the blockchain uses Alice's public key to verify that the transaction is authentic and adds the transaction to a block that will be later added to the blockchain.

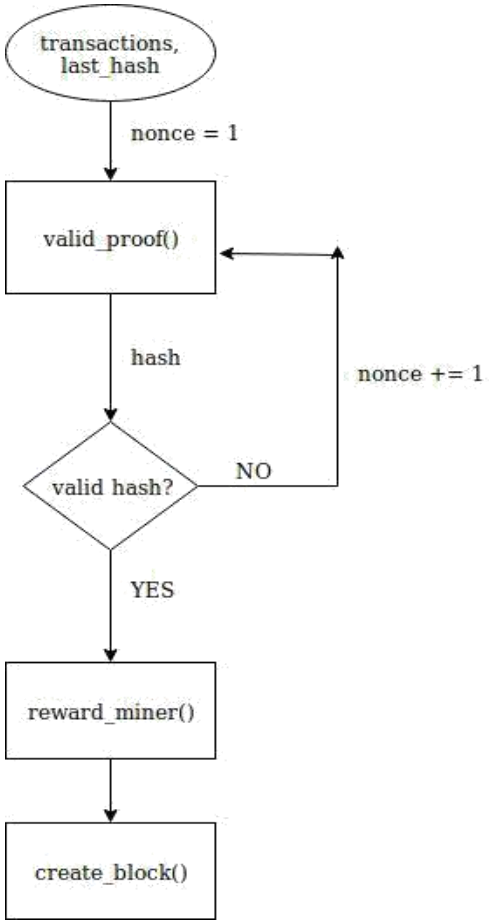


Authentication Process for Transactions on the Blockchain

4.2. Hashing Functions and Mining

All Bitcoin transactions are grouped in files called blocks. Bitcoin adds a new block of transactions every 10 minutes. Once a new block is added to the blockchain, it becomes immutable and can't be deleted or modified. A special group of participants in the network called miners (computers connected to the blockchain) are responsible for creating new blocks of transactions. A miner has to authenticate each transaction using the sender's public key, confirm that the sender has enough balance for the requested transaction, and add the transaction to the block. Miners are completely free to choose which transactions to include in the blocks, therefore the senders need to include a transaction fee to incentivise the miners to add their transactions to the blocks.

For a block to be accepted by the blockchain, it needs to be "mined". To mine a block, miners need to find an extremely rare solution to a cryptographic puzzle. If a mined block is accepted by the blockchain, the miner receives a reward in bitcoins which is an additional incentive to transaction fees. The mining process is also referred to as Proof of Work (PoW), and it's the main mechanism that enables the blockchain to be trustless and secure (more on blockchain security later).



The mining Process

4.2.1 Hashing and Blockchain's Cryptographic Puzzle

To understand the blockchain's cryptographic puzzle, we need to start with hash functions. A hash function is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hashes. Hash functions are usually used to accelerate database lookup by detecting duplicated records, and they are also widely used in cryptography. A cryptographic hash function allows one to easily verify that some input data maps to a given hash value, but if the input data is unknown, it is deliberately difficult to reconstruct it by knowing the stored hash value. [2]

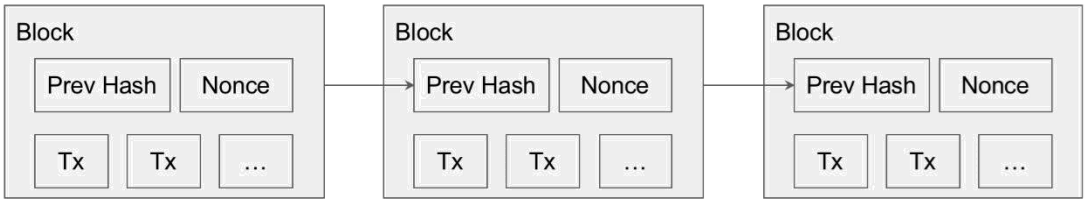
Bitcoins uses a cryptographic hash function called SHA-256. SHA-256 is applied to a combination of the block's data (bitcoin transactions) and a number called nonce. By changing the block data or the nonce, we get completely different hashes. For a block to be considered valid or "mined", the hash value of the block and the nonce needs to meet a certain condition. For example, the four leading digits of the hash needs to be equal to "0000". We can increase the mining complexity by making the condition more complex, for example we can increase the number of 0s that the hash value needs to start with.

The cryptographic puzzle that miners need to solve is to find a nonce value that makes the hash value satisfies the mining condition. You can use the app below to simulate block mining. When you type in the "Data" text box or change the nonce value, you can notice the change in the hash value. When you click the "Mine" button, the app starts with a nonce equals to zero, computes the hash value and checks if the leading four digits of the hash value is equal to "0000". If the leading four digits are not equal to "0000", it increments the nonce by one and repeats the whole process until it finds a nonce value that satisify the condition. If the block is considered mined, the background color turns green.

4.3. From Blocks to Blockchain

As discussed in the previous section, transactions are grouped in blocks and blocks are appended to the blockchain. In order to create a chain of blocks, each new block uses the previous block’s hash as part of its data. To create a new block, a miner selects a set of transactions, adds the previous block’s hash and mines the block in a similar fashion described above.

Any changes to the data in any block will affect all the hash values of the blocks that come after it and they will become invalid. This give the blockchain its immutability characteristic.

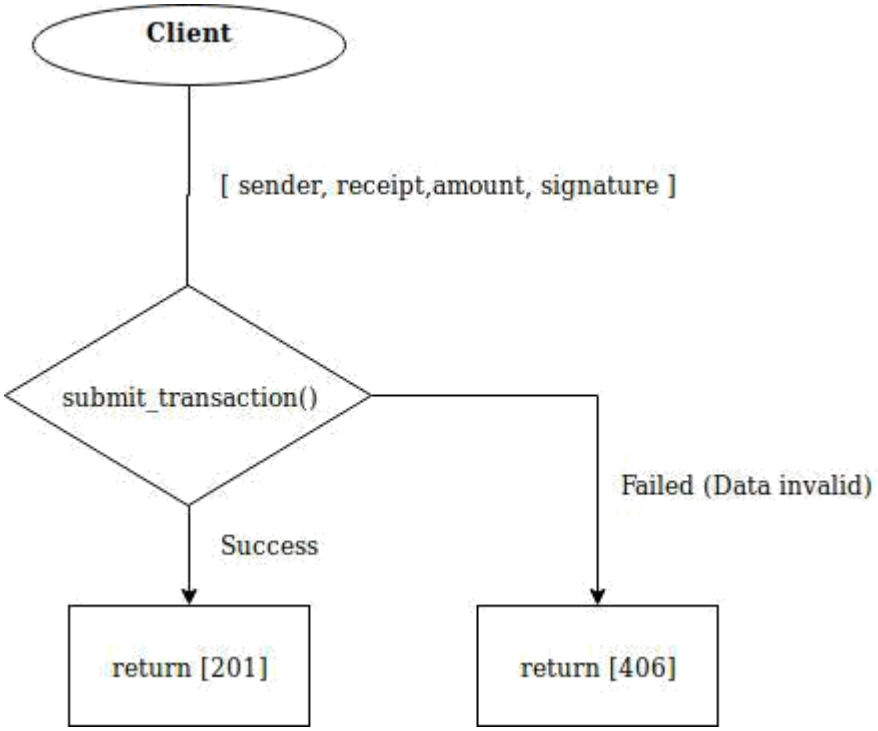


Blocks are chained together using the previous block's hash to form a Blockchain.

You can use the app below to simulate a blockchain with 3 blocks. When you type in the "Data" text box or change the nonce value, you can notice the change in the hash value and the "Prev" value (previous hash) of the next block. You can simulate the mining process by clicking on the "Mine" button of each individual block. After mining the 3 blocks, try changing the data in block 1 or 2, and you will notice that all the blocks that come after become invalid.

4.4. Adding Blocks to the Blockchain

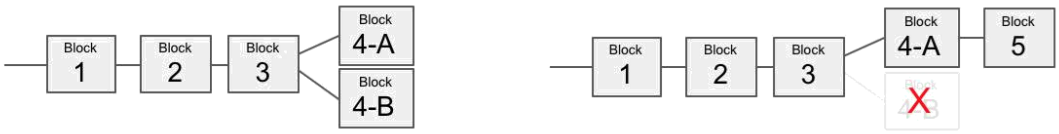
All the miners in the Bitcoin network compete with each other to find a valid block that will be added to the blockchain and get the reward from the network. Finding a nonce that validated a block is rare, but because of the number of miners, the probability of a miner in the network validating a block is extremely high. The first miner to submit a valid block gets his block added to the blockchain and receives the reward in bitcoins. But what happens if two miners or more submit their blocks at the same time?



Adding a new transaction

4.4.1 Resolving Conflicts

If 2 miners solve a block at almost the same time, then we will have 2 different blockchains in the network, and we need to wait for the next block to resolve the conflict. Some miners will decide to mine on top of blockchain 1 and others on top of blockchain 2. The first miner to find a new block resolves the conflict. If the new block was mined on top of blockchain 1, then blockchain 2 becomes invalid, the reward of the previous block goes to the miner from blockchain 1 and the transactions that were part of blockchain 2 and weren't added to the blockchain go back to the transactions pool and get added to the next blocks. In short, if there is a conflict on the blockchain, then the longest chain wins.



Resolving conflicts - The longest chain wins

4.5. Blockchain and Double-Spending

In this section, we will cover the most popular ways for performing double-spending attacks on the blockchain, and the measures that users should take to prevent damages from them.

4.5.1 Race Attack

An attacker sends the same coin in rapid succession to two different addresses. To prevent from this attack, it is recommended to wait for at least one block confirmation before accepting the payment.

4.5.2 Finney Attack

An attacker pre-mines a block with a transaction, and spends the same coins in a second transaction before releasing the block. In this scenario, the second transaction will not be validated. To prevent from this attack, it is recommended to wait for at least 6 block confirmations before accepting the payment.

4.5.3 Majority Attack (also called 51% attack)

In this attack, the attacker owns 51% of the computing power of the network. The attacker starts by making a transaction that is brodcasted to the entire network, and then mines a private blockchain where he double-spends the coins of the previous transaction. Since the attacker owns the majority of the computing power, he is guaranteed that he will have at some point a longer

chain than the "honest" network. He can then release his longer blockchain that will replace the "honest" blockchain and cancel the original transaction. This attack is highly unlikely, as it’s very expensive in blockchain networks like Bitcoin.

CHAPTER 5

IMPLEMENTATION USING RASPBERRY PI

5.1 ETHEREUM

Ethereum is an open-source, public, blockchain-based distributed computing platform and operating system featuring smart contract (scripting) functionality. It supports a modified version of Nakamoto consensus via transaction-based state transitions.

Ether is a cryptocurrency whose blockchain is generated by the Ethereum platform. Ether can be transferred between accounts and used to compensate participant mining nodes for computations performed. Ethereum provides a decentralized virtual machine, the Ethereum Virtual Machine (EVM), which can execute scripts using an international network of public nodes.

The Ethereum platform has its own cryptocurrency, called ether, but it also builds further on blockchain technology to create a decentralised platform for smart contracts — objects which contain code functions and that live on the blockchain, and are able to interact with other contracts, make decisions, store data, and send ether to others.

Smart contracts are implemented in a language called Solidity, which is based on JavaScript. The Solidity compiler is used to compile smart contracts to bytecode — just as is done with JavaScript or e.g. Python, Java and Android etc. code prior to execution — which is then executed via the Ethereum Virtual Machine (EVM). There is a cost associated with executing the transactions in a smart contract and this is something that we'll take a look at in a future post.

There are a number of different client applications available for Ethereum, with the original reference implementation, geth, written in Go. Some of these can mine ether and there is standalone mining software also. Plus GUI clients and an IDE for distributed applications.

In addition to the primary, public Ethereum blockchain network, mainnet, there are also test networks for experimentation, and you can create your own private networks, too.

5.2 Steps involved in creating a private Ethereum blockchain with IoT devices

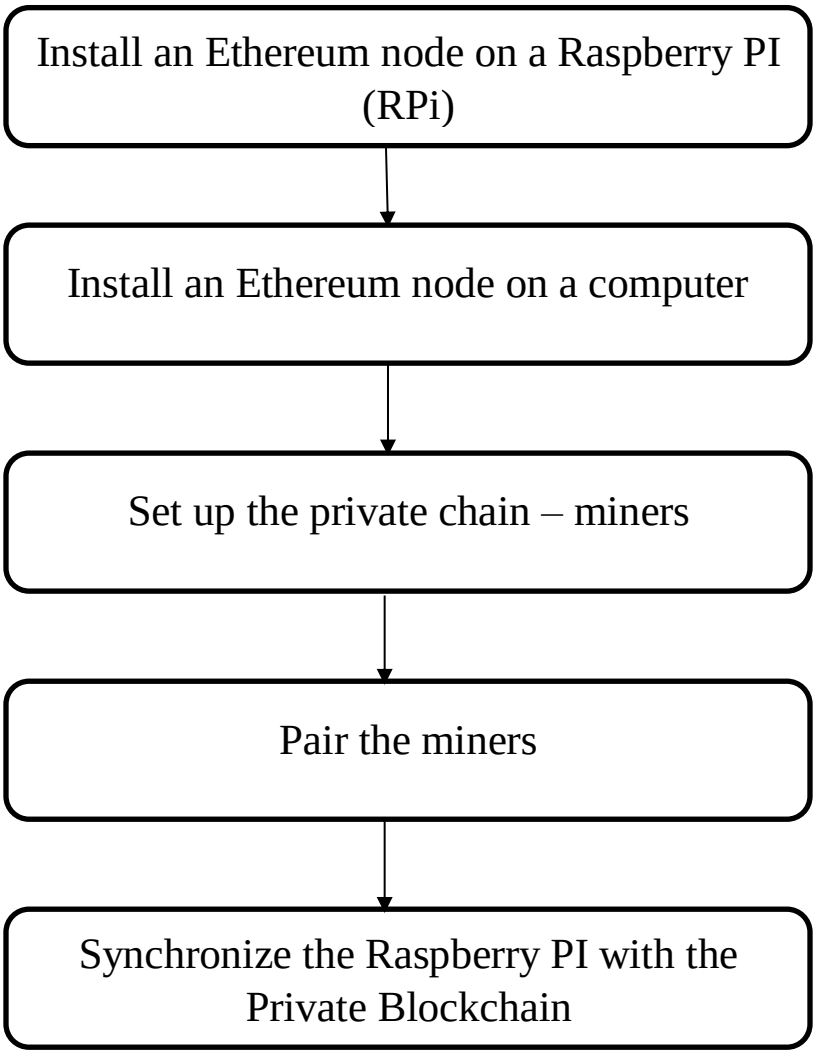


Fig: Flowchart of steps involved in creating a private Ethereum blockchain with IoT devices

Part 1: Install an Ethereum node on a Raspberry PI (RPi)

1. Check your configuration
2. Format your SD Card
3. Write the Raspbian image
4. Configure your RPi

   Log in the RPi with the default credential is:

   - username: **pi**
   - password: **raspberry**

   The RPi needs to be configured:

   - Change the **keyboard layout** (if required)
   - Set the **timezone**: required to allow a blockchain synchronisation between nodes
   - Enable **SSH**: securely access your RPi from your computer
   - Change your **hostname**: easily identified your RPi within your network (example: node1, node2, etc.)
   - We use **raspi-config** to setup all these settings.
   - Press Finish and reboot your RPi.
   - Finally, you should really change the default password of the pi account:

5. Setup Wi-Fi

   To setup your Wi-Fi interface, proceed as described below:

```
pi$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

   - Change the country code to set yours
   - Add the following lines at the end of the file and replace the ssid with the name of your WiFi and psk with your WiFi's password:

```
network={

 ssid="enter the SSID of your Wi-Fi network"

 psk="enter the password of your Wi-Fi network"

}
```

To apply your changes, restart your RPi or your Wi-Fi interface:

```
pi$ sudo ifdown wlan0

pi$ sudo ifup wlan0
```

You can now unplug your LAN cable.

6. Connect to your RPI

To connect remotely to your RPi through SSH, you have to know the IP address of your RPi device.You can find it by typing the following command from your RPi:

```
pi$ ifconfig
```

Or you can search it from your LAN by filtering IP adresses from a MAC prefix address (*b8:27:eb* is the MAC prefix address of RPi devices):

```
computer$ arp -na | grep -i b8:27:eb

? (192.168.1.31) at b8:27:eb:1c:aa:94 on en0 ifscope [ethernet]
```

If you have several RPi, you can locate them using the "**host**" command:

```
computer$ host 192.168.1.103

103.1.168.192.in-addr.arpa domain name pointer ethpinode1.
```

At this stage, you can continue the setup of your RPI remotely from your computer using the "**ssh**" ("**pi**" is the username defined on your RPi):

```
computer$ ssh pi@192.168.1.103
```

7. Install Geth

In this step, we are going to install the Go implementation of Ethereum client called **Geth**.

Connect to your RPi and retrieve the type of your CPU:

```
pi@ethpinode1:~ $ cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 4 (v7l)
BogoMIPS        : 38.40
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 1
model name      : ARMv7 Processor rev 4 (v7l)
BogoMIPS        : 38.40
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 2
model name      : ARMv7 Processor rev 4 (v7l)
BogoMIPS        : 38.40
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32
CPU implementer : 0x41
CPU architecture: 7
```

- Go to the download page of Go-Ethereum .
- Select the tab **Linux** from the stable releases page.
- Locate the row related to your CPU model and **copy the link address** . It's recommended to retrieve the latest version of Geth.
- Perform the following commands:

```
pi@ethpinode1:~ $ wget https://gethstore.blob.core.windows.net/builds/geth-linux-arm7-1.5.7-da2a22c3.tar.gz
--2019-03-21 21:18:25--  https://gethstore.blob.core.windows.net/builds/geth-linux-arm7-1.5.7-da2a22c3.tar.gz
Resolving gethstore.blob.core.windows.net (gethstore.blob.core.windows.net)... 40.113.27.176
Connecting to gethstore.blob.core.windows.net (gethstore.blob.core.windows.net)|40.113.27.176|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7813278 (7.5M) [application/octet-stream]
Saving to: 'geth-linux-arm7-1.5.7-da2a22c3.tar.gz'

geth-linux-arm7-1.5.7-da2a22c3.tar. 100%[=============================================================>]   7.45M  3.54MB/s    in 2.1s

2019-03-21 21:18:28 (3.54 MB/s) - 'geth-linux-arm7-1.5.7-da2a22c3.tar.gz' saved [7813278/7813278]

pi@ethpinode1:~ $ tar zxvf geth-linux-arm7-1.5.7-da2a22c3.tar.gz
geth-linux-arm7-1.5.7-da2a22c3/
geth-linux-arm7-1.5.7-da2a22c3/COPYING
geth-linux-arm7-1.5.7-da2a22c3/geth
pi@ethpinode1:~ $ cd geth-linux-arm7-1.5.7-da2a22c3
pi@ethpinode1:~/geth-linux-arm7-1.5.7-da2a22c3 $ sudo cp geth /usr/local/bin
pi@ethpinode1:~/geth-linux-arm7-1.5.7-da2a22c3 $ geth version
Geth
Version: 1.5.7-stable
Git Commit: da2a22c384a9b621ec853fe4b1aa651d606cf42b
Protocol Versions: [63 62]
Network Id: 1
Go Version: go1.7.4
OS: linux
GOPATH=
GOROOT=/usr/local/go
pi@ethpinode1:~/geth-linux-arm7-1.5.7-da2a22c3 $
```

- Now, Ethereum is installed.

8. Run Geth

```
pi@ethpinode1:~/geth-linux-arm7-1.5.7-da2a22c3 $ geth
I0321 21:20:29.524176 cmd/utils/flags.go:607] WARNING: No etherbase set and no accounts found as default
I0321 21:20:29.525014 ethdb/database.go:83] Allotted 128MB cache and 1024 file handles to /home/pi/.ethereum/geth/chaindata
I0321 21:20:29.594189 ethdb/database.go:176] closed db:/home/pi/.ethereum/geth/chaindata
I0321 21:20:29.597208 node/node.go:176] instance: Geth/v1.5.7-stable-da2a22c3/linux/go1.7.4
I0321 21:20:29.597319 ethdb/database.go:83] Allotted 128MB cache and 1024 file handles to /home/pi/.ethereum/geth/chaindata
I0321 21:20:29.669209 eth/backend.go:191] Protocol Versions: [63 62], Network Id: 1
I0321 21:20:33.096356 eth/backend.go:209] WARNING: Wrote default ethereum genesis block
I0321 21:20:33.096914 eth/backend.go:219] Chain config: {ChainID: 1 Homestead: 1150000 DAO: 1920000 DAOSupport: true EIP150: 2463000 EIP155: 26
75000 EIP158: 2675000}
I0321 21:20:33.099055 core/blockchain.go:217] Last header: #0 [d4e56740…] TD=17179869184
I0321 21:20:33.099151 core/blockchain.go:218] Last block: #0 [d4e56740…] TD=17179869184
I0321 21:20:33.099209 core/blockchain.go:219] Fast block: #0 [d4e56740…] TD=17179869184
I0321 21:20:33.102976 p2p/server.go:340] Starting Server
I0321 21:20:35.954021 p2p/discover/udp.go:227] Listening, enode://bfdbd85d40d9192727545886fbec3d33600dd66fd516e6028883aae25e29eb0ecc04e081b9bea
adcef7334f61de58797c4e4e32ccdecfcb943aad27f0ef7bed2@[::]:30303
I0321 21:20:35.955773 p2p/server.go:608] Listening on [::]:30303
I0321 21:20:35.970634 node/node.go:341] IPC endpoint opened: /home/pi/.ethereum/geth.ipc
```

Press CTRL-C to stop the Ethereum server.

Part 2: Install an Ethereum node on a computer

1. Homebrew

Homebrew is a package manager that you should install on your computer:

```
computer$ brew --version
```

If Homebrew is not installed on your computer, install it using the following command:

```
computer$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2. Install Geth

In this step, we install the Go implementation of Ethereum protocol (Geth).

Geth can be installed with the following commands:

```
~ — pi@ethpinode1: ~/geth-linux-arm7-1.8.23-c9427004 — ssh pi@192.168.1.103   ~ — -bash
jaijiths-MacBook-Air:~ jaijithbc$ brew update
Already up-to-date.
jaijiths-MacBook-Air:~ jaijithbc$ brew tap ethereum/ethereum
==> Tapping ethereum/ethereum
Cloning into '/usr/local/Homebrew/Library/Taps/ethereum/homebrew-ethereum'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
Unpacking objects: 100% (8/8), done.
remote: Total 8 (delta 1), reused 3 (delta 0), pack-reused 0
Tapped 3 formulae (36 files, 36.0KB).
jaijiths-MacBook-Air:~ jaijithbc$ brew install ethereum
==> Downloading https://homebrew.bintray.com/bottles/ethereum-1.8.22.mojave.bottle.tar.gz
######################################################################## 100.0%
==> Pouring ethereum-1.8.22.mojave.bottle.tar.gz
🍺  /usr/local/Cellar/ethereum/1.8.22: 21 files, 286MB
==> `brew cleanup` has not been run in 30 days, running now...
```

Check the version of Geth using:

```
jaijiths-MacBook-Air:~ jaijithbc$ geth version
Geth
Version: 1.8.22-stable
Architecture: amd64
Protocol Versions: [63 62]
Network Id: 1
Go Version: go1.11.5
Operating System: darwin
GOPATH=
GOROOT=/usr/local/Cellar/go/1.11.5/libexec
jaijiths-MacBook-Air:~ jaijithbc$
```

Ensure that the version of Geth is the same as the one installed on your RPi. If not, upgrade your nodes.

3. Run Geth

Start Geth using the following command:

```
jaijiths-MacBook-Air:~ jaijithbc$ geth
INFO [03-21|21:55:07.290] Maximum peer count                       ETH=25 LES=0 total=25
INFO [03-21|21:55:07.307] Starting peer-to-peer node               instance=Geth/v1.8.22-stable/darwin-amd64/go1.11.5
INFO [03-21|21:55:07.308] Allocated cache and file handles         database=/Users/jaijithbc/Library/Ethereum/geth/chaindata cache=512 handles=
461168601842738790 3
INFO [03-21|21:55:07.315] Writing default main-net genesis block
INFO [03-21|21:55:07.688] Persisted trie from memory database      nodes=12356 size=1.88mB time=81.791531ms gcnodes=0 gcsize=0.00B gctime=0s li
venodes=1 livesize=0.00B
INFO [03-21|21:55:07.689] Initialised chain configuration          config="{ChainID: 1 Homestead: 1150000 DAO: 1920000 DAOSupport: true EIP150:
 2463000 EIP155: 2675000 EIP158: 2675000 Byzantium: 4370000 Constantinople: 7280000  ConstantinopleFix: 7280000 Engine: ethash}"
INFO [03-21|21:55:07.689] Disk storage enabled for ethash caches   dir=/Users/jaijithbc/Library/Ethereum/geth/ethash count=3
INFO [03-21|21:55:07.689] Disk storage enabled for ethash DAGs     dir=/Users/jaijithbc/.ethash                      count=2
INFO [03-21|21:55:07.689] Initialising Ethereum protocol           versions="[63 62]" network=1
INFO [03-21|21:55:07.708] Loaded most recent local header          number=0 hash=d4e567…cb8fa3 td=17179869184 age=49y11mo6d
INFO [03-21|21:55:07.708] Loaded most recent local full block      number=0 hash=d4e567…cb8fa3 td=17179869184 age=49y11mo6d
INFO [03-21|21:55:07.708] Loaded most recent local fast block      number=0 hash=d4e567…cb8fa3 td=17179869184 age=49y11mo6d
INFO [03-21|21:55:07.709] Regenerated local transaction journal    transactions=0 accounts=0
INFO [03-21|21:55:07.719] New local node record                    seq=1 id=e29fbfab938c538b ip=127.0.0.1 udp=30303 tcp=30303
INFO [03-21|21:55:07.719] Started P2P networking                   self=enode://c653d6e24fbc97680c35f6f59557bbf3adaf7effde8dc9b10684886893cde81
d0e488e6bce58f07da739fb1ed22dab7fbfe6c8eb54ffa9e08db2faad3addfc33@127.0.0.1:30303
INFO [03-21|21:55:07.723] IPC endpoint opened                      url=/Users/jaijithbc/Library/Ethereum/geth.ipc
```

When you start Geth like that, without any command line argument, it immediately starts synchronizing with the main chain, which takes several days.

Press CTRL-C to stop the Ethereum server.

Part 3: Set up the private chain – miners

1. Create the datadir folder

When running a private blockchain, it is highly recommended to use a specific folder to store the data (database and wallet) of the private blockchain without impacting the folders used to store the data coming from the public blockchain.

From your computer, create the folder that will host your first miner:

Repeat the operation for the second miner:

```
Last login: Thu Mar 21 22:49:00 on ttys000
jaijiths-MacBook-Air:~ jaijithbc$ CD ChainSkills/
jaijiths-MacBook-Air:~ jaijithbc$ cd ChainSkills/
jaijiths-MacBook-Air:ChainSkills jaijithbc$ ls
genesis.json.save       miner1                  miner2
```

2. Create the Genesis file

Each blockchain starts with a genesis block that is used to initialize the blockchain and defines the terms and conditions to join the network.

Our genesis block is called "**genesis.json"** and is stored under "**~/ChainSkills**" folder.
Create a text file under ~/ChainSkills, called genesis.json, with the following content:

```
{
 "nonce": "0x0000000000000042",
 "mixhash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
 "difficulty": "0x400",
 "alloc": {},
 "coinbase": "0x0000000000000000000000000000000000000000",
 "timestamp": "0x00",
 "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
 "extraData": "0x436861696e536b696c6c732047656e6573697320426c6f636b",
 "gasLimit": "0xffffffff",
 "config": {
   "chainId": 42,
   "homesteadBlock": 0,
   "eip155Block": 0,
   "eip158Block": 0
 }
}
```

Among the parameters, we have the following ones:

- difficulty: if the value is low, the transactions will be quickly processed within our private blockchain.
- gasLimit: define the limit of Gas expenditure per block. The gasLimit is set to the maximum to avoid being limited to our tests.

3. Initialize the private block chain

It's time to initialize the private blockchain with the genesis block.

This operation will create the initial database stored under the data directory dedicated to each miner.

3.1 Initialize miner #1

Type the following command to create the blockchain for the first miner:

```
jaijiths-MacBook-Air:ChainSkills jaijithbc$ geth --datadir ~/ChainSkills/miner1 init genesis.json
INFO [03-21|23:12:21.699] Maximum peer count                       ETH=25 LES=0 total=25
INFO [03-21|23:12:21.719] Allocated cache and file handles         database=/Users/jaijithbc/ChainSkills/miner1/geth/chaindata cache=16 handles
=16
INFO [03-21|23:12:21.745] Writing custom genesis block
INFO [03-21|23:12:21.746] Persisted trie from memory database      nodes=0 size=0.00B time=31.775µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=
1 livesize=0.00B
INFO [03-21|23:12:21.747] Successfully wrote genesis state         database=chaindata                                          hash=c0990b…cdd1
ab
INFO [03-21|23:12:21.747] Allocated cache and file handles         database=/Users/jaijithbc/ChainSkills/miner1/geth/lightchaindata cache=16 ha
ndles=16
INFO [03-21|23:12:21.755] Writing custom genesis block
INFO [03-21|23:12:21.755] Persisted trie from memory database      nodes=0 size=0.00B time=6.063µs  gcnodes=0 gcsize=0.00B gctime=0s livenodes=
1 livesize=0.00B
INFO [03-21|23:12:21.756] Successfully wrote genesis state         database=lightchaindata                                       hash=c0990b
…cdd1ab
jaijiths-MacBook-Air:ChainSkills jaijithbc$ geth --datadir ~/ChainSkills/miner2 init genesis.json
INFO [03-21|23:12:57.576] Maximum peer count                       ETH=25 LES=0 total=25
INFO [03-21|23:12:57.586] Allocated cache and file handles         database=/Users/jaijithbc/ChainSkills/miner2/geth/chaindata cache=16 handles
=16
INFO [03-21|23:12:57.593] Writing custom genesis block
INFO [03-21|23:12:57.593] Persisted trie from memory database      nodes=0 size=0.00B time=16.39µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1
 livesize=0.00B
INFO [03-21|23:12:57.594] Successfully wrote genesis state         database=chaindata                                          hash=c0990b…cdd1
ab
INFO [03-21|23:12:57.594] Allocated cache and file handles         database=/Users/jaijithbc/ChainSkills/miner2/geth/lightchaindata cache=16 ha
ndles=16
INFO [03-21|23:12:57.601] Writing custom genesis block
INFO [03-21|23:12:57.601] Persisted trie from memory database      nodes=0 size=0.00B time=3.761µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1
 livesize=0.00B
INFO [03-21|23:12:57.601] Successfully wrote genesis state         database=lightchaindata                                       hash=c0990b
…cdd1ab
jaijiths-MacBook-Air:ChainSkills jaijithbc$
```

3.2 Initialize miner #2

Repeat the same operation to initialize the second miner by specifying its own target folder (~/ChainSkills/miner2)

4. Create accounts

4.2 Accounts for miner #1

Create the default account that will be used to run the node.

This account will receive all ethers created by the miner in the private blockchain. These ethers will serve to test our solutions by paying the gas required to process each transaction.

To create the default account for the miner #1, type the following command. Keep the password in a safe place:

```
jaijiths-MacBook-Air:ChainSkills jaijithbc$ geth --datadir ~/ChainSkills/miner1 account new
INFO [03-21|23:13:54.898] Maximum peer count                       ETH=25 LES=0 total=25
Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase:
Repeat passphrase:
Address: {fbf7050338d2636b0a6351451b8f007e6ccaa77d}
```

Add an additional account for testing purpose:

```
jaijiths-MacBook-Air:ChainSkills jaijithbc$ geth --datadir ~/ChainSkills/miner1 account new
INFO [03-21|23:14:40.104] Maximum peer count                       ETH=25 LES=0 total=25
Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase:
Repeat passphrase:
Address: {abee35d13a1b64857c376ec1a58a37bcfd7a225a}
```

The wallet for these accounts is located right here:

```
jaijiths-MacBook-Air:ChainSkills jaijithbc$ ls -al ~/ChainSkills/miner1/keystore
total 16
drwx------  4 jaijithbc  staff  128 Mar 21 23:14 .
drwxr-xr-x  4 jaijithbc  staff  128 Mar 21 23:12 ..
-rw-------  1 jaijithbc  staff  491 Mar 21 23:14 UTC--2019-03-21T17-44-04.836936000Z--fbf7050338d2636b0a6351451b8f007e6ccaa77d
-rw-------  1 jaijithbc  staff  491 Mar 21 23:14 UTC--2019-03-21T17-44-47.254389000Z--abee35d13a1b64857c376ec1a58a37bcfd7a225a
```

To list all accounts of your node, use the following command:

```
jaijiths-MacBook-Air:ChainSkills jaijithbc$ geth --datadir ~/ChainSkills/miner1 account list
INFO [03-21|23:15:38.313] Maximum peer count                       ETH=25 LES=0 total=25
Account #0: {fbf7050338d2636b0a6351451b8f007e6ccaa77d} keystore:///Users/jaijithbc/ChainSkills/miner1/keystore/UTC--2019-03-21T17-44-04.836936000Z--fbf7050338d2636b0a6351451b8f007e6ccaa77d
Account #1: {abee35d13a1b64857c376ec1a58a37bcfd7a225a} keystore:///Users/jaijithbc/ChainSkills/miner1/keystore/UTC--2019-03-21T17-44-47.254389000Z--abee35d13a1b64857c376ec1a58a37bcfd7a225a
```

4.3 Accounts for miner #2

Repeat the same operation to create the default account for the second miner. The difference lies in the target folder (~/ChainSkills/miner2).

```
jaijiths-MacBook-Air:ChainSkills jaijithbc$ geth --datadir ~/ChainSkills/miner2 account new
INFO [03-21|23:16:35.136] Maximum peer count                       ETH=25 LES=0 total=25
Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase:
Repeat passphrase:
Address: {d1ba35b34de39a0465b7987b4a247e3aa99809b5}
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$ geth --datadir ~/ChainSkills/miner2 account new
INFO [03-21|23:16:59.957] Maximum peer count                       ETH=25 LES=0 total=25
Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase:
Repeat passphrase:
Address: {0a54a20d2a7fd4f568a0512482c63faf3e6db587}
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$ geth --datadir ~/ChainSkills/miner2 account list
INFO [03-21|23:17:26.280] Maximum peer count                       ETH=25 LES=0 total=25
Account #0: {d1ba35b34de39a0465b7987b4a247e3aa99809b5} keystore:///Users/jaijithbc/ChainSkills/miner2/keystore/UTC--2019-03-21T17-46-42.413968000Z--d1ba35b34de39a0465b7987b4a247e3aa99809b5
Account #1: {0a54a20d2a7fd4f568a0512482c63faf3e6db587} keystore:///Users/jaijithbc/ChainSkills/miner2/keystore/UTC--2019-03-21T17-47-06.488786000Z--0a54a20d2a7fd4f568a0512482c63faf3e6db587
jaijiths-MacBook-Air:ChainSkills jaijithbc$
```

5 Prepare the miners

5.1 Miner #1: setup

Let's start by creating a file that will contain the password for our default account, which is the first account we created for each miner. Create a password.sec file under ~/ChainSkills/miner1/ that contains the password you configured for the first account on miner1, in plain text.

To start the miner #1, we will require to run the following command:

```
computer$ geth --identity "miner1" --networkid 42 --datadir "~/ChainSkills/miner1"
--nodiscover --mine --rpc --rpcport "8042" --port "30303" --unlock 0 --password
~/ChainSkills/miner1/password.sec --ipcpath "~/Library/Ethereum/geth.ipc"
```

The meaning of the main parameters is the following:

- identity: name of our node
- networkid: this network identifier is an arbitrary value that will be used to pair all nodes of the same network. This value must be different from 0 to 3 (already used by the live chains)
- datadir: folder where our private blockchain stores its data
- rpc and rpcport: enabling HTTP-RPC server and giving its listening port number
- port: network listening port number, on which nodes connect to one another to spread new transactions and blocks
- nodiscover: disable the discovery mechanism (we will pair our nodes later)
- mine: mine ethers and transactions
- unlock: id of the default account
- password: path to the file containing the password of the default account
- ipcpath: path where to store the filename for IPC socket/pipe

```
jaijiths-MacBook-Air:miner1 jaijithbc$ ls
geth            keystore        password.sec
jaijiths-MacBook-Air:miner1 jaijithbc$ geth --identity "miner1" --networkid 42 --datadir "~/ChainSkills/miner1" --nodiscover --mine --rpc --rpc
port "8042" --port "30303" --unlock 0 --password ~/ChainSkills/miner1/password.sec --ipcpath "~/Library/Ethereum/geth.ipc"
INFO [03-21|23:26:51.153] Maximum peer count                       ETH=25 LES=0 total=25
INFO [03-21|23:26:51.165] Starting peer-to-peer node               instance=Geth/miner1/v1.8.22-stable/darwin-amd64/go1.11.5
INFO [03-21|23:26:51.165] Allocated cache and file handles         database=/Users/jaijithbc/ChainSkills/miner1/geth/chaindata cache=512 handle
s=4611686018427387903
INFO [03-21|23:26:51.192] Initialised chain configuration          config="{ChainID: 42 Homestead: 0 DAO: <nil> DAOSupport: false EIP150: <nil>
 EIP155: 0 EIP158: 0 Byzantium: <nil> Constantinople: <nil>  ConstantinopleFix: <nil> Engine: unknown}"
INFO [03-21|23:26:51.192] Disk storage enabled for ethash caches   dir=/Users/jaijithbc/ChainSkills/miner1/geth/ethash count=3
INFO [03-21|23:26:51.192] Disk storage enabled for ethash DAGs     dir=/Users/jaijithbc/.ethash                        count=2
INFO [03-21|23:26:51.193] Initialising Ethereum protocol           versions="[63 62]" network=42
INFO [03-21|23:26:51.254] Loaded most recent local header          number=0 hash=c0990b…cdd1ab td=1024 age=49y11mo6d
INFO [03-21|23:26:51.254] Loaded most recent local full block      number=0 hash=c0990b…cdd1ab td=1024 age=49y11mo6d
INFO [03-21|23:26:51.254] Loaded most recent local fast block      number=0 hash=c0990b…cdd1ab td=1024 age=49y11mo6d
INFO [03-21|23:26:51.258] Regenerated local transaction journal    transactions=0 accounts=0
INFO [03-21|23:26:51.284] New local node record                    seq=1 id=a7f19c8408f4ade8 ip=127.0.0.1 udp=0 tcp=30303
INFO [03-21|23:26:51.284] Started P2P networking                   self="enode://bf2a47a9cba6757d885f4414411092bd81c7dbcd914b3bf3ce43606070f919
ec9ff0a88067129064333dd0e3078a382dabb8a7ecd8922b7e0b1d4822f3d0432f@127.0.0.1:30303?discport=0"
INFO [03-21|23:26:51.292] IPC endpoint opened                      url=/Users/jaijithbc/Library/Ethereum/geth.ipc
INFO [03-21|23:26:51.295] HTTP endpoint opened                     url=http://127.0.0.1:8042                       cors= vhosts=localhost
WARN [03-21|23:26:51.295] ------------------------------------------------------------------
WARN [03-21|23:26:51.295] Referring to accounts by order in the keystore folder is dangerous!
WARN [03-21|23:26:51.295] This functionality is deprecated and will be removed in the future!
WARN [03-21|23:26:51.295] Please use explicit addresses! (can search via `geth account list`)
WARN [03-21|23:26:51.295] ------------------------------------------------------------------
INFO [03-21|23:26:52.295] Unlocked account                         address=0xfbf7050338D2636b0a6351451b8f007e6cCAA77D
INFO [03-21|23:26:52.295] Transaction pool price threshold updated price=1000000000
INFO [03-21|23:26:52.295] Updated mining threads                   threads=0
INFO [03-21|23:26:52.295] Transaction pool price threshold updated price=1000000000
INFO [03-21|23:26:52.296] Etherbase automatically configured       address=0xfbf7050338D2636b0a6351451b8f007e6cCAA77D
INFO [03-21|23:26:52.296] Commit new mining work                   number=1 sealhash=5899f0…314802 uncles=0 txs=0 gas=0 fees=0 elapsed=198.042µ
s
```

We recommend that you store the Geth command into a runnable script. In our example, this script is called "**startminer1.sh**" and is located here: ~/ChainSkills/miner1

```
#!/bin/bash



geth --identity "miner1" --networkid 42 --datadir "~/ChainSkills/miner1" --
nodiscover --mine --rpc --rpcport "8042" --port "30303" --unlock 0 --password
~/ChainSkills/miner1/password.sec --ipcpath "~/Library/Ethereum/geth.ipc"
```

5.2 Miner #1: start

Make the script runnable:

```
jaijiths-MacBook-Air:miner1 jaijithbc$ cd ~/ChainSkills/miner1
jaijiths-MacBook-Air:miner1 jaijithbc$ chmod +x startminer1.sh
jaijiths-MacBook-Air:miner1 jaijithbc$
jaijiths-MacBook-Air:miner1 jaijithbc$
jaijiths-MacBook-Air:miner1 jaijithbc$ ./startminer1.sh
INFO [03-21|23:30:41.276] Maximum peer count                       ETH=25 LES=0 total=25
INFO [03-21|23:30:41.286] Starting peer-to-peer node               instance=Geth/miner1/v1.8.22-stable/darwin-amd64/go1.11.5
INFO [03-21|23:30:41.286] Allocated cache and file handles         database=/Users/jaijithbc/ChainSkills/miner1/geth/chaindata cache=512 handle
s=4611686018427387903
INFO [03-21|23:30:41.304] Initialised chain configuration          config="{ChainID: 42 Homestead: 0 DAO: <nil> DAOSupport: false EIP150: <nil>
 EIP155: 0 EIP158: 0 Byzantium: <nil> Constantinople: <nil>  ConstantinopleFix: <nil> Engine: unknown}"
INFO [03-21|23:30:41.304] Disk storage enabled for ethash caches   dir=/Users/jaijithbc/ChainSkills/miner1/geth/ethash count=3
INFO [03-21|23:30:41.304] Disk storage enabled for ethash DAGs     dir=/Users/jaijithbc/.ethash                        count=2
INFO [03-21|23:30:41.304] Initialising Ethereum protocol           versions="[63 62]" network=42
INFO [03-21|23:30:41.363] Loaded most recent local header          number=0 hash=c0990b…cdd1ab td=1024 age=49y11mo6d
INFO [03-21|23:30:41.363] Loaded most recent local full block      number=0 hash=c0990b…cdd1ab td=1024 age=49y11mo6d
INFO [03-21|23:30:41.363] Loaded most recent local fast block      number=0 hash=c0990b…cdd1ab td=1024 age=49y11mo6d
INFO [03-21|23:30:41.363] Loaded local transaction journal         transactions=0 dropped=0
INFO [03-21|23:30:41.363] Regenerated local transaction journal    transactions=0 accounts=0
INFO [03-21|23:30:41.378] New local node record                    seq=2 id=a7f19c8408f4ade8 ip=127.0.0.1 udp=0 tcp=30303
INFO [03-21|23:30:41.378] Started P2P networking                   self="enode://bf2a47a9cba6757d885f4414411092bd81c7dbcd914b3bf3ce43606070f919
ec9ff0a88067129064333dd0e3078a382dabb8a7ecd8922b7e0b1d4822f3d0432f0@127.0.0.1:30303?discport=0"
INFO [03-21|23:30:41.379] IPC endpoint opened                      url=/Users/jaijithbc/Library/Ethereum/geth.ipc
INFO [03-21|23:30:41.379] HTTP endpoint opened                     url=http://127.0.0.1:8042                       cors= vhosts=localhost
WARN [03-21|23:30:41.380] ------------------------------------------------------------------
WARN [03-21|23:30:41.380] Referring to accounts by order in the keystore folder is dangerous!
WARN [03-21|23:30:41.380] This functionality is deprecated and will be removed in the future!
WARN [03-21|23:30:41.380] Please use explicit addresses! (can search via `geth account list`)
WARN [03-21|23:30:41.380] ------------------------------------------------------------------
INFO [03-21|23:30:42.379] Unlocked account                         address=0xfbf7050338D2636b0a6351451b8f007e6cCAA77D
INFO [03-21|23:30:42.379] Transaction pool price threshold updated price=1000000000
INFO [03-21|23:30:42.379] Updated mining threads                   threads=0
INFO [03-21|23:30:42.380] Transaction pool price threshold updated price=1000000000
INFO [03-21|23:30:42.380] Etherbase automatically configured       address=0xfbf7050338D2636b0a6351451b8f007e6cCAA77D
INFO [03-21|23:30:42.380] Commit new mining work                   number=1 sealhash=cbf779…7e85d9 uncles=0 txs=0 gas=0 fees=0 elapsed=173.247µ
s
```

You will notice that the server and the mining process start.
You default account will receive ethers mined by the node.

5.3 Miner #1: JavaScript console

You can manage your miner using the Geth Javascript console.

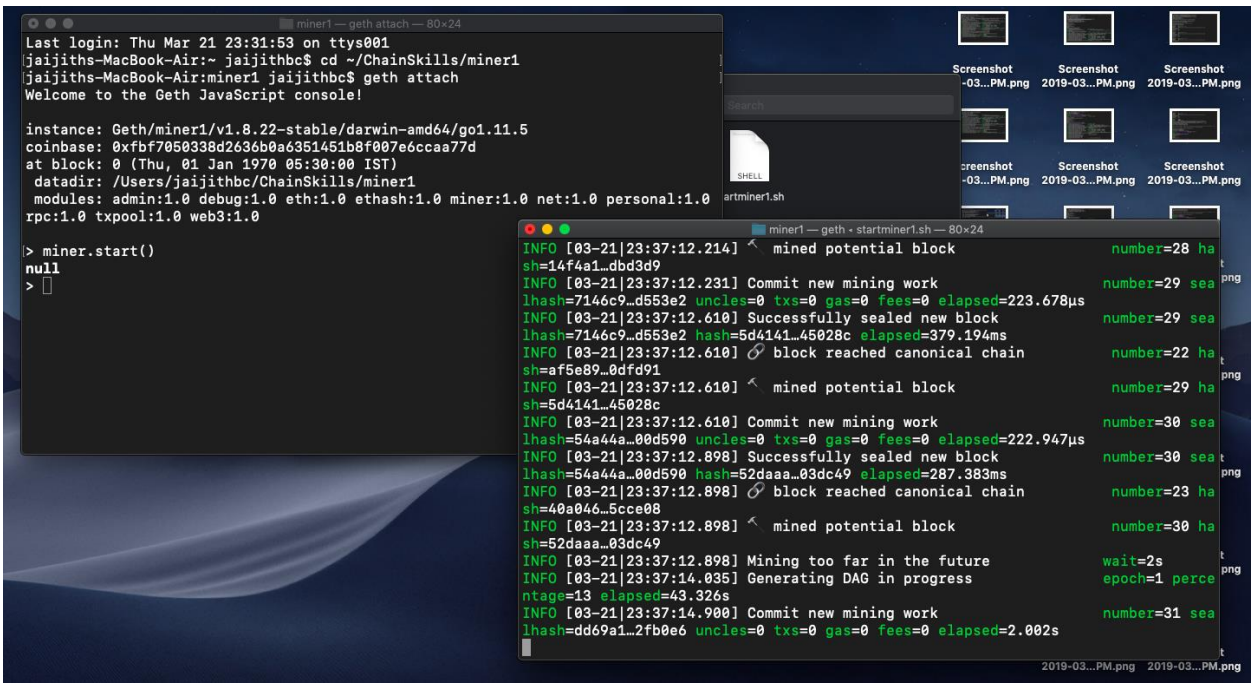
For example, you can start and stop mining from the console or send transactions.

This console needs to be attached to a running instance of Geth.

Open a new terminal session and type "**geth attach**".

If you want to start or stop the mining process, proceed as below:

```
computer$ geth attach

...

> miner.start()

...

> miner.stop()
...
```



5.4 Miner #1: stop

To stop your miner, go to its console window and press CTRL-C.

Another option is to kill the Geth running process as illustrated here below:

```
computer$ ps aux | grep geth
```

```
eloudsa 43872 0.0 2.2 556717632 363492 s001 S+ 3:49PM 1:58.01 geth --identity
miner1 --dev



computer$ kill -INT 43872
```

Similarly, perform same operations in case of 5.5, 5.6 and 5.7.

5.5 Miner #2: setup

5.6 Miner #2: JavaScript console

5.7 Miner #2: stop

6 Send ethers within miner #1

To ensure that our mining node is properly installed, let's try to send some ethers between accounts created on each miner.

6.1 Start the miner and its console

Let's begin to start the miner #1:

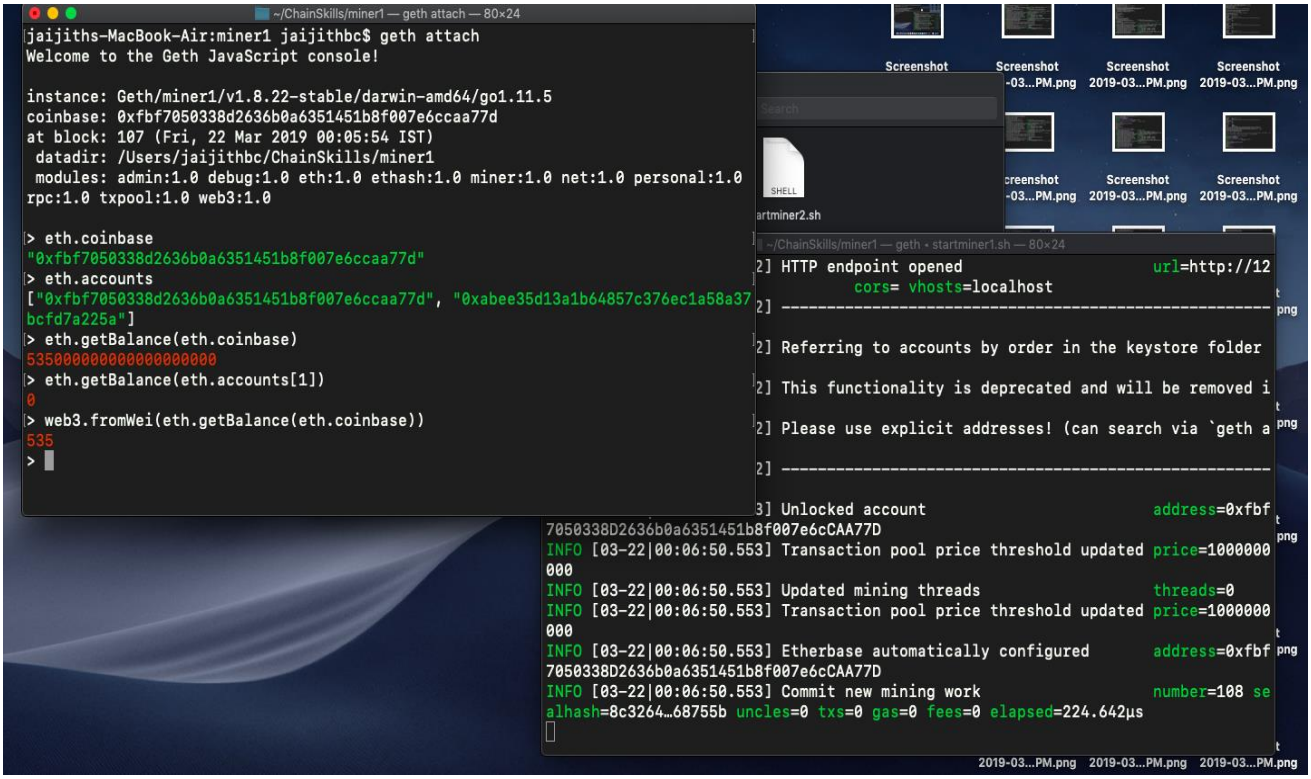
```
computer$ cd ~/ChainSkills/miner1

computer$ ./startminer1.sh
...
```

Open a second terminal and start the Geth console:

```
computer$ geth attach
...
>
```

6.2 Check balances

- Get the default account with "eth.coinbase":
- List the accounts with "eth.accounts":
- Check balances with "eth.getBalance(<account id>)":
- The default account has plenty of ethers obtained during the mining process. We will use them to test our solutions.
- By default, the balance is expressed in Wei that is the base unit of ether.



6.3 Stop mining

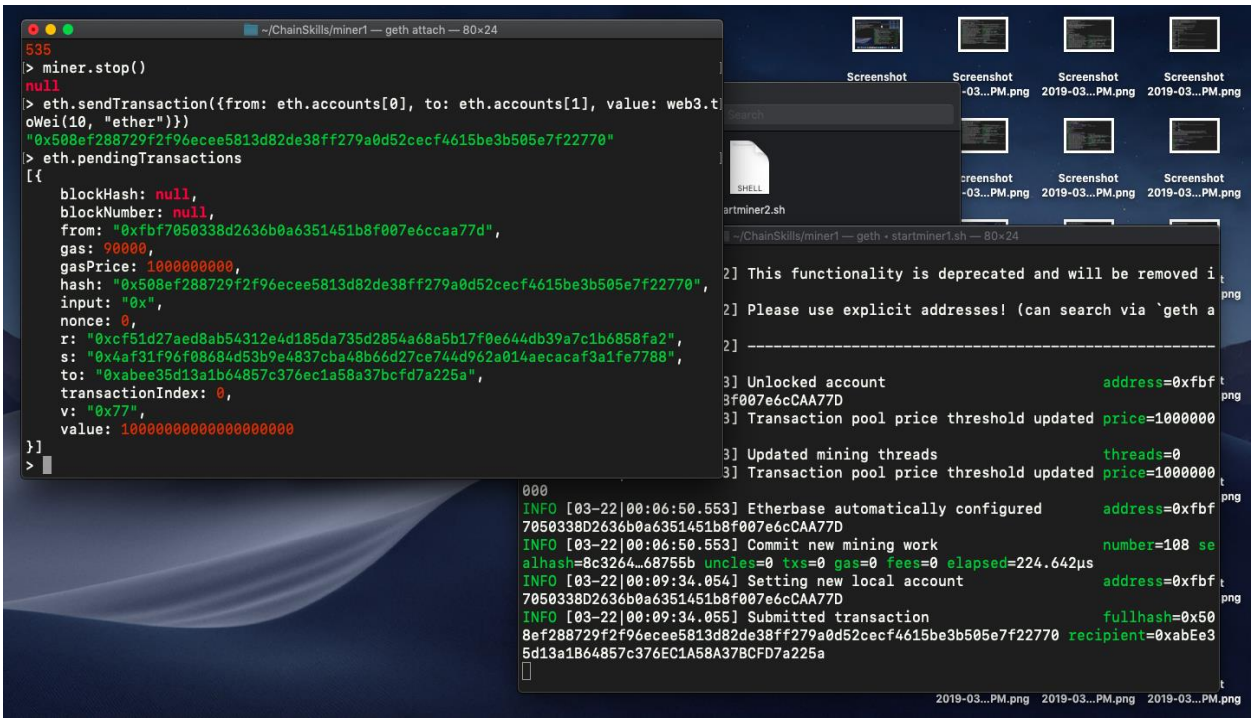
We stop the mining process to let us review the content of a transaction:

6.4 Send ethers

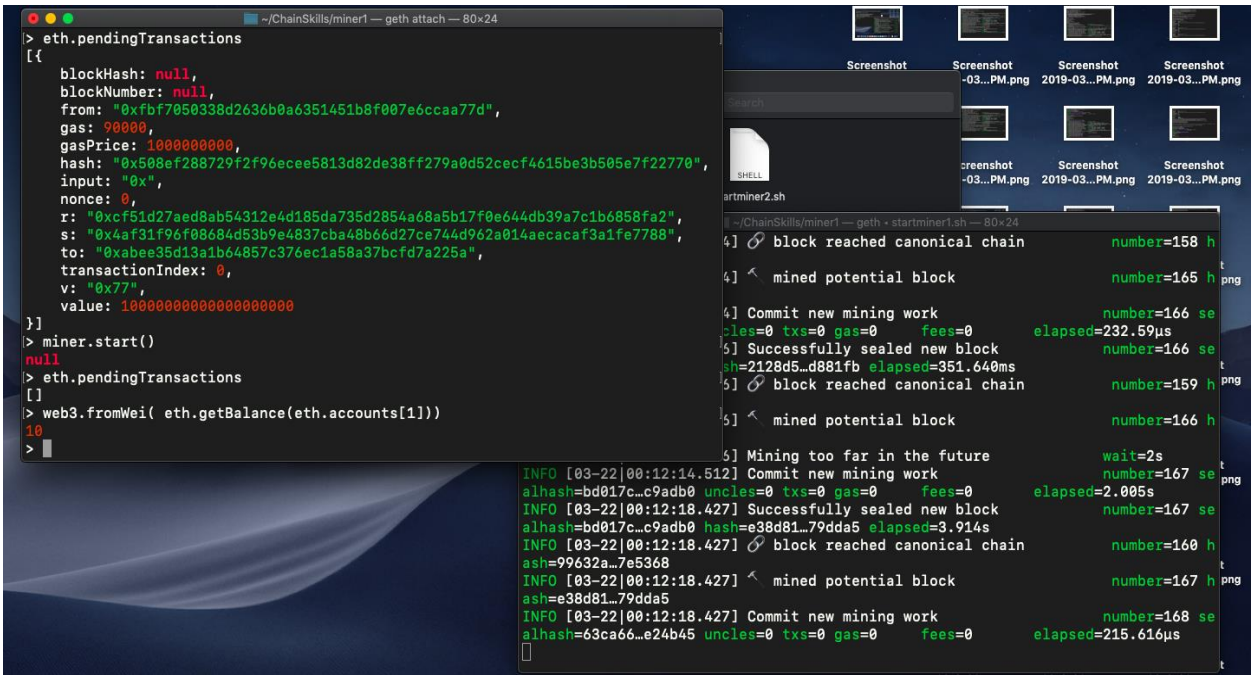
- Through the Geth console, we send 10 ethers from account #0 to account #1:
- The command returns a transaction hash.
- As the mining process has been stopped, we can review the list of pending transactions:

We can easily identify the following information:

- The sender address
- The recipient address
- The value to transfer (expressed in Wei)
- The hash of the transaction
- The cost of the transaction (gas x gasPrice) in Wei

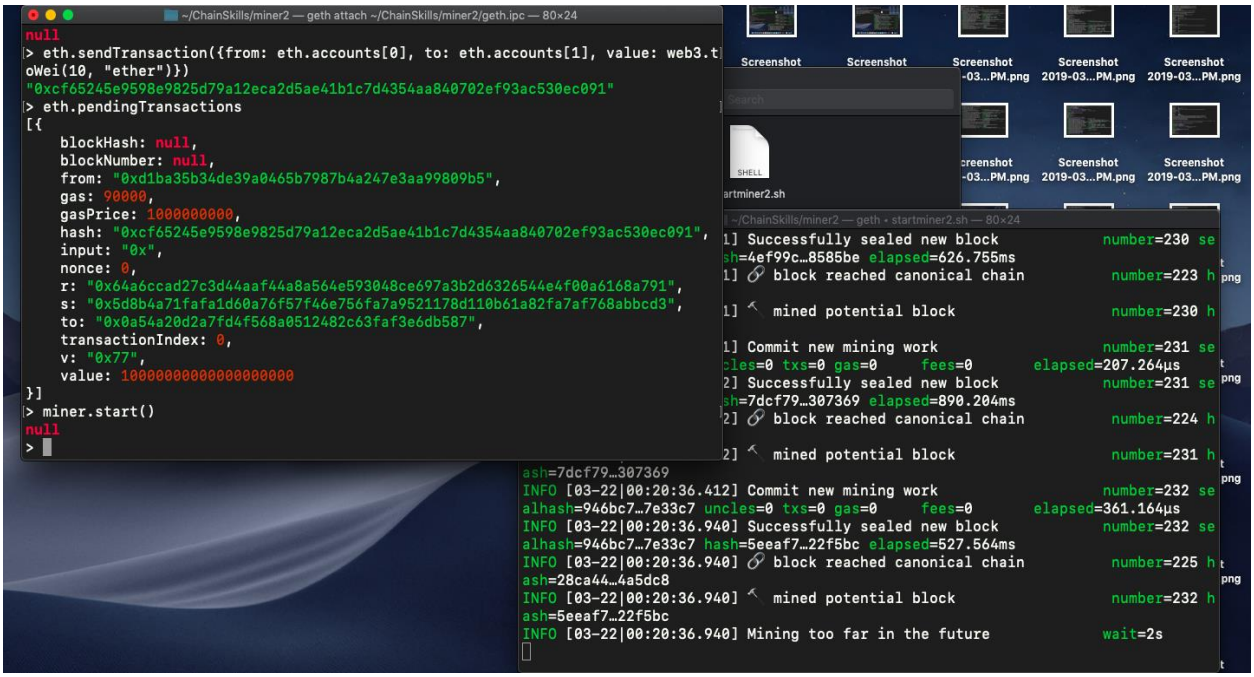


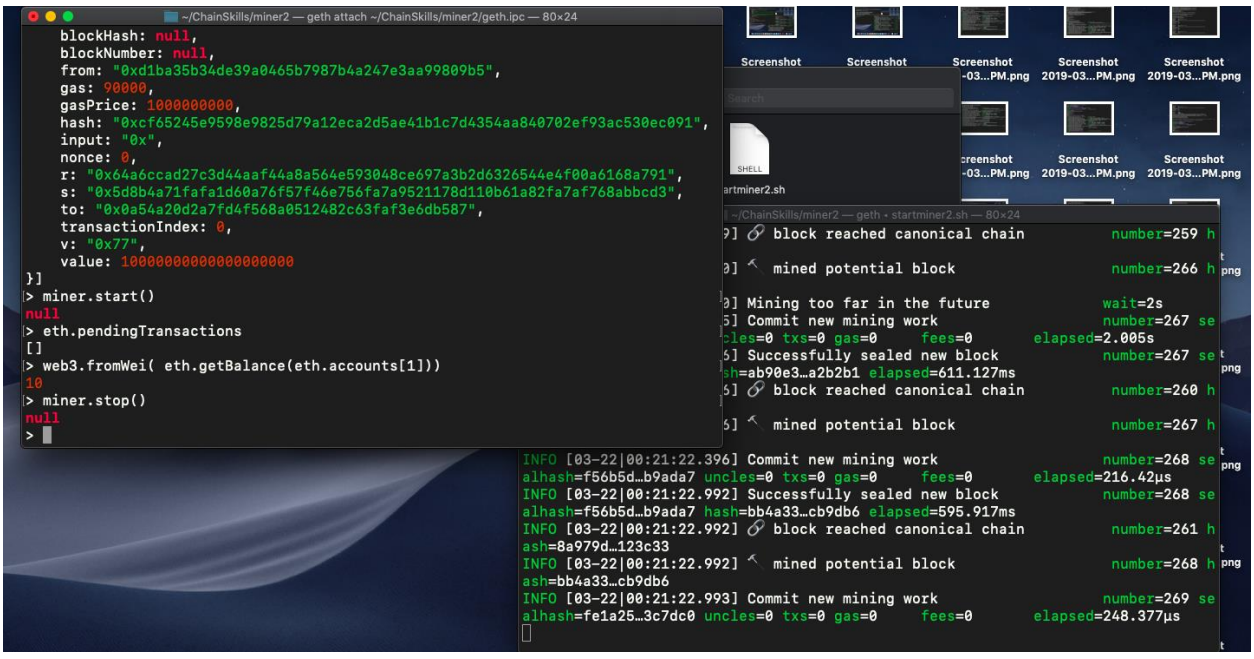
- Let the miner process the transaction
- The transaction is processed
- Check the balance of the recipient



- The recipient address has received 10 ethers.
- miner # 1 is properly installed!
- Let’s do the same for miner #2.

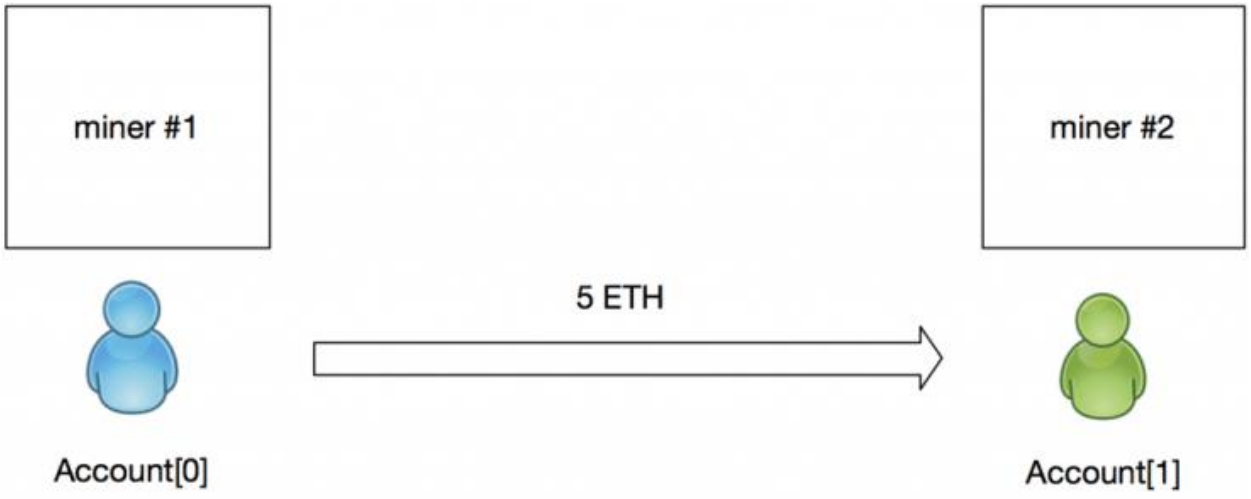
7 Send ethers within miner #2





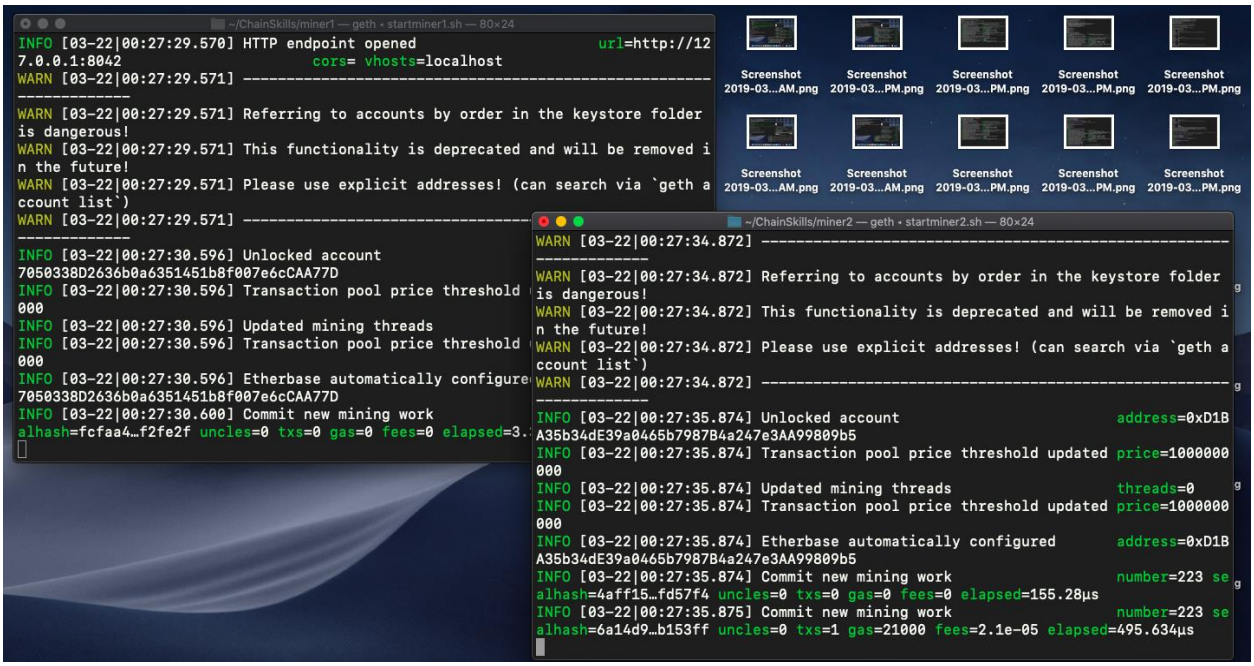
8 Send ethers between nodes of the private block chain

- Now, let's try to send ethers between accounts of our miners.
- We will send 5 ethers from the default account of the miner #1 to the account #1 of the miner #2.



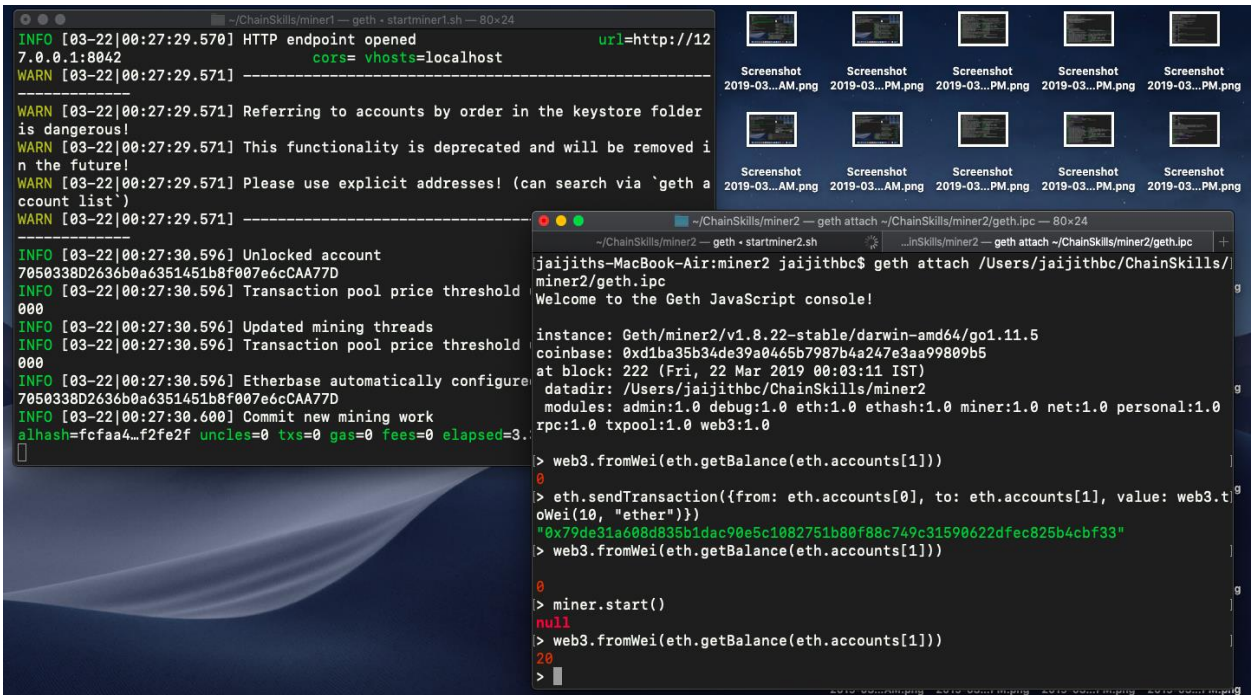
8.1 Start miners

First, start both miners (miner #1 and miner #2).



8.2 Check balance from miner #2

From the Geth console attached to the miner 2, check the initial balance of the account 1:

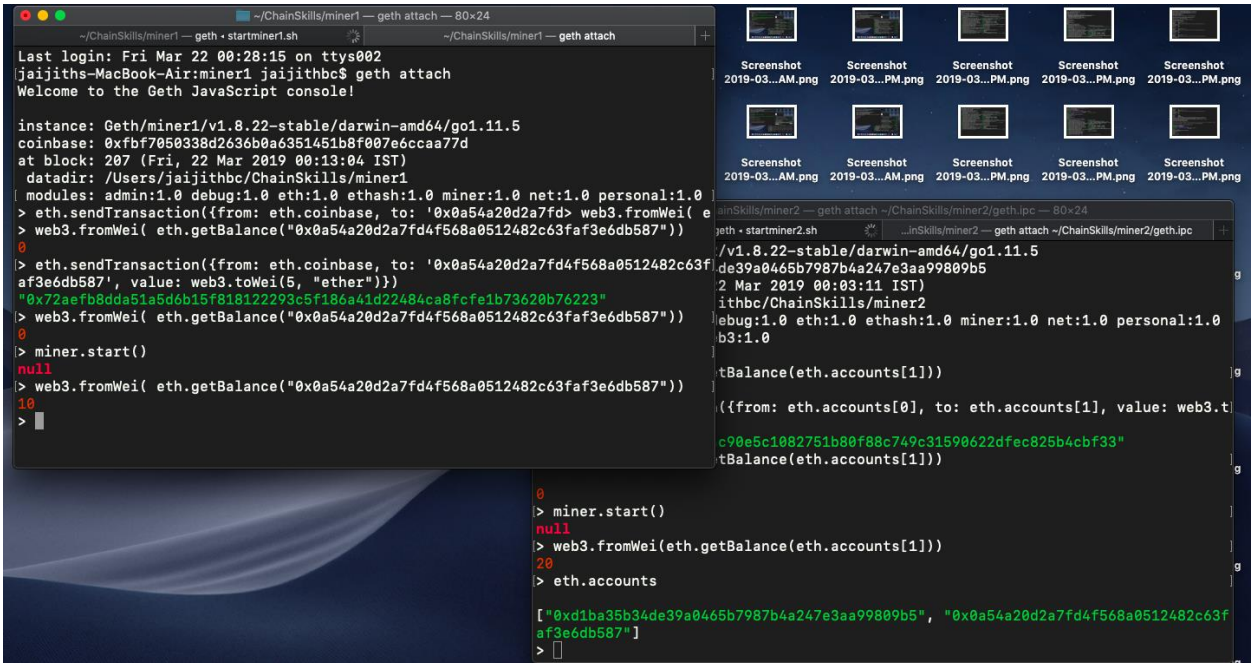


8.3 Send Ethers from miner #1

From the console of the first miner, send 5 ethers twice from the default account (eth.coinbase) to the address of the account 1 in miner #2.

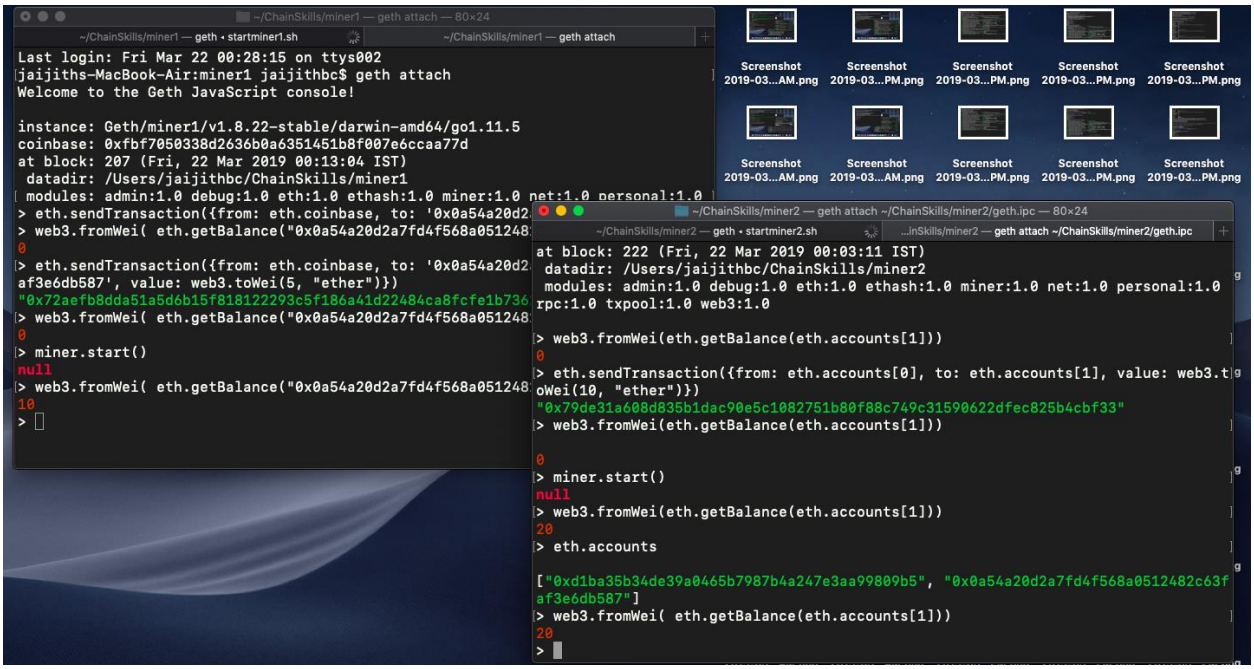
8.4 Check balance from miner #1

- Check the balance from the Geth console attached to the miner #1
- We have 10 ethers while we should expect 30 ethers



8.5 Check balance from miner #2

- Check the balance from the Geth console attached to the miner #2.
- We have 20 ethers while we should expect 30 ethers.



- Because our miners are not connected to the same blockchain.
- Each miner is running its own version of the private blockchain. The transactions are not distributed within the same private blockchain.
- To synchronise the blockchain, we have to pair the nodes with each other.

Part 4: Pair the miners

1. Clean your miners

   Sometimes, miners were not able to pair or the synchronization process became too slow.
   The easiest way is to reset the chaindata of the private blockchain installed on each miner.

1.1 Stop miners

First ensure that your miners are stopped.

As a reminder, you can either press ^C on the miner's console, or search and kill the "geth" process:

```
computer$ ps aux | grep geth



eloudsa 43872 0.0 2.2 556717632 363492 s001 S+ 3:49PM 1:58.01 geth --identity
miner1 --dev



computer$ kill -INT 43872
```

1.2 Delete the chain data

Delete chain data for miners:

```
computer$ rm -rf ~/ChainSkills/miner1/geth

computer$ rm -rf ~/ChainSkills/miner2/geth
```

1.3 Initialize miners

We initialize our miners with the genesis block.

```
jaijiths-MacBook-Air:ChainSkills jaijithbc$ rm -rf ~/ChainSkills/miner1/geth
jaijiths-MacBook-Air:ChainSkills jaijithbc$ rm -rf ~/ChainSkills/miner2/geth
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$ geth --datadir ~/ChainSkills/miner1 init genesis.json
INFO [03-22|00:42:36.562] Maximum peer count                       ETH=25 LES=0 total=25
INFO [03-22|00:42:36.579] Allocated cache and file handles         database=/Users/jaijithbc/ChainSkills/miner1/geth/chaindata cache=16 handles
=16
INFO [03-22|00:42:36.586] Writing custom genesis block
INFO [03-22|00:42:36.586] Persisted trie from memory database      nodes=0 size=0.00B time=13.612µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=
1 livesize=0.00B
INFO [03-22|00:42:36.587] Successfully wrote genesis state         database=chaindata                                          hash=c0990b…cdd1
ab
INFO [03-22|00:42:36.587] Allocated cache and file handles         database=/Users/jaijithbc/ChainSkills/miner1/geth/lightchaindata cache=16 ha
ndles=16
INFO [03-22|00:42:36.594] Writing custom genesis block
INFO [03-22|00:42:36.594] Persisted trie from memory database      nodes=0 size=0.00B time=5.855µs  gcnodes=0 gcsize=0.00B gctime=0s livenodes=
1 livesize=0.00B
INFO [03-22|00:42:36.595] Successfully wrote genesis state         database=lightchaindata                                       hash=c0990b
…cdd1ab
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$
jaijiths-MacBook-Air:ChainSkills jaijithbc$ geth --datadir ~/ChainSkills/miner2 init genesis.json
INFO [03-22|00:43:06.168] Maximum peer count                       ETH=25 LES=0 total=25
INFO [03-22|00:43:06.177] Allocated cache and file handles         database=/Users/jaijithbc/ChainSkills/miner2/geth/chaindata cache=16 handles
=16
INFO [03-22|00:43:06.184] Writing custom genesis block
INFO [03-22|00:43:06.184] Persisted trie from memory database      nodes=0 size=0.00B time=20.087µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=
1 livesize=0.00B
INFO [03-22|00:43:06.185] Successfully wrote genesis state         database=chaindata                                          hash=c0990b…cdd1
ab
INFO [03-22|00:43:06.185] Allocated cache and file handles         database=/Users/jaijithbc/ChainSkills/miner2/geth/lightchaindata cache=16 ha
ndles=16
INFO [03-22|00:43:06.192] Writing custom genesis block
INFO [03-22|00:43:06.192] Persisted trie from memory database      nodes=0 size=0.00B time=4.06µs   gcnodes=0 gcsize=0.00B gctime=0s livenodes=
1 livesize=0.00B
INFO [03-22|00:43:06.193] Successfully wrote genesis state         database=lightchaindata                                       hash=c0990b
…cdd1ab
```

2. Get IP address
   Get the IP address of your computer running miners.

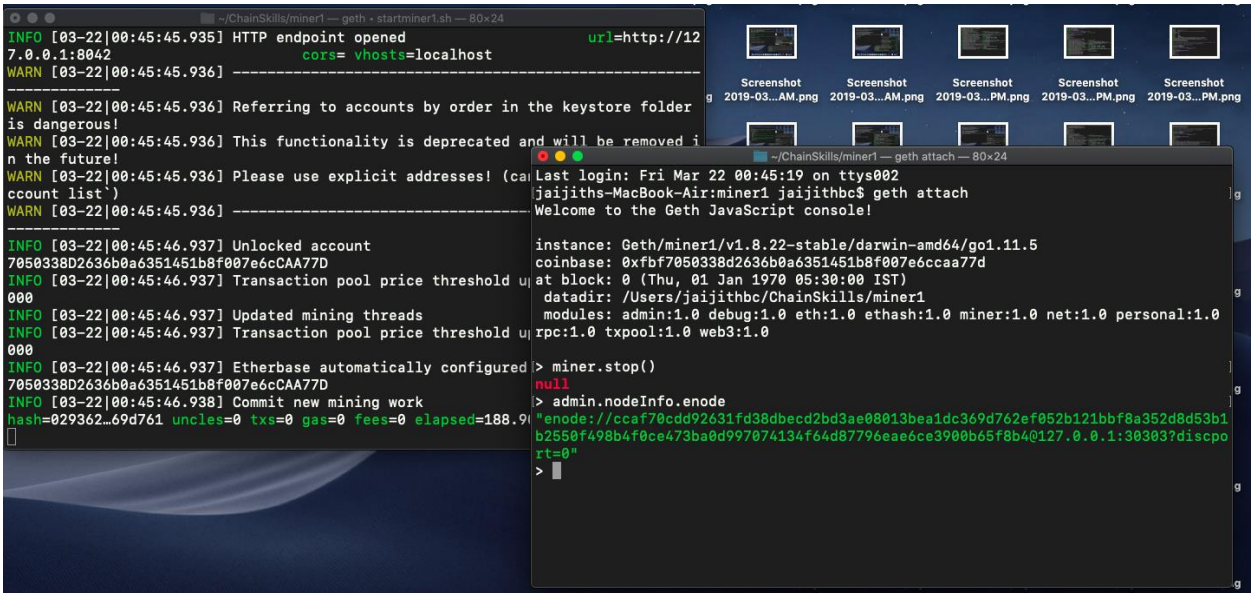
```
jaijiths-MacBook-Air:~ jaijithbc$ ipconfig getifaddr en1
jaijiths-MacBook-Air:~ jaijithbc$ ipconfig getifaddr en0
172.20.10.4
jaijiths-MacBook-Air:~ jaijithbc$
```

Replace the interface according to your network settings:

- **en0**: wired/ethernet
- **en1**: wireless

3. Get Node info from miner #1

3.1 Node info for miner #1



3.2 Get Node info from miner #2

4. Pair the nodes

There are different ways to pair nodes.

Here, we illustrate how to define permanent static nodes stored in a file called "**static-nodes.json**". This file will contain the node information of our miners.
Based on our environment, we will have the following content:

```
[

"enode://b8863bf7c8bb13c3afc459d5bf6e664ed4200f50b86aebf5c70d205d32dd77cf2a888b8ad
f4a8e55ab13e8ab5ad7ec93b7027e73ca70f87af5b425197712d272@192.168.1.103:30303",

"enode://41be9d79ebe23b59f21cbaf5b584bec5760d448ff6f57ca65ada89c36e7a05f20d9cfdd09
1b81464b9c2f0601555c29c3d2d88c9b8ab39b05c0e505dc297ebb7@192.168.1.103:30304"

]
```

You will notice that we have replaced the placeholder [::] with the IP address of our computer.
The last part (**?discport=0**) has been removed.
Each node information is separated by a comma character.

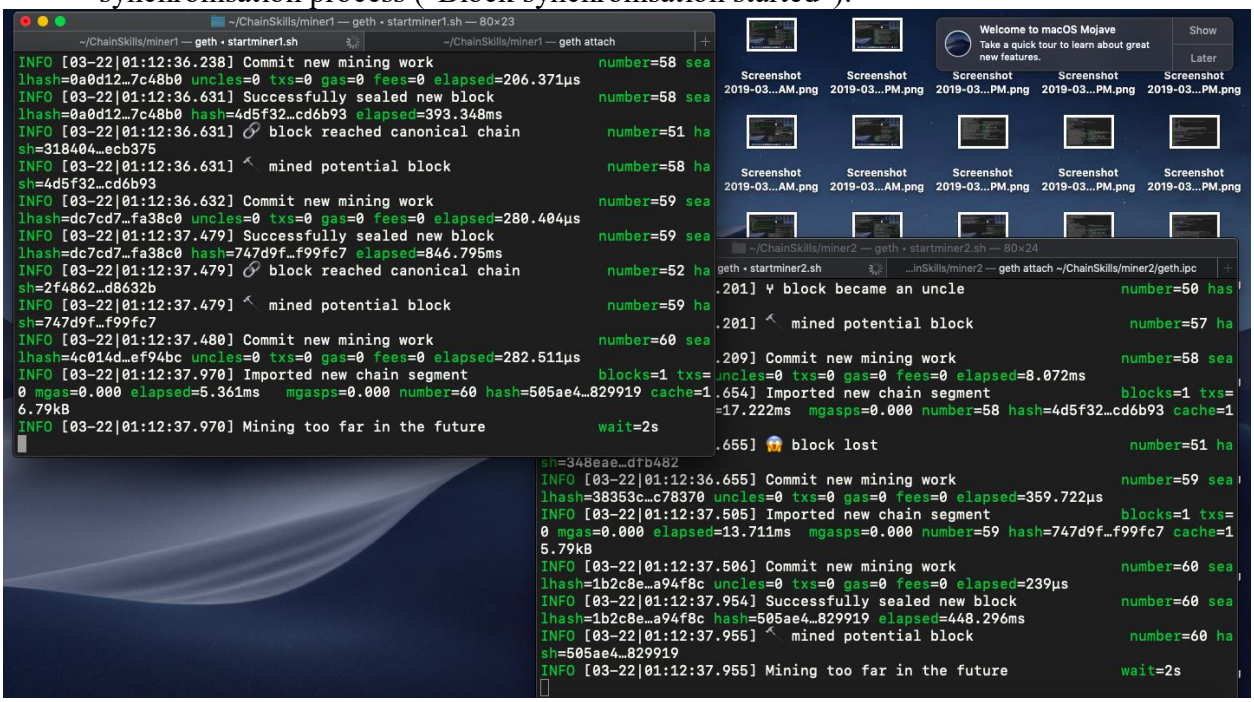
Based on our example, the file "**static-nodes.json**" must be stored under the following location:

- ~/ChainSkills/miner1
- ~/ChainSkills/miner2

When the miner starts, the Geth process will process the file automatically.

5. Restart your miners

- Stop and start the miners to ensure that they will properly reload the "**static-nodes.json**" file.
- If you check the console of your miners, you should see a line mentioning the synchronisation process ("Block synchronisation started"):



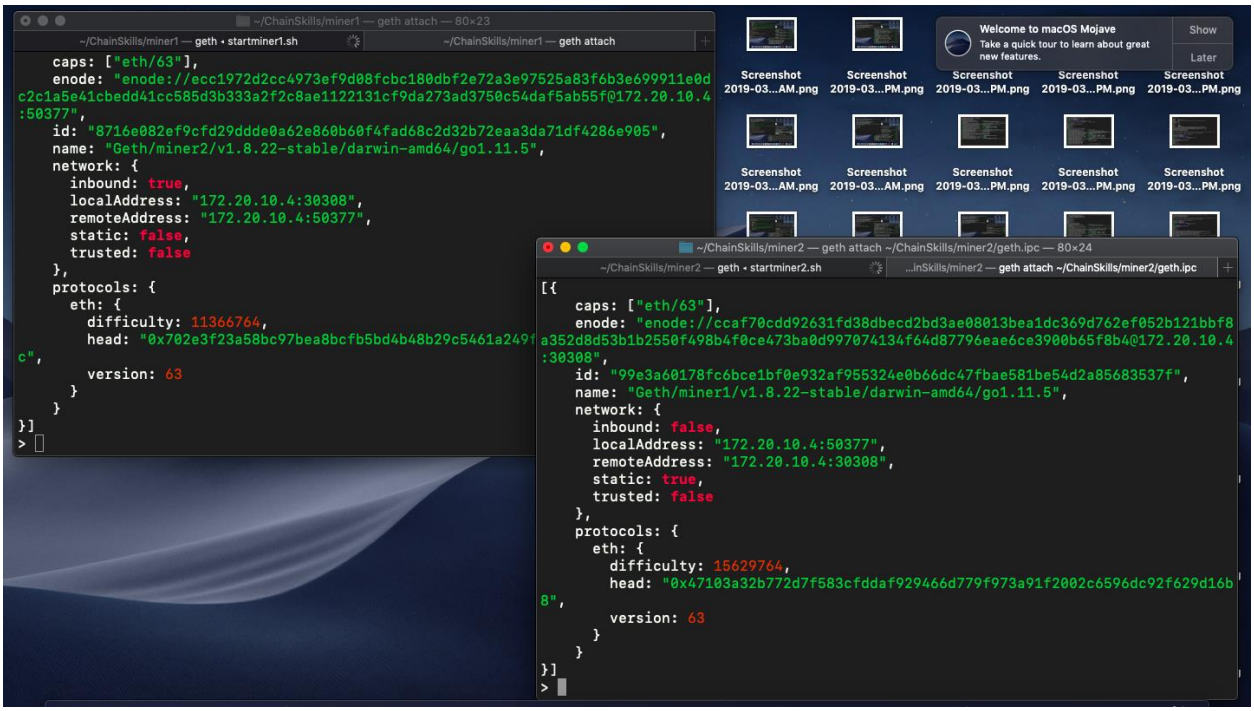
6. Check the synchronization process

6.1 Check from miner #1

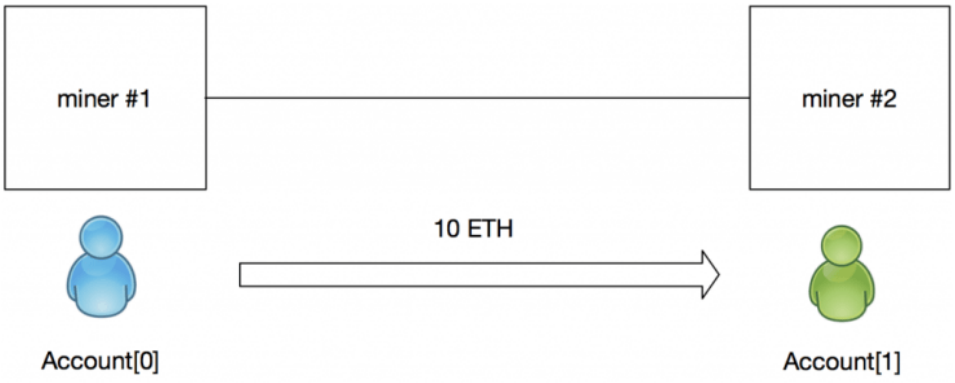
We can see that our node is paired to miner #2 identified by its IP address and its port number (30304).

6.2 Check from miner #2

We can see that our node is paired to miner #1 identified by its IP address and its port number (50377).

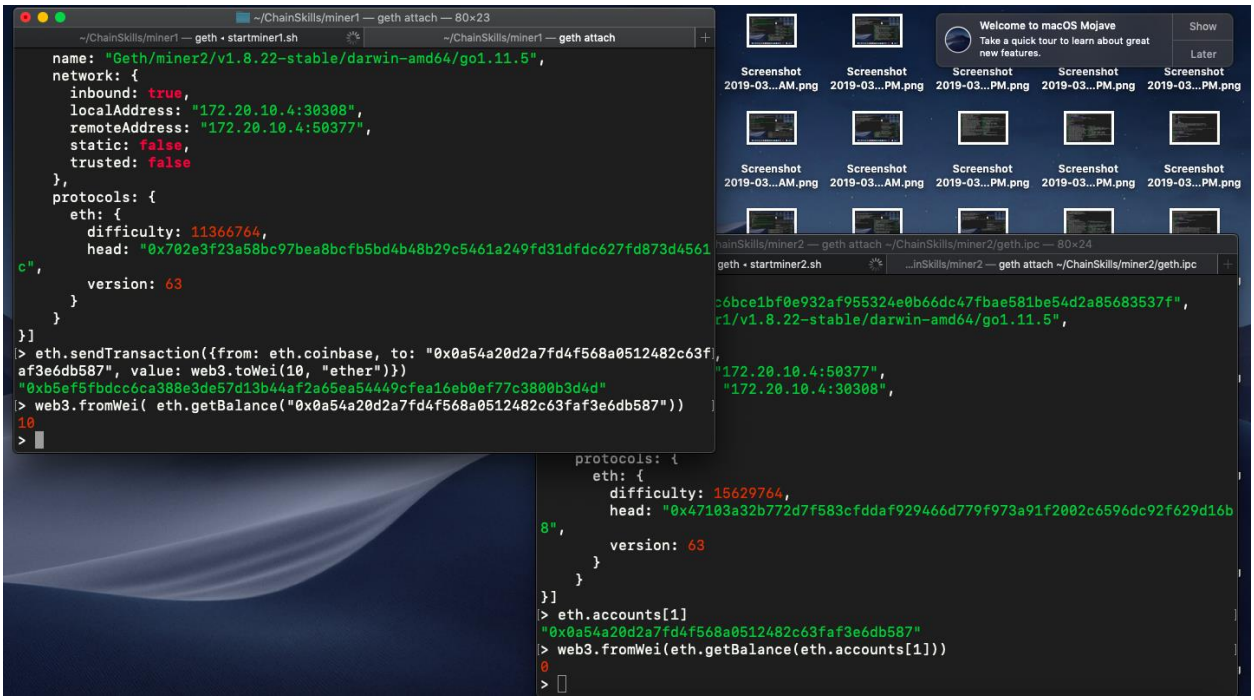


7. Validate the synchronization
   - Let's validate the synchronisation process by sending some ethers between accounts defined on each miner.
   - We are going to send 10 ethers between the following accounts:
   - miner #1(eth.coinbase) -> miner #2(eth.accounts[1])
   - eth.coinbase is the default account that receive the rewards for the mining process. In the miner #1, eth.coinbase is the same as eth.accounts[0].

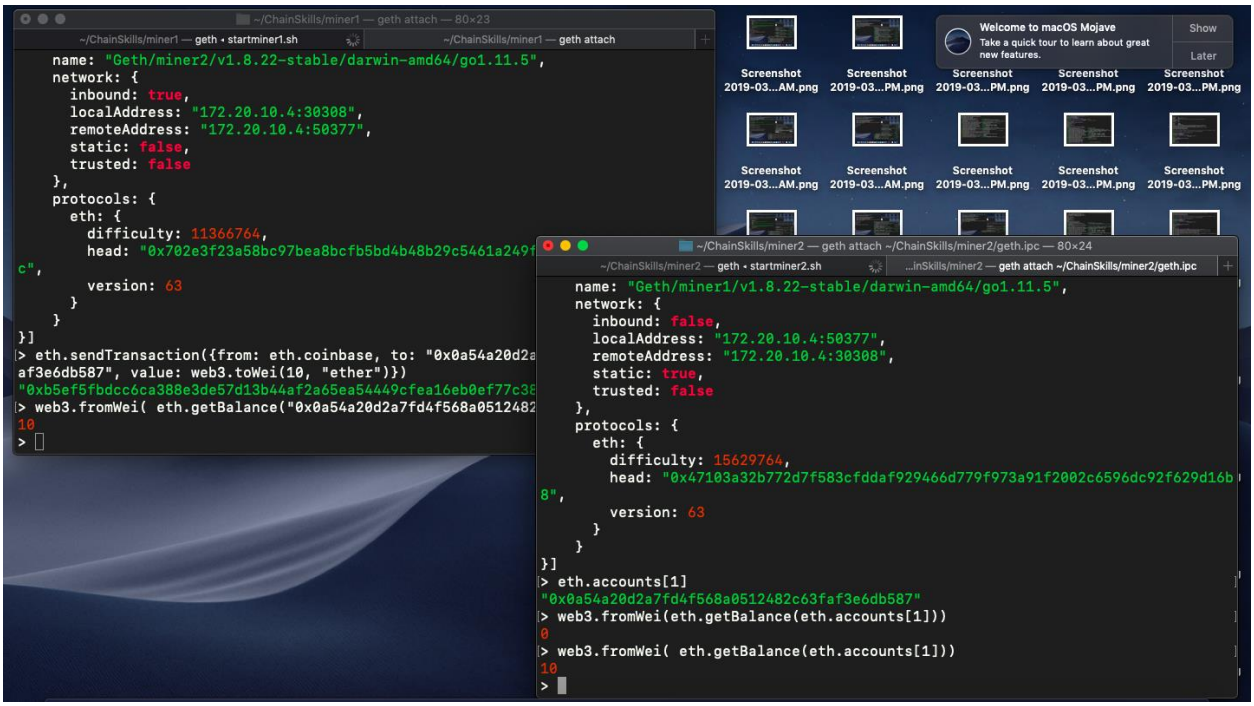


7.1 Send ethers from Miner #1 to Miner #2

- Start a Geth console linked to the miner #2, retrieve the address of the account[1] and check its initial balance
- The account #1 has no ether.
- From the Geth console linked to the **miner #1**, send 10 ethers from the default account to the address of the account[1]
- From the **miner #1**, check if the recipient has received the ethers



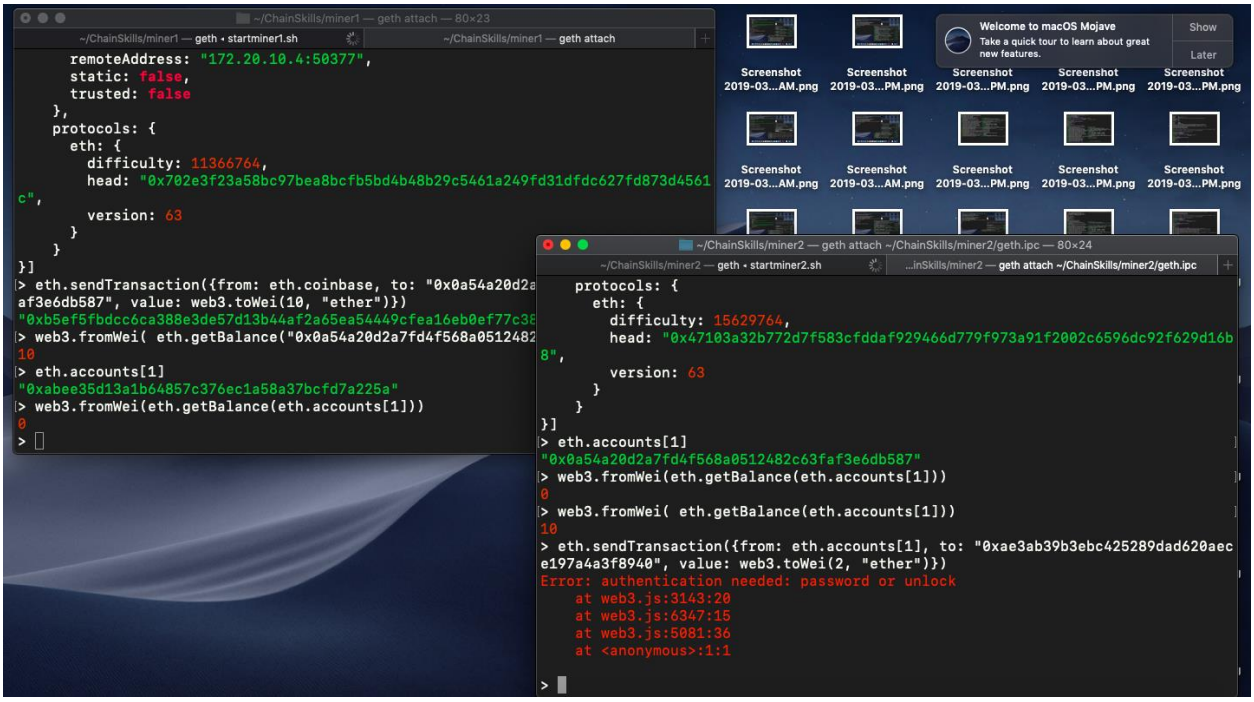
- From the **miner #2**, check that you have the same balance



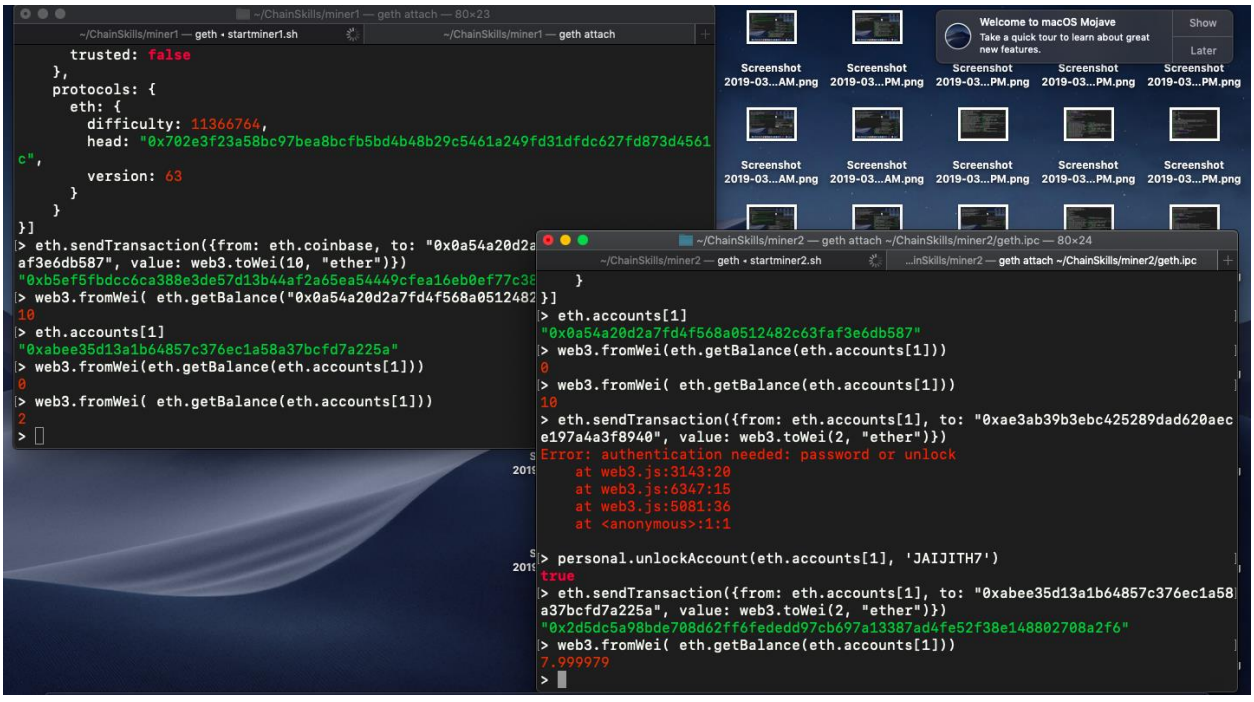
7.2 Send ethers from Miner #2 to Miner #1

- We are going to send 2 ethers between the following accounts:
- miner #2(eth.accounts[1]) -> miner #1(eth.accounts[1])

- From the Geth console linked to the miner #1, check the initial balance of the account that will receive ethers
- The account #1 has 0 ether.
- From the Geth console linked to the **miner #2**, send 2 ethers from the account #1
  Oops! We cannot send the transaction because the account #1 is locked.
  To send transactions from an account, Ethereum requires that you unlock this account. The coinbase account is unlocked by our startminer.sh script thanks to the password file. But all other accounts are locked.



- From the **miner #2**, unlock the account #1 with its password
- From the **miner #2**, we are ready to send our transaction
- From the miner #2, check that our balance has changed (2 ethers plus some transaction fees)
- From the **miner #1**, check that the recipient has received the ethers



Part 5: Synchronize the Raspberry PI with the Private Blockchain

1. Create the data dir folder

As already done for the miners, we have to use a specific folder that will host the data of the private blockchain.

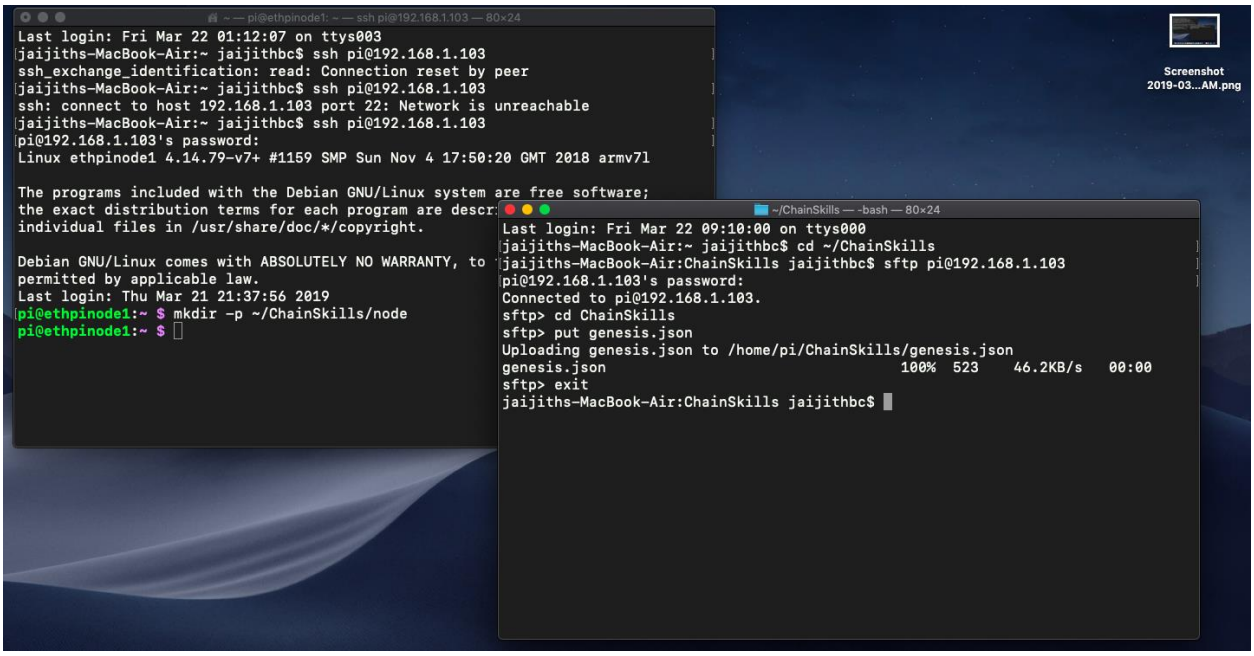
Create this folder with the following command:

```
pi$ mkdir -p ~/ChainSkills/node
```

2. Transfer the genesis file

From your computer, upload the genesis.json file to the RPi:

```
computer$ cd ~/ChainSkills



computer$ sftp pi@192.168.1.31

pi@192.168.1.31's password:

Connected to 192.168.1.31.



sftp> cd ChainSkills



sftp> put genesis.json

Uploading genesis.json to /home/pi/ChainSkills/genesis.json

genesis.json 100% 420 83.8KB/s 00:00



sftp> exit
```



3. Initialize the node

It's time to initialize the private blockchain on the RPi with the genesis block.
The private blochain data will reside in the folder "node".

```
pi$ cd ~/ChainSkills

pi$ geth --datadir ~/ChainSkills/node init genesis.json



I0105 00:13:28.961585 cmd/utils/flags.go:615] WARNING: No etherbase set and no accounts found as default

...

I0105 00:13:29.441262 cmd/geth/chaincmd.go:131] successfully wrote genesis block and/or chain rule set:
6e92f8b23bcdfdf34dc813cfaf1d84b71beac80530506b5d63a2df10fe23a660
```

The log provides the following information:

- you need a default account
- the blockchain has been successfully created

4. Create accounts

- Create an initial account that will be used to run the node.

- To create the default account, type the following command. Keep the password in a safe place

```
pi@ethpinode1:~/ChainSkills $ geth --datadir ~/ChainSkills/node account new
WARN [03-22|09:23:53.064] Sanitizing cache to Go's GC limits       provided=1024
 updated=309
INFO [03-22|09:23:53.072] Maximum peer count                       ETH=25 LES=0
total=25
Your new account is locked with a password. Please give a password. Do not forge
t this password.
Passphrase:
Repeat passphrase:
Address: {2e57c2d87d0f1fcdc6e41f4f13e93a7d475904a4}
pi@ethpinode1:~/ChainSkills $
```

- The account creation might take a few seconds, don't worry.
- The wallet for these accounts is located

```
pi@ethpinode1:~/ChainSkills $ ls -l ~/ChainSkills/node/keystore
total 8
-rw------- 1 pi pi 491 Mar 22 09:24 UTC--2019-03-22T03-53-59.991386949Z--2e57c2d
87d0f1fcdc6e41f4f13e93a7d475904a4
-rw------- 1 pi pi 491 Mar 22 09:24 UTC--2019-03-22T03-54-42.329187659Z--37c73ab
0bb64cd07586e34839cbe81bec066cf70
```

- To list all accounts available on your node, use the following command:

```
pi@ethpinode1:~/ChainSkills $ geth --datadir ~/ChainSkills/node account list
WARN [03-22|09:25:22.896] Sanitizing cache to Go's GC limits       provided=1024
 updated=309
INFO [03-22|09:25:22.903] Maximum peer count                       ETH=25 LES=0
total=25
Account #0: {2e57c2d87d0f1fcdc6e41f4f13e93a7d475904a4} keystore:///home/pi/Chain
Skills/node/keystore/UTC--2019-03-22T03-53-59.991386949Z--2e57c2d87d0f1fcdc6e41f
4f13e93a7d475904a4
Account #1: {37c73ab0bb64cd07586e34839cbe81bec066cf70} keystore:///home/pi/Chain
Skills/node/keystore/UTC--2019-03-22T03-54-42.329187659Z--37c73ab0bb64cd07586e34
839cbe81bec066cf70
pi@ethpinode1:~/ChainSkills $
```

5. Prepare the node

We are ready to start the node from our RPi.
To start the node, we will require to run the following command:

```
geth --identity "node1" --fast --networkid 42 --datadir /home/pi/ChainSkills/node
--nodiscover --rpc --rpcport "8042" --port "30303" --unlock 0 --password
"/home/pi/ChainSkills/node/password.sec" --ipcpath /home/pi/.ethereum/geth.ipc
```

The meaning of the main parameters is the following:

- **identity**: name of our node
- **fast**: fast syncing of the database (more details here)
- **networkid**: this network contains an arbitrary value that will be used to identify all nodes of the network. This value must be different from 0 to 3 (used by the live chains)
- **datadir**: folder where is stored in our private blockchain

- **rpc and rpcport**: enabling HTTP-RPC server and giving its listening port number
- **port**: network listening port number
- **nodiscover**: disable the discovery mechanism (we will pair our nodes later)
- **unlock**: id of the default account
- **password**: path to the file containing the password of the default account
- **ipcpath**: path where to store the filename for IPC socket/pipe

You can find more information about Geth parameters right here.

We recommend you to store the Geth command into a runnable script. In our example, this script is called "**startnode.sh**" and is located here: ~/ChainSkills/node

```
#!/bin/bash



geth --identity "node1" --fast --networkid 42 --datadir /home/pi/ChainSkills/node
--nodiscover --rpc --rpcport "8042" --port "30303" --unlock 0 --password
"/home/pi/ChainSkills/node/password.sec" --ipcpath /home/pi/.ethereum/geth.ipc
```

You should then create the password file in the datadir folder of your node, and this file should just contain the password of the default account. In our example, this file is called "**password.sec**" and is located here: ~/ChainSkills/node

6. Start the node

- Make the script runnable:

```
pi$ cd ~/ChainSkills/node



pi$ chmod +x startnode.sh
```

- Let's start the node:

```
pi$ ./startnode.sh
...
```

- You will notice that the server starts.

7. JavaScript console
   - You can manage your node using the Geth Javascript console.
   - To use this console from your RPi, open a second SSH session attached to your running instance of Geth.
   - Open a new terminal session and type "**geth attach**".

```
pi$ geth attach
...
>
```

8. Stop the mining nodes
   - If your miners are still running on your computer, go to their console window and press CTRL-C.
   - Another option is to kill the Geth running process as illustrated here below:

```
computer$ ps aux | grep geth



eloudsa 43872 0.0 2.2 556717632 363492 s001 S+ 3:49PM 1:58.01 geth --identity
miner1 --dev



computer$ kill -INT 43872
```

9. Synchronize the block chain

   In this step, we are going to link the RPi to the private blockchain already synchronised on our miners.

   9.1 Get Node info
   - Let's start the node:

```
pi$ cd ~/ChainSkills/node

pi$ ./startnode.sh
...
```

   - From your second terminal session, start the Geth console:

```
pi$ geth attach
...
```

```
>
```

- Retrieve the node information:

```
> admin.nodeInfo.enode

"enode://c52b3349d899e1f8ea67c2d01df35c3a40532dec41174460b777bab020079e1a546313552
b91d5f61adb86ed4e74dd80c0ced70c91d658ef0e4f05969a1cf78e@[::]:30303?discport=0"
```

9.2 Update the file "static-nodes.json"

- The file "**static-nodes.json**" created in <u>part 4</u> has to be updated by adding the information of the node deployed on the RPi.
- This file is on your computer under one of the following locations:
  - ~/ChainSkills/miner1
  - ~/ChainSkills/miner2
- Based on our environment, we will have the following content (adjust the values according to your environment):

```
[

"enode://b8863bf7c8bb13c3afc459d5bf6e664ed4200f50b86aebf5c70d205d32dd77cf2a888b8ad
f4a8e55ab13e8ab5ad7ec93b7027e73ca70f87af5b425197712d272@192.168.1.39:30303",

"enode://41be9d79ebe23b59f21cbaf5b584bec5760d448ff6f57ca65ada89c36e7a05f20d9cfdd09
1b81464b9c2f0601555c29c3d2d88c9b8ab39b05c0e505dc297ebb7@192.168.1.39:30304",
"enode://c52b3349d899e1f8ea67c2d01df35c3a40532dec41174460b777bab020079e1a546313552
b91d5f61adb86ed4e74dd80c0ced70c91d658ef0e4f05969a1cf78e@192.168.1.31:30303"
]
```

- The first two entries are related to miner #1 and miner #2. The last row identified the node deployed on the RPi (with its IP address and port number).

- **Make sure your IP addresses are still up-to-date as they tend to change on a local network.**
- This new version of "**static-nodes.json**" must be stored under the following locations:
  - [miner #1] ~/ChainSkills/miner1
  - [miner #2] ~/ChainSkills/miner2
  - [RPi] ~/ChainSkills/node
- You can transfer this file to the RPi using the SFTP command:

```
computer$ cd ~/ChainSkills/miner1
```

```
computer$ sftp pi@192.168.1.31

pi@192.168.1.31's password:

Connected to 192.168.1.31.



sftp> cd ChainSkills/node



sftp> put static-nodes.json

Uploading static-nodes.json to /home/pi/ChainSkills/node/static-nodes.json

static-nodes.json 100% 480 44.8KB/s 00:00



sftp> exit
```

9.3 Restart your block chain

Stop and start each node of your blockchain:

- miner #1
- miner #2
- RPi

10. Check the synchronization process

- Open the Geth console linked to the miner #1:

```
computer$ geth attach
...
>
```

- Check which nodes is paired to the miner #1:

```
> admin.peers[{

 caps: ["eth/62", "eth/63"],
```

```
 id:
"41be9d79ebe23b59f21cbaf5b584bec5760d448ff6f57ca65ada89c36e7a05f20d9cfdd091b81464b
9c2f0601555c29c3d2d88c9b8ab39b05c0e505dc297ebb7",

 name: "Geth/miner2/v1.5.5-stable-ff07d548/darwin/go1.7.4",

 network: {

 localAddress: "192.168.1.39:30303",

remoteAddress: "192.168.1.39:63533"

 },

 protocols: {

 eth: {

 difficulty: 892873381,

 head: "0x263550838a2c63ffe70f91a5bf1851dce951d28d90e7863d19805711bac578e3",

 version: 63

 }

 }

}, {

 caps: ["eth/63"],

 id:
"c52b3349d899e1f8ea67c2d01df35c3a40532dec41174460b777bab020079e1a546313552b91d5f61
adb86ed4e74dd80c0ced70c91d658ef0e4f05969a1cf78e",

 name: "Geth/node1/v1.5.5-stable-ff07d548/linux/go1.7.4",

 network: {

 localAddress: "192.168.1.39:30303",

remoteAddress: "192.168.1.31:50142"
 },
 protocols: {
```

```
 eth: {
 difficulty: 891744952,
 head: "0x6ae07a4c2636835445a68d68219e1bb41c04a9519559a9cc899687218c00253d",
 version: 63
 }
 }
}]
```

- We can see that two nodes are paired with the miner #1: miner #2 and the RPi.
- You can repeat this process on each node to ensure that they are paired with each other.

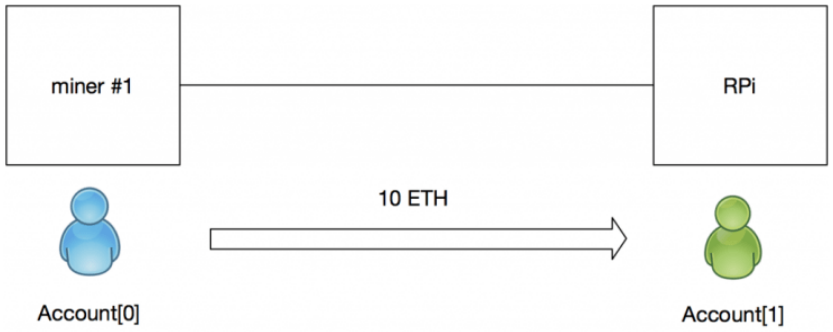
11. Validate the synchronization

- Let's validate the synchronisation process by sending some ethers between accounts defined on each node (miners and RPi).
- Before you proceed, ensure that the mining process is running on both miners.

11.1 Send ethers from Miner #1 to RPI

We are going to send 10 ethers between the following accounts:

- miner #1(eth.coinbase) -> RPI(eth.accounts[1])



- From the Geth console linked to the **RPi**, check the initial balance of the account[1] that will receive ethers:

```
pi$ geth attach
...
> eth.accounts[1]
"0x6af547b83493fd59bf5a2e67546b65191392f45a"

>  web3.fromWei(eth.getBalance(eth.accounts[1]))
0
```

- The account #1 has 0 ether.

- From the Geth console linked to the **miner #1**, send 10 ethers from the default account to the address of the **account[1]** created in the RPi:

```
computer$ geth attach
...
> eth.sendTransaction({from: eth.coinbase, to:
"0x6af547b83493fd59bf5a2e67546b65191392f45a", value: web3.toWei(10,
"ether")})
```

- From the **miner #1 (**or the **miner #2)**, check if the recipient has received the ethers:

```
> web3.fromWei(
eth.getBalance("0x6af547b83493fd59bf5a2e67546b65191392f45a"))
10
```

- From the **RPI**, check that you have the same balance:

```
> web3.fromWei( eth.getBalance(eth.accounts[1]))
10
```

- Of course, the transaction will be processed ONLY if the mining process is started from at least one miner.

Step 11.2 – Send ethers from RPI to Miner #2

We are going to send 2 ethers between the following accounts:

- RPI(eth.accounts[1]) -> miner #2(eth.accounts[1])

- From the Geth console linked to the **miner #2**, check the initial balance of the account that will receive ethers:

```
computer$ geth attach ipc:/Users/eloudsa/ChainSkills/miner2/geth.ipc
...
> eth.accounts[1]
```

```
"0xfa919b49ef34a821fb4cadfdfa5cc6593cb46fe1"

>  web3.fromWei(eth.getBalance(eth.accounts[1]))
7.99958
```

- From the Geth console linked to the **RPI**, send 2 ethers from the account #1 to the address of the account[1] created on the miner #2:

```
pi$ geth attach
...
> eth.sendTransaction({from: eth.accounts[1], to:
"0xfa919b49ef34a821fb4cadfdfa5cc6593cb46fe1", value: web3.toWei(2,
"ether")})
Error: account is locked
 at web3.js:3119:20
 at web3.js:6023:15
 at web3.js:4995:36
 at <anonymous>:1:1
```

- From the **RPi**, unlock the account #1 with its password:

```
> personal.unlockAccount(eth.accounts[1], 'type your password')
true
```

- From the **RPi**, we are ready to send our transaction:

```
> eth.sendTransaction({from: eth.accounts[1], to:
"0xfa919b49ef34a821fb4cadfdfa5cc6593cb46fe1", value: web3.toWei(2, "ether")})
"0x8324e6534d37e250e9d757d3fe867ef3b741cf1b54daa083bf64a24c25a34978"
```

- From the **RPi**, check the balance:

```
> web3.fromWei( eth.getBalance(eth.accounts[1]))
7.99958
```

- From the **miner #2**, check that the recipient has received the ethers:

```
> web3.fromWei( eth.getBalance(eth.accounts[1]))
9.99958
```

CHAPTER 6

ADVANTAGES

a. No central authority
b. Elimination of intermediaries
c. Real-time settlement
d. Drastic reduction in operational costs
e. High levels of transparency

CHAPTER 7

CONCLUSION

Blockchain technology gets a hype in the Information Technology (IT) world because of its security, immutability, transparency, and decentralization as best defined that "Blockchain is shared, immutable ledgers for recording the history of transactions. It fosters a new generation of transactional applications that establish trust, accountability, and transparency- from contracts to deeds to payments".

CHAPTER 8

REFERENCES

- https://chainskills.com/2017/02/24/create-a-private-ethereum-blockchain-with-iot-devices-16/
- https://ieeexplore.ieee.org/search/
- https://stackoverflow.com/
- https://github.com/ethereum/go-ethereum