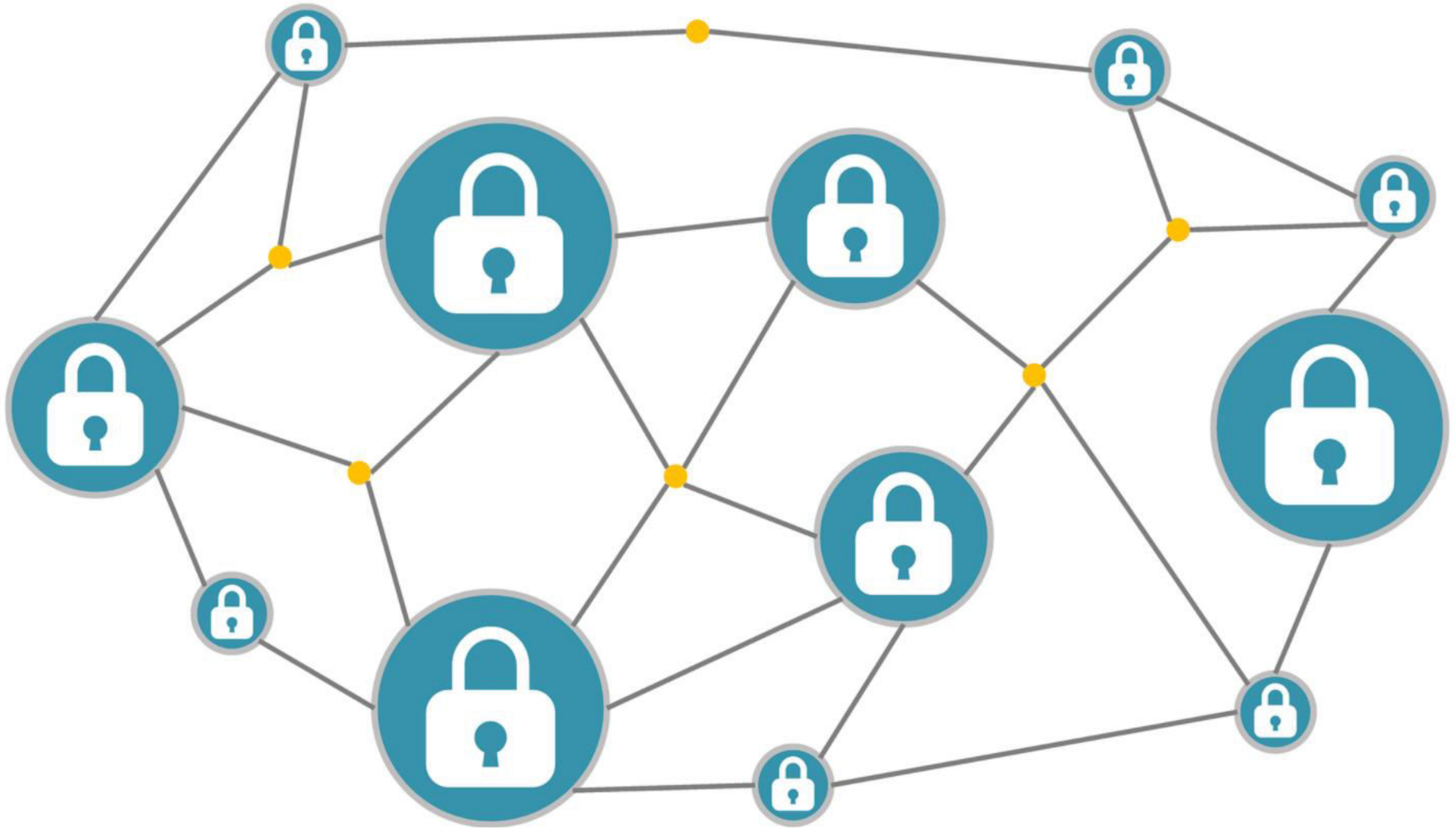
Blockchain Paradigms

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Blockchain

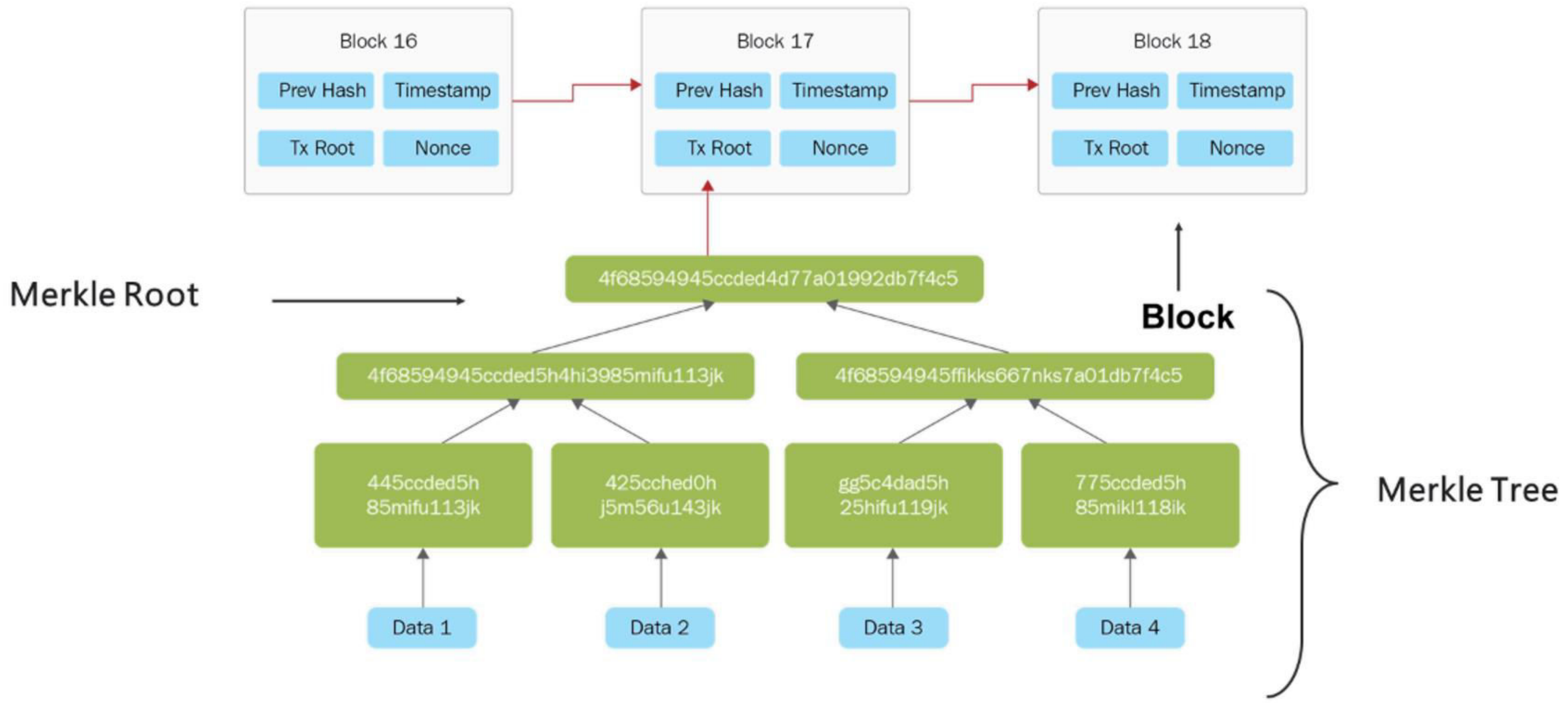
- Structure
- Security
- Consensus protocol



UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES
Département d'informatique

Structure


Block 16
Prev Hash
Timestamp
Tx Root
Nonce
Block 17
Prev Hash
Timestamp
Tx Root
Nonce
Block 18
Prev Hash
Timestamp
Tx Root
Nonce
Merkle Root
4f68594945ccded4d77a01992db7f4c5
Block
4f68594945ccded5h4hi3985mifu113jk
4f68594945ffikks667nks7a01db7f4c5
445ccded5h 85mifu113jk
425cched0h j5m56u143jk
gg5c4dad5h 25hifu119jk
775ccded5h 85mikl118ik
Merkle Tree
Data 1
Data 2
Data 3
Data 4

Security

- Double spending
- DDos attack
- User Wallets
- 51% attack

Consensus protocol

- Use
- Proof of work
- Proof of stake

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

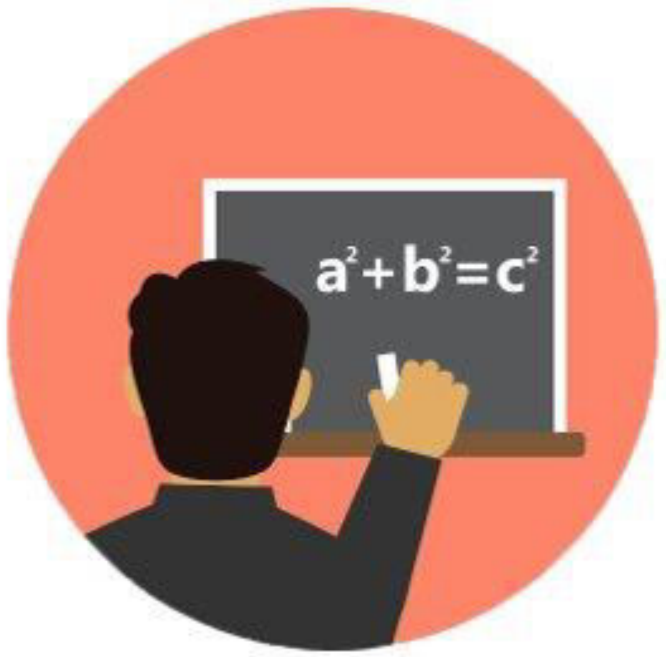
Proof of work

- Algorithm

- Advantages/Disadvantage

Algorithm

- Target hash
- Nonce
- Transaction validation



Advantages/Disadvantages

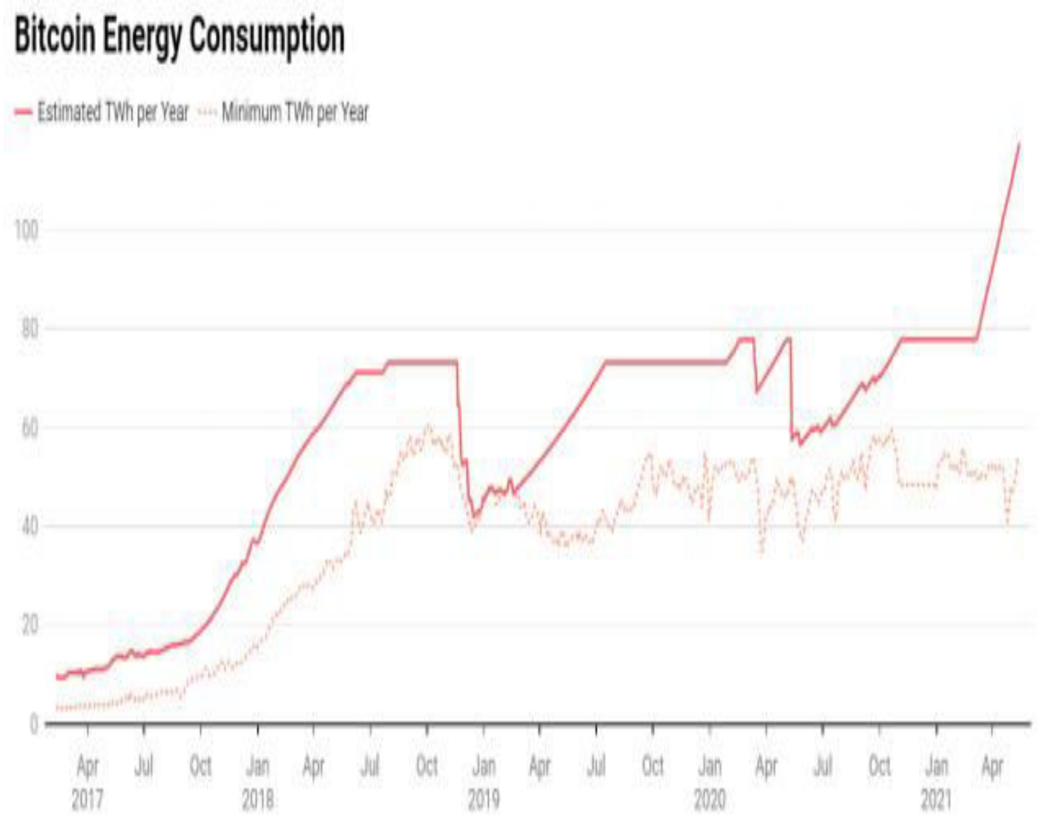
- Fair reward distribution
- Centralization
- Energy Consumption

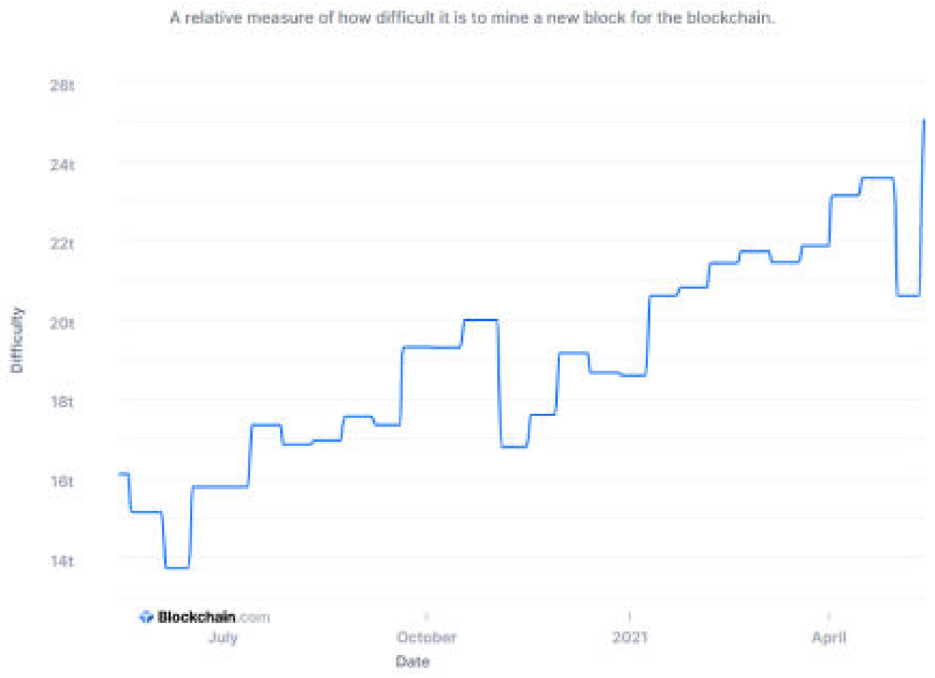
UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

Département d'informatique

Energy Consumption





UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Proof of stake

- Algorithm

- Advantages/Disadvantages

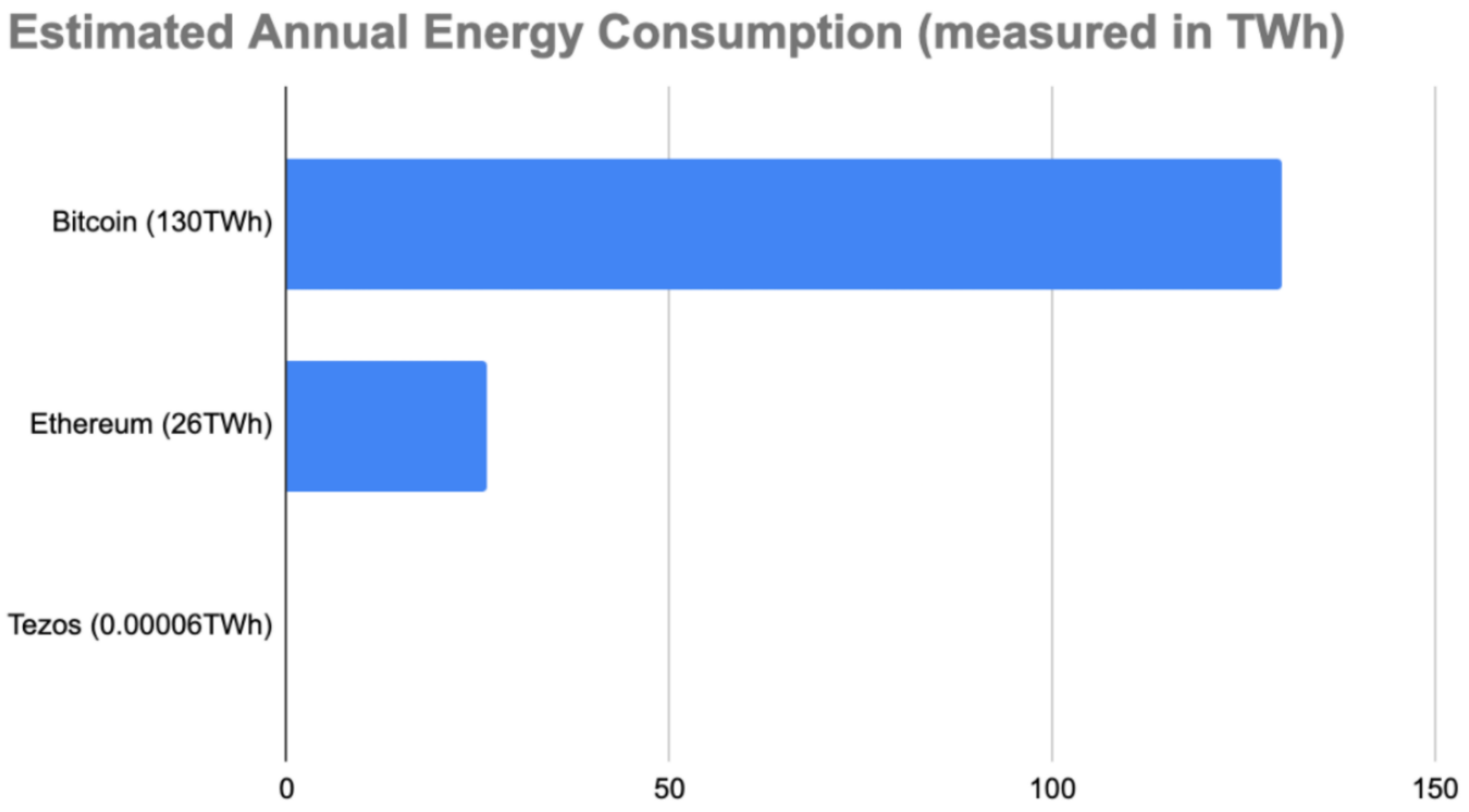
Algorithm

- Stake
- Coin age selection
- Randomised block selection

Advantages/Disadvantages

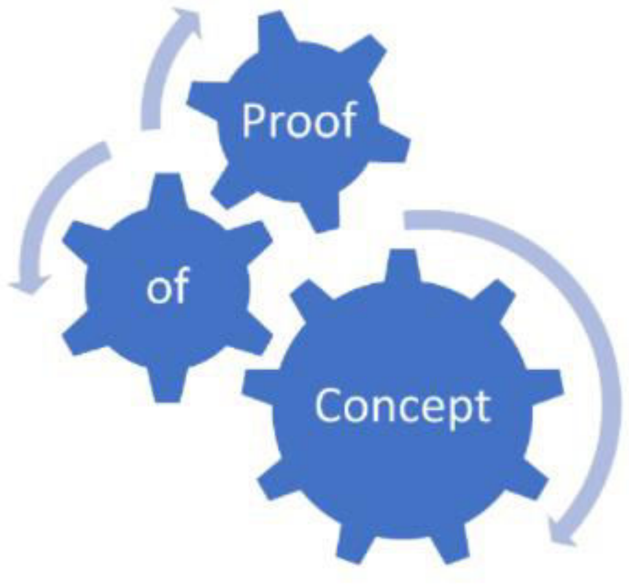
- Fair reward distribution
- Centralization
- Energy Consumption

Energy Consumption



Proof of concept

- Proof of work
- Proof of stake



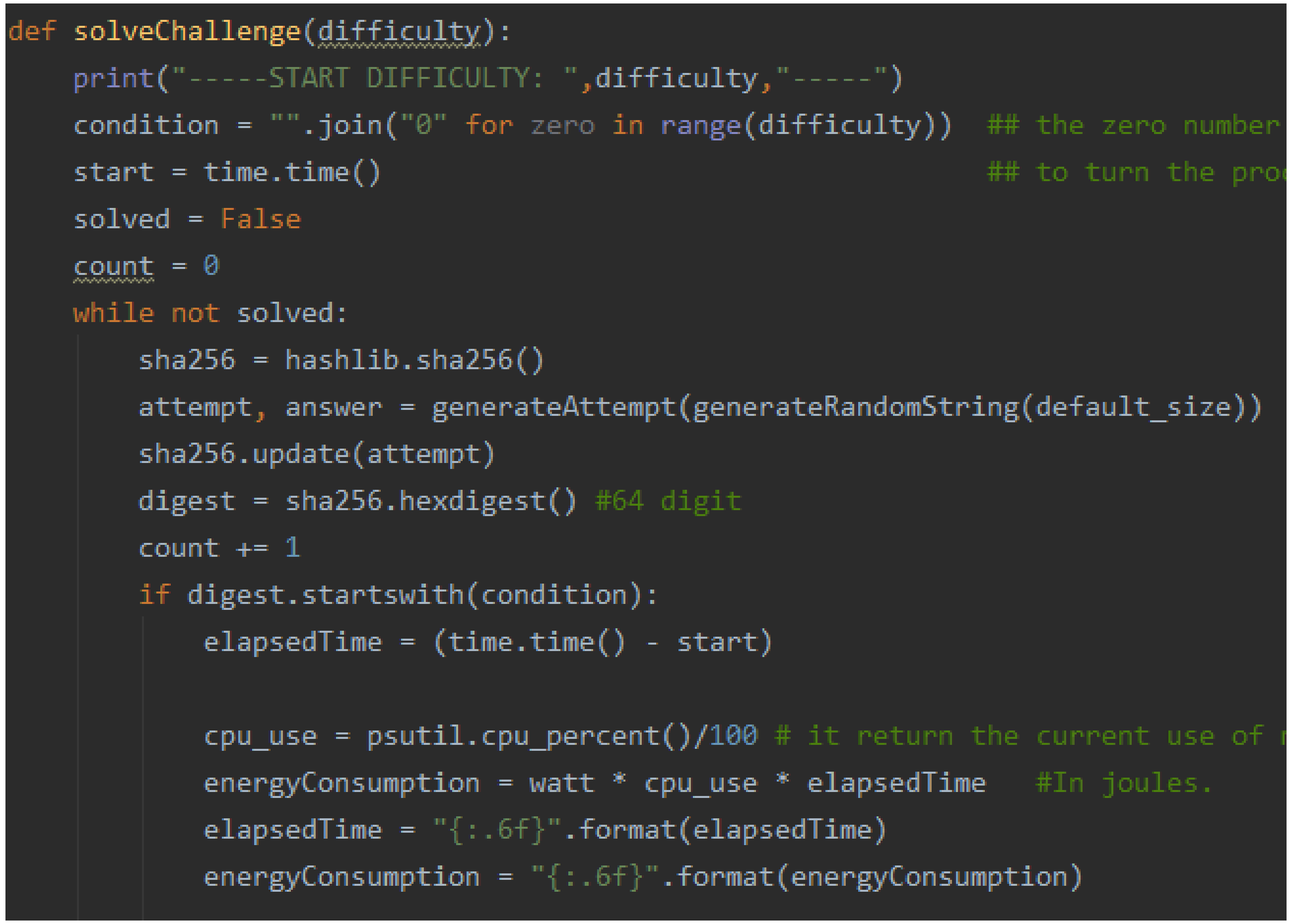
Proof of work

- Implementation

- Results

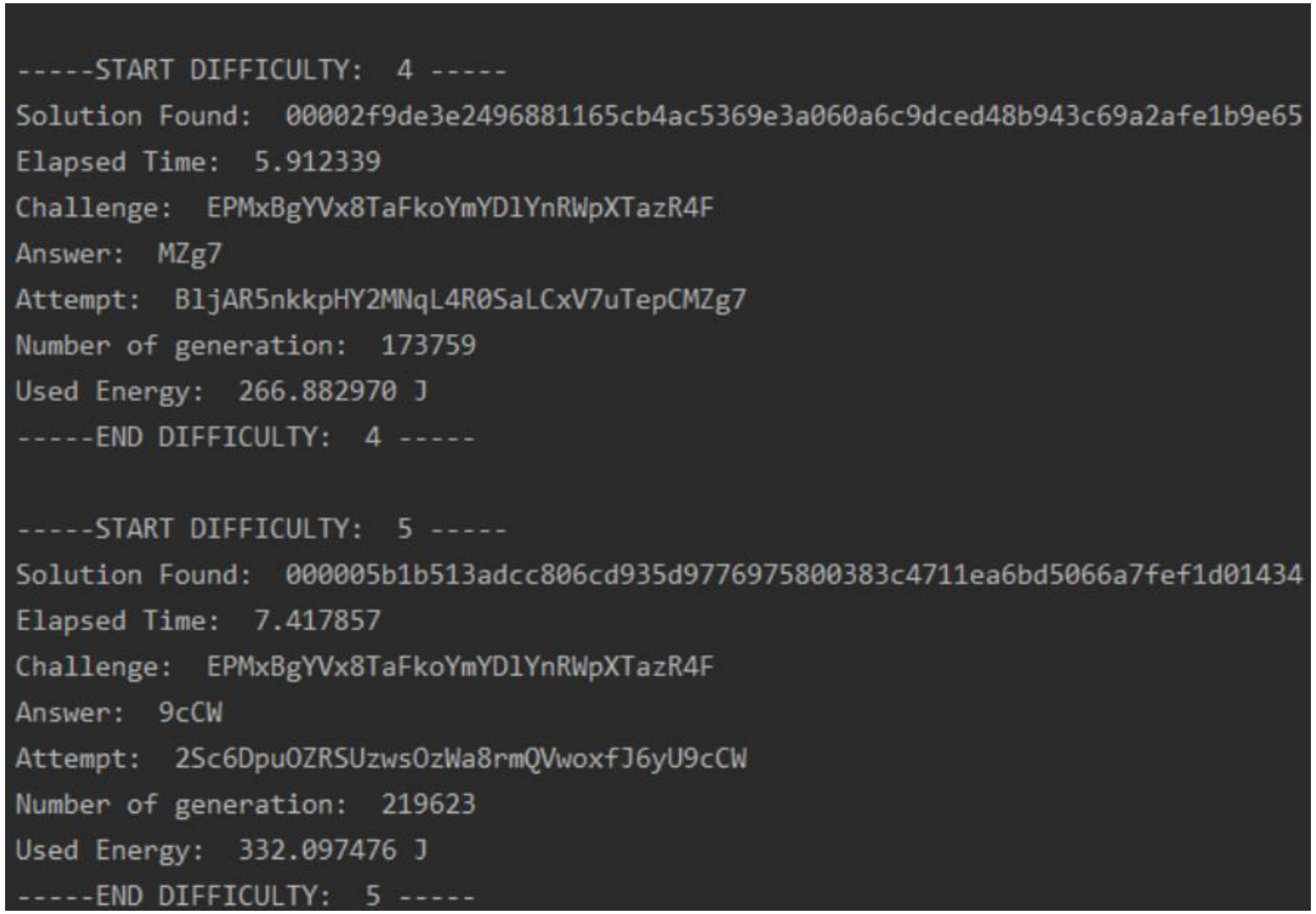
UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Implementation



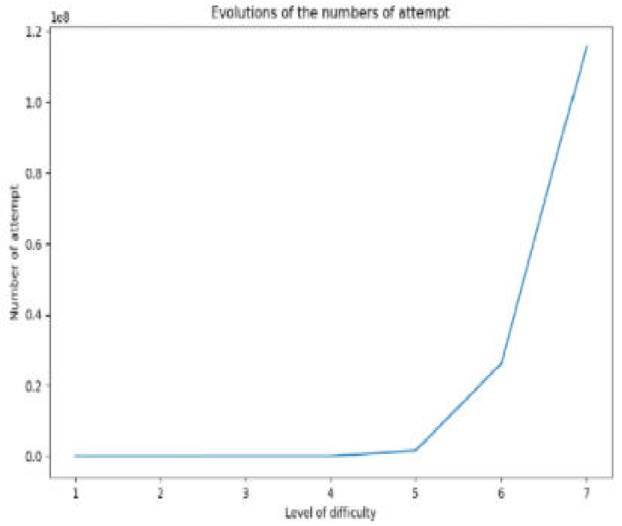
```
def solveChallenge(difficulty):
    print("-----START DIFFICULTY: ",difficulty,"-----")
    condition = "".join("0" for zero in range(difficulty))  ## the zero number
    start = time.time()                                     ## to turn the proc
    solved = False
    count = 0
    while not solved:
        sha256 = hashlib.sha256()
        attempt, answer = generateAttempt(generateRandomString(default_size))
        sha256.update(attempt)
        digest = sha256.hexdigest() #64 digit
        count += 1
        if digest.startswith(condition):
            elapsedTime = (time.time() - start)

            cpu_use = psutil.cpu_percent()/100 # it return the current use of r
            energyConsumption = watt * cpu_use * elapsedTime   #In joules.
            elapsedTime = "{:.6f}".format(elapsedTime)
            energyConsumption = "{:.6f}".format(energyConsumption)
```

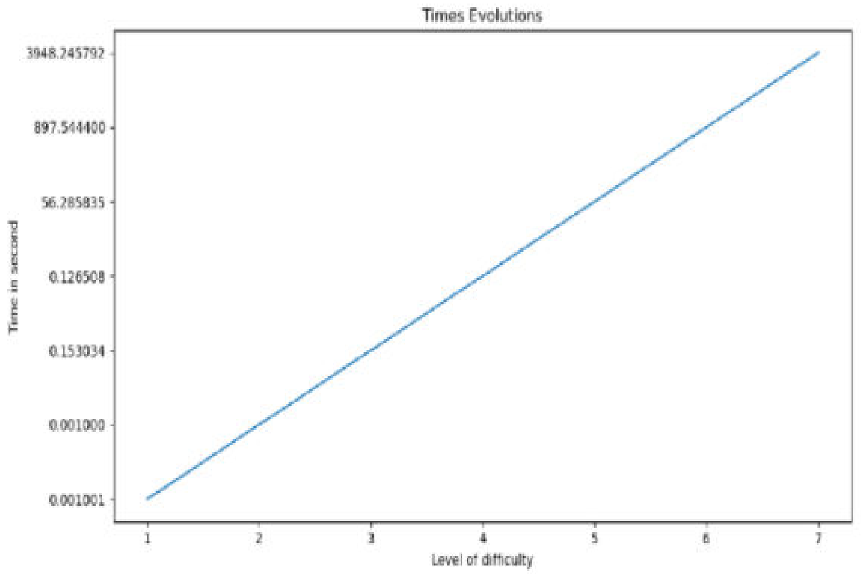
Implementation

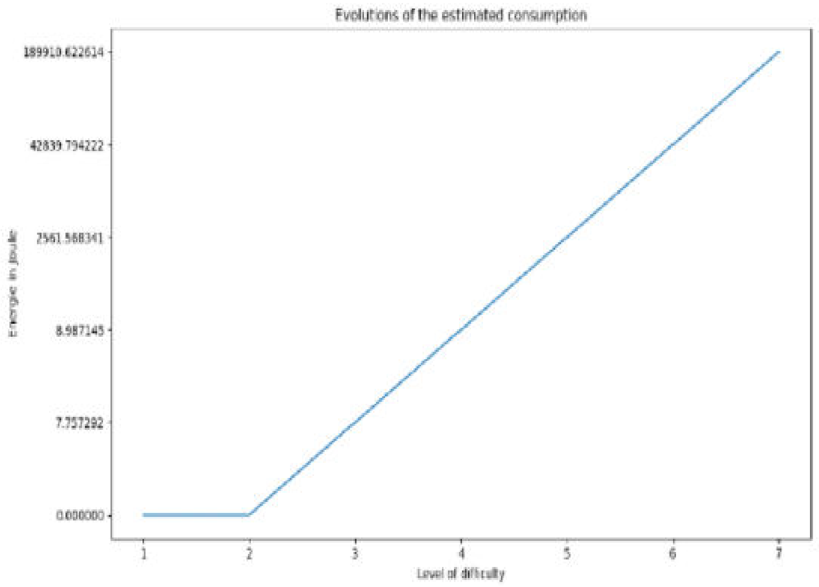


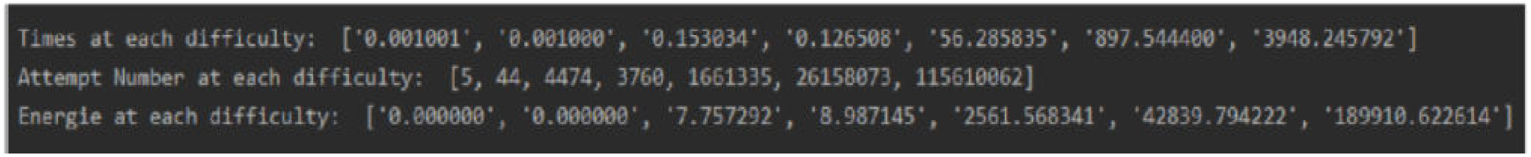
```
-----START DIFFICULTY:  4 -----
Solution Found:  00002f9de3e2496881165cb4ac5369e3a060a6c9dced48b943c69a2afe1b9e65
Elapsed Time:  5.912339
Challenge:  EPMxBgYVx8TaFkoYmYDlYnRWpXTazR4F
Answer:  MZg7
Attempt:  BljAR5nkkpHY2MNqL4R0SaLCxV7uTepCMZg7
Number of generation:  173759
Used Energy:  266.882970 J
-----END DIFFICULTY:  4 -----

-----START DIFFICULTY:  5 -----
Solution Found:  000005b1b513adcc806cd935d9776975800383c4711ea6bd5066a7fef1d01434
Elapsed Time:  7.417857
Challenge:  EPMxBgYVx8TaFkoYmYDlYnRWpXTazR4F
Answer:  9cCW
Attempt:  2Sc6DpuOZRSUzwsOzWa8rmQVwoxfJ6yU9cCW
Number of generation:  219623
Used Energy:  332.097476 J
-----END DIFFICULTY:  5 -----
```

Results









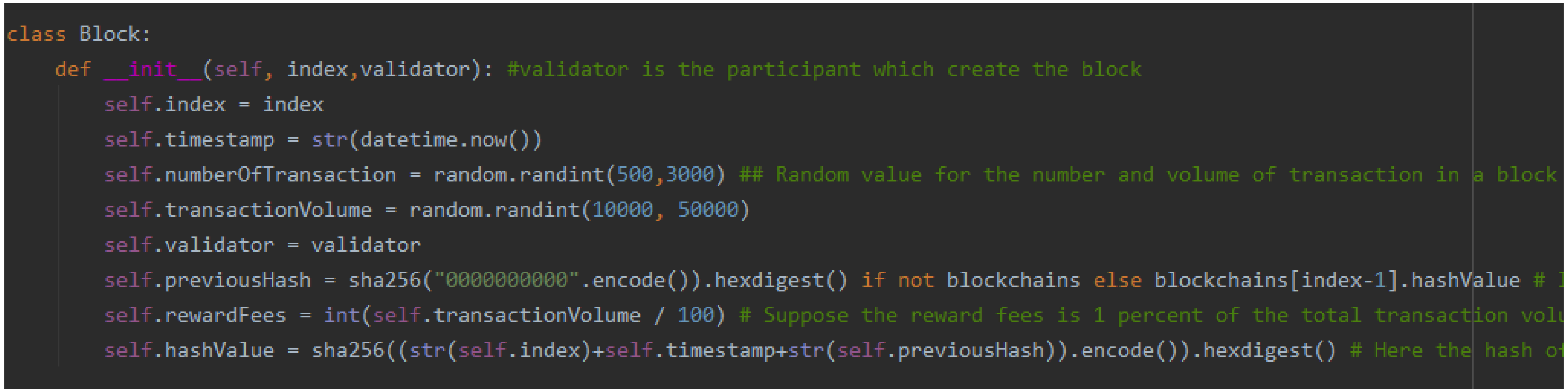
```
Times at each difficulty:  ['0.001001', '0.001000', '0.153034', '0.126508', '56.285835', '897.544400', '3948.245792']
Attempt Number at each difficulty:  [5, 44, 4474, 3760, 1661335, 26158073, 115610062]
Energie at each difficulty:  ['0.000000', '0.000000', '7.757292', '8.987145', '2561.568341', '42839.794222', '189910.622614']
```

Proof of stake

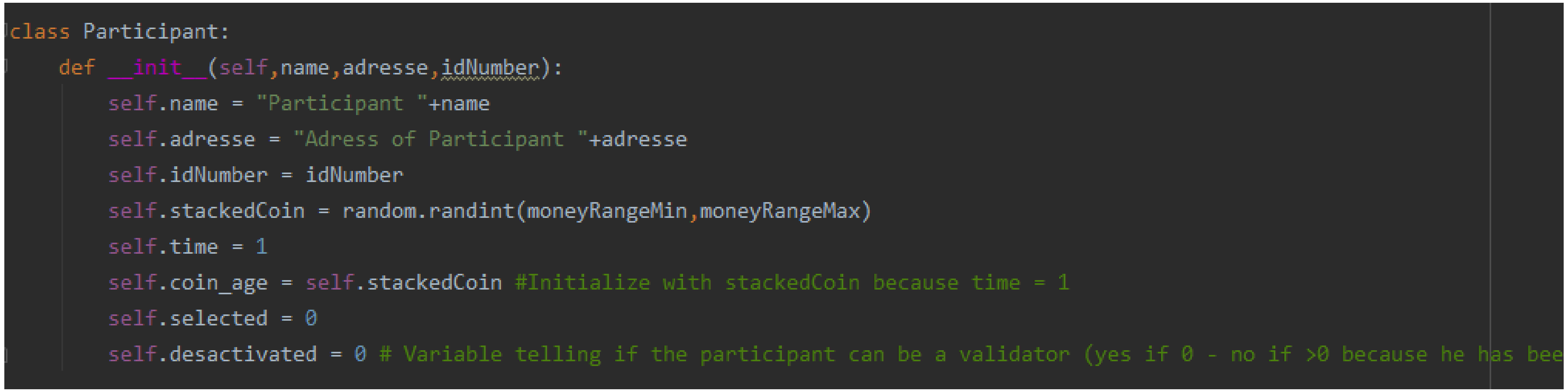
- Implementation

- Results

Implementation

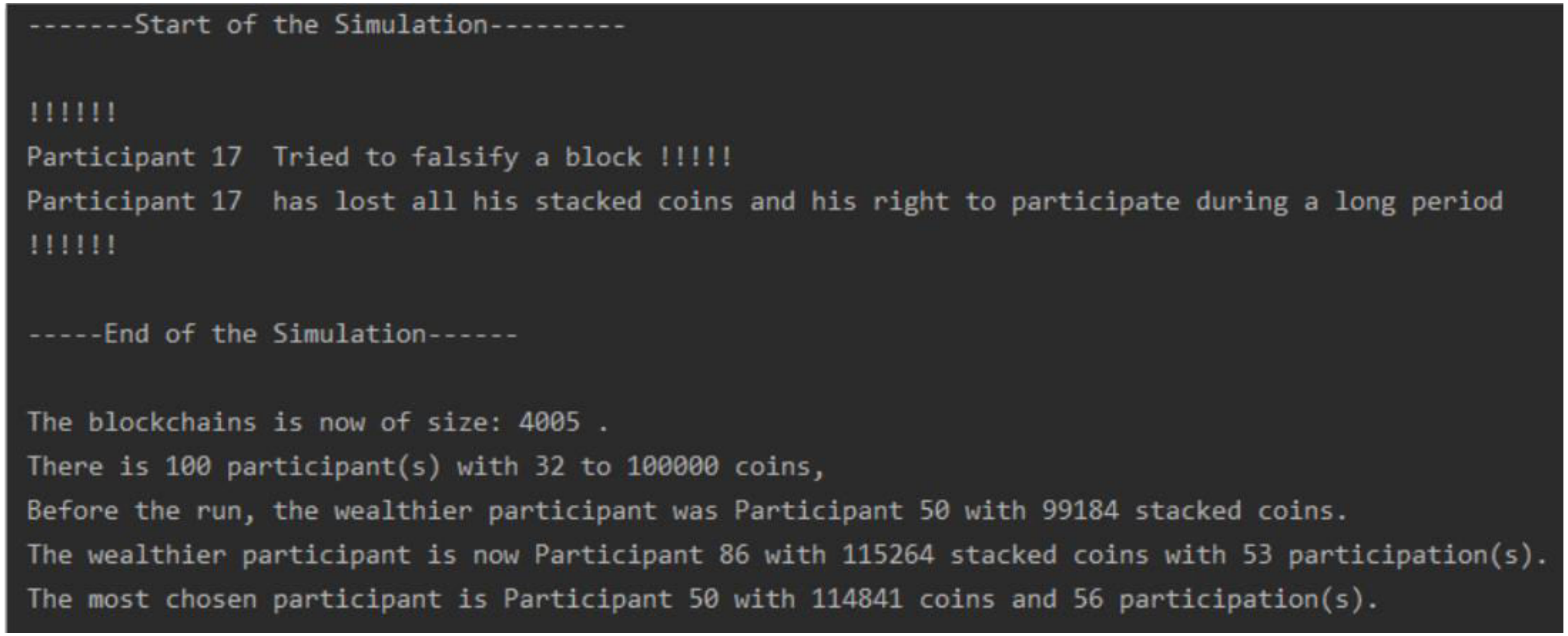


```python
class Block:
    def __init__(self, index,validator): #validator is the participant which create the block
        self.index = index
        self.timestamp = str(datetime.now())
        self.numberOfTransaction = random.randint(500,3000) ## Random value for the number and volume of transaction in a block
        self.transactionVolume = random.randint(10000, 50000)
        self.validator = validator
        self.previousHash = sha256("0000000000".encode()).hexdigest() if not blockchains else blockchains[index-1].hashValue # I
        self.rewardFees = int(self.transactionVolume / 100) # Suppose the reward fees is 1 percent of the total transaction volu
        self.hashValue = sha256((str(self.index)+self.timestamp+str(self.previousHash)).encode()).hexdigest() # Here the hash o
```



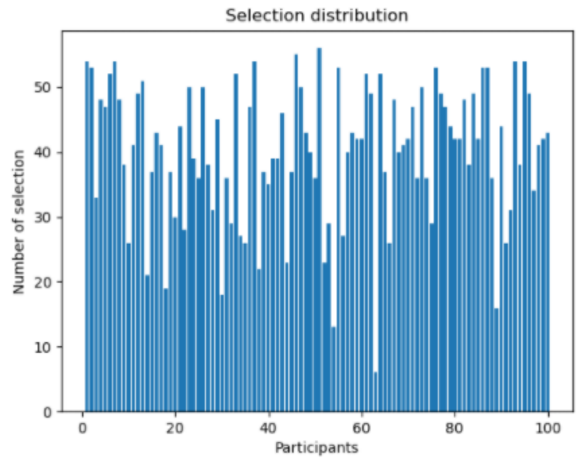
```python
class Participant:
    def __init__(self,name,adresse,idNumber):
        self.name = "Participant "+name
        self.adresse = "Adress of Participant "+adresse
        self.idNumber = idNumber
        self.stackedCoin = random.randint(moneyRangeMin,moneyRangeMax)
        self.time = 1
        self.coin_age = self.stackedCoin #Initialize with stackedCoin because time = 1
        self.selected = 0
        self.desactivated = 0 # Variable telling if the participant can be a validator (yes if 0 - no if >0 because he has bee
```

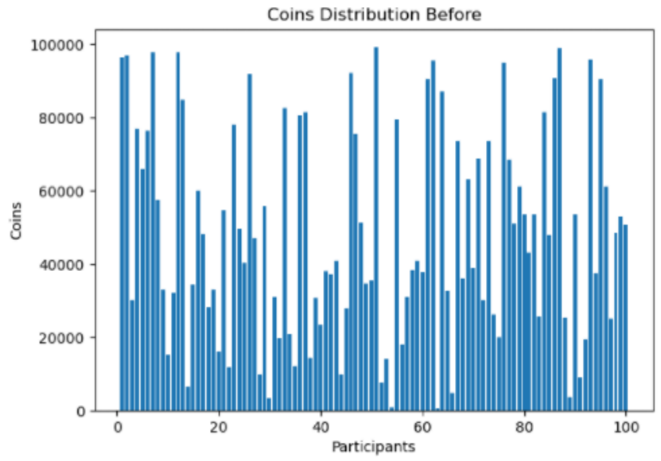
Implementation

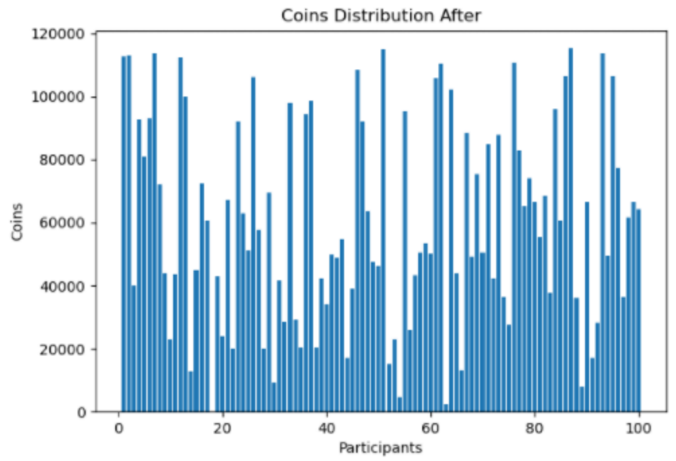


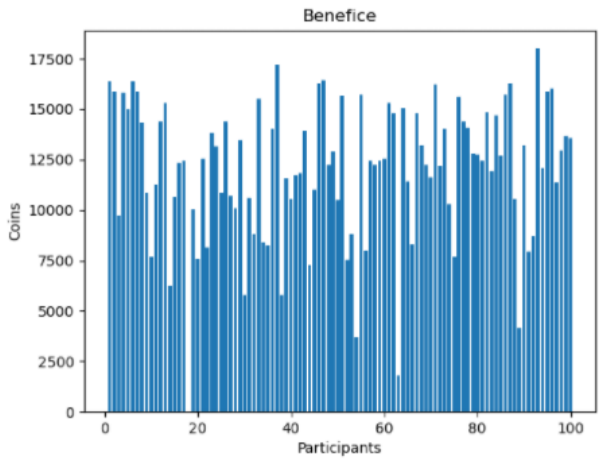
```
-------Start of the Simulation---------

!!!!!!
Participant 17  Tried to falsify a block !!!!!
Participant 17  has lost all his stacked coins and his right to participate during a long period
!!!!!!

-----End of the Simulation------

The blockchains is now of size: 4005 .
There is 100 participant(s) with 32 to 100000 coins,
Before the run, the wealthier participant was Participant 50 with 99184 stacked coins.
The wealthier participant is now Participant 86 with 115264 stacked coins with 53 participation(s).
The most chosen participant is Participant 50 with 114841 coins and 56 participation(s).
```

Results


Selection distribution
Number of selection
Participants


Coins Distribution Before
Coins
Participants


Coins Distribution After
Coins
Participants


Benefice
Coins
Participants

Thank you for your attention