Blockchain: A powerful tool in the fight against counterfeit shoes

Jevan Chow Jun Kiat, Ryan Suan Zhan Hui, Sim Wei Xuan, Tan Kian Yun, Yong Javen
Infocomm Technology Cluster
Singapore Institute of Technology
Singapore 567739
2101144@sit.singaporetech.edu.sg, 2100776@sit.singaporetech.edu.sg, 2100895@sit.singaporetech.edu.sg,
2103012@sit.singaporetech.edu.sg, 2100992@sit.singaporetech.edu.sg

***Abstract*—As technology continues to grow exponentially, there has been a rise in fake products being sold at the same cost as their real counterparts. This is usually done by freelance sellers who are trying to find an easy way to earn the extra buck. One such product that is prone to the creation of fake versions is shoes, and counterfeit shoes have become a growing problem in the global footwear industry, causing significant economic losses to legitimate businesses and posing serious risks to consumer safety. To address this issue, many companies are exploring the use of blockchain technology as a means of verifying the authenticity of shoes and preventing counterfeiting. What was finalized is a program that can ensure the authenticity of a shoe transactions and that buyers do not fall victim to cyber-attacks such as man-in-the-middle attacks by using blockchain technology.**

I. Introduction

Blockchain technology offers a decentralized, transparent, and secure way to store and transfer data, making it ideal for a wide range of applications, including supply chain management. In the fashion industry, particularly in the realm of shoes, blockchain technology and cryptography methods can be used to combat the rising problem of counterfeit products, which not only deceive buyers by offering poor-quality products, but also harm the reputation of genuine brands by associating them with inferior products. Our project is a cryptography program that will encrypt multiple data with various cryptography methods. It will simulate transactions between a shoe seller and a buyer in a blockchain network.

II. Background Research

A. Literature Review

In paper [1], the authors mentioned how blockchain allows transactions take place in a decentralized manner and has been majorly used in various fields such as finance and IoT, and despite its popularity, various issues such as scalability and security problems still arise. Therefore, our program will reduce or even eliminate such security flaws by ensuring that the scalability is more balanced throughout without majorly affecting its performance.

Like paper [1], the authors of paper [2] also mentioned about blockchain becoming more popular in various industries, and reiterated about how in blockchain, a buyer and seller need not even trust each other, as the cryptography used in such blockchain methodology already ensures the security of transactions within the network. Our program will demonstrate how a buyer and seller who may or may not know each other perform a seamless transaction, ensuring that the trust between the two parties is prominent.

In paper [3], the author mentioned how electronic cash allows online payments to be sent directly from one party to another without going through a bank or other financial firm, making them more of a third party to these kinds of transactions. Interestingly, this paper is the original research paper that introduced Bitcoin, the first cryptocurrency, and its underlying blockchain technology.

In paper [4], the author attributed his research to the creation of Bitcoin, courtesy of the author of paper [3]. This paper introduced Ethereum, a blockchain platform that supports the creation of decentralized applications and smart contracts. This platform inspired our group to research on Ethereum and understand how such blockchain tools are designed.

B. Existing Tools and Solutions

As part of our research, our group took inspiration from one of the popular existing tools in which their functionality is somewhat related to our program, and a brief description of some of the tools are listed below.

Ethereum [5] is a is a decentralized, open source blockchain platform which is designed to allow developers to build and deploy decentralized applications (DApps) on the blockchain. It allows users to create and execute smart contracts using a proof-of-work consensus algorithm. An Ethereum blockchain explorer such as Etherscan can be used to gather data on transactions, addresses, blocks, and tokens on the Ethereum network.

III. Proposed Solution

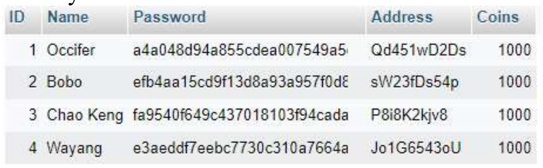
Our proposed solution uses three python files to simulate a real transaction between a manufacturer and a buyer, with the addition of a miner. A database was created using MySQL to store various manufacturer and buyer data. For a start, our program has one manufacturer and four buyers (which can be configured in the database). To simulate real cash, each manufacturer and buyer starts off with 1000 coins (also configured in database), where a buyer can spend coins on 4 different brands of shoes (Nike, Adidas, Puma, Under Armour), which cost 10, 20, 30 and 40 coins respectively. Upon a successful transaction, the buyer's and manufacturer's coins will be simultaneously updated in the database.

Three python files are required for each of the three 'actors', named *manufacturer.py*, *buyer.py* and *miner.py*. A brief description of each of our python files is explained below.

A. sneaksecure.sql

A SQL database was created to store data of the buyers, manufacturer and shoe models. The following are the three tables stored in the database, namely buyer, manufacturers and shoe.

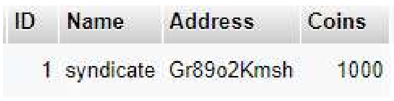
1. Buyer table



ID	Name	Password	Address	Coins
1	Occifer	a4a048d94a855cdea007549a5	Qd451wD2Ds	1000
2	Bobo	efb4aa15cd9f13d8a93a957f0d8	sW23fDs54p	1000
3	Chao Keng	fa9540f649c437018103f94cada	P8i8K2kjv8	1000
4	Wayang	e3aeddf7eebc7730c310a7664a	Jo1G6543oU	1000

Table 1: buyer table

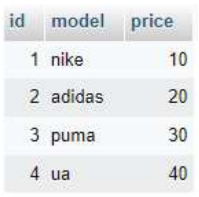
2. Manufacturers table



ID	Name	Address	Coins
1	syndicate	Gr89o2Kmsh	1000

Table 2: manufacturers table

3. Shoe table



id	model	price
1	nike	10
2	adidas	20
3	puma	30
4	ua	40

Table 3: shoe table

B. manufacturer.py

Our python file will first start listening for buyers who want to purchase using our program. When it receives a transaction request from the it will establish an acknowledgement first before the whole transaction of buying shoes can begin. The communication is encrypted with symmetric-key algorithm. Fig. 1 shows the communication process between the manufacturer and buyer.

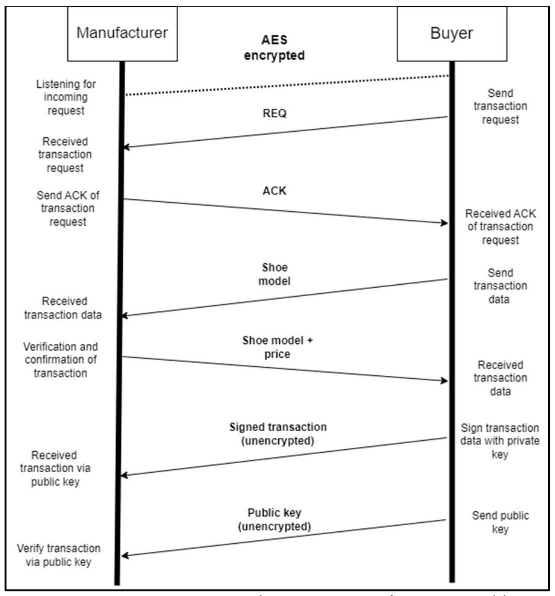

Manufacturer
AES encrypted
Buyer
Listening for incoming request
Send transaction request
REQ
Received transaction request
Send ACK of transaction request
ACK
Received ACK of transaction request
Shoe model
Send transaction data
Received transaction data
Verification and confirmation of transaction
Shoe model + price
Received transaction data
Signed transaction (unencrypted)
Sign transaction data with private key
Received transaction via public key
Public key (unencrypted)
Send public key
Verify transaction via public key

Fig. 1: Communication process between manufacturer and buyer

1. Encryption of communications channel using Advanced Encryption Standard (AES)

In our program, we have implemented encryption using AES for secure communication between the buyer and manufacturer. AES is a symmetric encryption algorithm, which utilises the same key for both the encryption and decryption of the communication. Data is encrypted using AES in Cipher Block Chaining (CBC) mode with the key and initialisation vector (IV). Before encryption in CBC mode, each block of plaintext is XORed together with the previous ciphertext block. Throughout the whole communication process, the data is divided into fixed-size blocks, each of these blocks is then encrypted and send over via socket connection. Upon receiving the encrypted data, the manufacturer will then decrypt the data using the same key.

C. buyer.py

In order to start sending a transaction request, the buyer will have to first enter their credentials, namely their username and password. In our project, the password is set to be the same as the corresponding username. The password will be hashed using the Secure Hash Algorithm 256 (SHA256) and stored in the database. Access into the system will only be given to users that exist in the buyer's database. As mentioned above, the transaction between buyer and manufacturer will be encrypted with symmetric-key algorithm AES.

1. Generation of Elliptic Curve Digital Signature Algorithm (ECDSA) key pair

Subsequently, the buyer generates an Elliptic Curve Digital Signature Algorithm (ECDSA) key pair using the SECP256k1 elliptic curve. The private key is saved to a file while the public key is propagated throughout the network by sending its *.pem* bytes over the socket. The SECP256k1 curve is a widely used curve in Bitcoin and other blockchain-based cryptocurrencies. The SECP256k1 curve is chosen because of its robustness, efficiency, and security properties.

The buyer's private key is generated using the *ecdsa.SigningKey.generate()* function, which generates a new random private key. The buyer's public key is derived from the private key using the *sk.get_verifying_key()* function. The private key is saved to a file named *buyer_private_key.pem*, and the public key is saved to a file named *buyer_public_key.pem*.

2. Public key distribution

The buyer's public key is then read from the *buyer_public_key.pem* file and distributed over the network using the *socket.sendall()* function. This public key is used by the manufacturer to verify the digital signature on the transaction data.

To sign the transaction data, the buyer uses the *sign()* method of the private key object to generate a digital signature. The *sign()* method takes the transaction data as input and returns a signature that is unique to the transaction data and the buyer's private key.

Finally, the signature is encoded in base64 format using the *base64.b64encode()* function. The encoded signature is sent to the manufacturer along with the transaction data, which allows the manufacturer to verify the authenticity of the transaction data using the buyer's public key.

3. Private key storage management

The buyer's private key is encrypted using the Password-Based Encryption Scheme 2 (PBES2). Firstly, the buyer is prompted to enter a password for the key. The password is subsequently encoded in bytes and used as the key to encrypt the private ECDSA key. The Password-Based Key Derivation Function 2 (PBKDF2) is utilised with the SHA256 hashing algorithm. PBKDF2 takes in the password, the salt, and an iteration count as inputs and returns an encryption key. The salt is randomly generated, and the iteration count is set to 100000 in order to resist any brute-force attacks. The private key is subsequently serialised into PEM-encoded format. The serialised private key is then encrypted using the encryption key and saved.

The encryption of the private key ensures improved security since it is protected by a password that only the buyer knows. The password is required for the decryption of the private key and use it for signing transactions. The encrypted private key can be safely stored on the buyer's computer or sent over a network without fear of exposing the private key to unauthorized parties.

D. miner.py

In our program, miners are a representative of any nodes in the blockchain network. They can mine blocks to partake in the blockchain and verify the validity of transactions and blocks in the blockchain. A *Block* class is created in this program. The class has several attributes such as the index, timestamp, data, previous hash, nonce, and hash. The index indicates the block position in the blockchain. The timestamps denote the time the block was added to the blockchain. The data attributes store the transaction details. The previous hash attribute stores the hash of the previous block. The nonce attribute indicates the number of attempts the miner executed to mine a block that conforms to the consensus algorithm. The hash attribute is the SHA256 hash of the block's index, timestamp, data, previous hash and nonce. The following is the *Block* class:

```
class Block:
    def __init__(self, index, timestamp, data,
previous_hash, nonce):
        self.index = index
        self.timestamp = timestamp
        self.data = data
        self.previous_hash = previous_hash
        self.nonce = nonce
        self.hash = self.calculate_hash()


    def calculate_hash(self):
        sha = hashlib.sha256()
        sha.update(str(self.index).encode('utf-8') +
                   str(self.timestamp).encode('utf-8')
+
                   str(self.data).encode('utf-8') +
                   str(self.previous_hash).encode('utf-
8') +
                   str(self.nonce).encode('utf-8'))
        return sha.hexdigest()
```

1. Creation of Genesis Block

The genesis block is the very first block in a blockchain. The genesis block acts as a point of reference for all subsequent blocks, therefore it must first be created before any mining of blocks can commence. In addition, since every block in the blockchain will consist of the hash of the previous block, a chain of blocks will be led back to the genesis block. This plays a crucial role in the security of the blockchain, as any attempt to modify or mimic the genesis block will require a large amount of computational power and will be easily detectable.

2. Consensus Algorithm

The consensus algorithm is a mechanism used in a blockchain to ensure that all nodes in the network agree on the contents of the blockchain. Miners will compete to solve a cryptographic puzzle using their computational resources. The first miner to solve the puzzle is usually rewarded with a block reward and the new block generated will be added to the blockchain. The difficulty of the cryptographic puzzle is as and when adjusted, so that the rate of new blocks mined can be controlled, ensuring new blocks are added to the blockchain at a steady rate. Some popular consensus algorithms include the Proof of Work (PoW) algorithm used in the Bitcoin network and the Proof of Stake (PoS) algorithm used in the Ethereum network. In our program, we have implemented the Proof of Work (PoW) algorithm.

The *mine_block()* function generates a new block by performing a brute-force search for a nonce value. The hash of the nonce value combined with the other block data will need to have a specified number of leading zeros (i.e., the difficulty level). The difficulty parameter determines the number of leading zeros required for the hash to be considered valid.

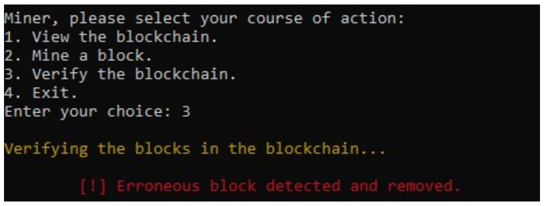
Before mining a new block, the function reads the previous block's hash value from the blockchain. Next, it will append the signed transaction data and creates a new block with the current timestamp, previous hash, and nonce. The block's hash is computed by calling the *calculate_hash()* method of the Block class. The while loop continues to increment the nonce value until the block's hash meets the specified difficulty level. Once a valid block is found, the function returns the block object.

The PoW consensus algorithm is an important feature of blockchain networks because it ensures that each new block added to the chain is computationally expensive to produce, making it difficult for malicious actors to modify the blockchain's history. By requiring nodes to compete to solve a computational problem, PoW creates a trustless and decentralized system that is resilient to attacks.

3. Verification of new transaction

The program checks if the buyer has sufficient coins for the transaction by comparing the UTXO (unspent transaction output) and transaction value. When a transaction is received, the nodes on the network will check if the sender has sufficient UTXOs to cover the transaction value. If there are enough UTXOs, the transaction is deemed valid. If there are not enough UTXOs, the transaction is rejected and the transaction discarded. If the buyer has sufficient coins, the code verifies the transaction signature using the buyer's public key. If the signature is verified successfully, the transaction is conducted with coins being transacted in the process between the buyer and manufacturer. Thereafter, the new block consisting of the transaction detail is added to the blockchain network.

The implementation of the *verify_blockchain()* function will verify the integrity of the blockchain. The value of the previous hash attribute in a block will be compared with the hash value of the previous block. In the event where the hash values are mismatched, this means that the more recent block might be a counterfeit. From Fig. 2, the program will proceed to remove the block from the blockchain. This will ensure the integrity and security of the blockchain network.

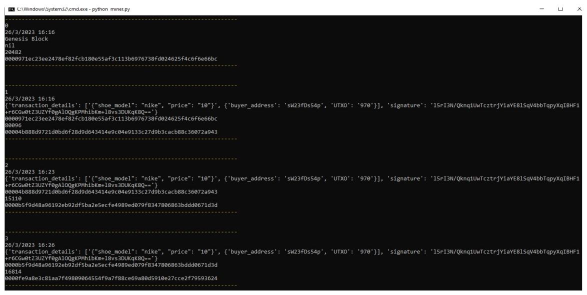


```
Miner, please select your course of action:
1. View the blockchain.
2. Mine a block.
3. Verify the blockchain.
4. Exit.
Enter your choice: 3

Verifying the blocks in the blockchain...

        [!] Erroneous block detected and removed.
```

Fig. 2: Removal of erroneous block to ensure integrity

4. View blockchain

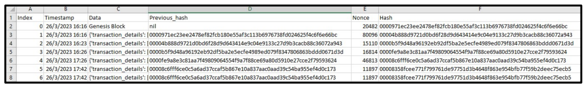
There is a function for users to view the entire blockchain in in the command line interface (CLI), seen in Fig. 3.



```
C:\Windows\System32\cmd.exe - python miner.py
----------------------------------------------------------------------
0
26/3/2023 16:16
Genesis Block
nil
20482
0000971ec23ee2478ef82fcb180e55af3c113b6976738fd024625f4c6f6e66bc
----------------------------------------------------------------------


----------------------------------------------------------------------
1
26/3/2023 16:16
{'transaction_details': ['{"shoe_model": "nike", "price": "10"}', {'buyer_address': 'sW23fDs54p', 'UTXO': '970'}], 'signature': 'l5rI3N/Qknq1UwTcztrjYiaYE8lSqV4bbTqpyXqIBHF1
+r6CGw0tZ3UZYf0gAlOQgKPMhibKm+l8vs3DUKqK8Q=='}
0000971ec23ee2478ef82fcb180e55af3c113b6976738fd024625f4c6f6e66bc
80096
00004b888d9721d0bd6f28d9d643414e9c04e9133c27d9b3cacb88c36072a943
----------------------------------------------------------------------


----------------------------------------------------------------------
2
26/3/2023 16:23
{'transaction_details': ['{"shoe_model": "nike", "price": "10"}', {'buyer_address': 'sW23fDs54p', 'UTXO': '970'}], 'signature': 'l5rI3N/Qknq1UwTcztrjYiaYE8lSqV4bbTqpyXqIBHF1
+r6CGw0tZ3UZYf0gAlOQgKPMhibKm+l8vs3DUKqK8Q=='}
00004b888d9721d0bd6f28d9d643414e9c04e9133c27d9b3cacb88c36072a943
15110
0000b5f9d48a96192eb92df5ba2e5ecfe4989ed079f8347806863bddd0671d3d
----------------------------------------------------------------------


----------------------------------------------------------------------
3
26/3/2023 16:26
{'transaction_details': ['{"shoe_model": "nike", "price": "10"}', {'buyer_address': 'sW23fDs54p', 'UTXO': '970'}], 'signature': 'l5rI3N/Qknq1UwTcztrjYiaYE8lSqV4bbTqpyXqIBHF1
+r6CGw0tZ3UZYf0gAlOQgKPMhibKm+l8vs3DUKqK8Q=='}
0000b5f9d48a96192eb92df5ba2e5ecfe4989ed079f8347806863bddd0671d3d
16814
0000fe9a8e3c81aa7f49809064554f9a7f88ce69a80d5910e27cce2f79593624
----------------------------------------------------------------------
```

Fig. 3: View blockchain in CLI

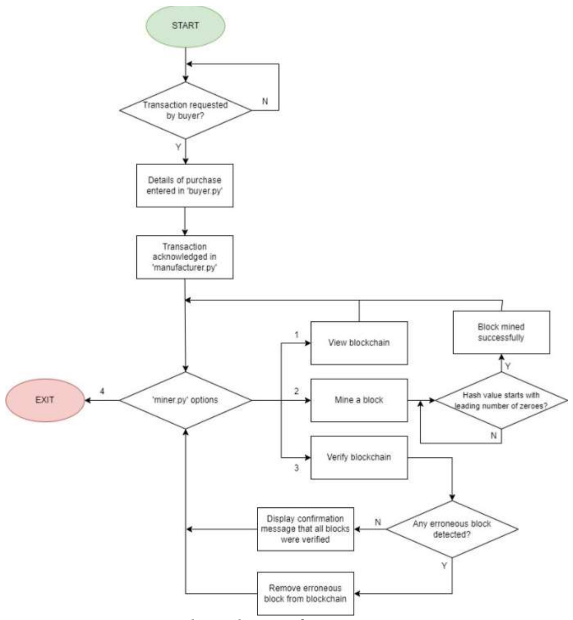
Users can also view the entire blockchain in an excel, seen in Fig. 4. This provides better clarity and readability as compared to viewing the blockchain in CLI.



	A	B	C	D	E	F
1	Index	Timestamp	Data	Previous_hash	Nonce	Hash
2	0	26/3/2023 16:16	Genesis Block	nil	20482	0000971ec23ee2478ef82fcb180e55af3c113b6976738fd024625f4c6f6e66bc
3	1	26/3/2023 16:16	{'transaction_details': [	0000971ec23ee2478ef82fcb180e55af3c113b6976738fd024625f4c6f6e66bc	80096	00004b888d9721d0bd6f28d9d643414e9c04e9133c27d9b3cacb88c36072a943
4	2	26/3/2023 16:23	{'transaction_details': [	00004b888d9721d0bd6f28d9d643414e9c04e9133c27d9b3cacb88c36072a943	15110	0000b5f9d48a96192eb92df5ba2e5ecfe4989ed079f8347806863bddd0671d3d
5	3	26/3/2023 16:26	{'transaction_details': [	0000b5f9d48a96192eb92df5ba2e5ecfe4989ed079f8347806863bddd0671d3d	16814	0000fe9a8e3c81aa7f49809064554f9a7f88ce69a80d5910e27cce2f79593624
6	4	26/3/2023 17:26	{'transaction_details': [	0000fe9a8e3c81aa7f49809064554f9a7f88ce69a80d5910e27cce2f79593624	46813	00008c6fff6ce0c5a6ad37ccaf5b867e10a837aac0aad39c54ba955ef4d0c173
7	5	26/3/2023 17:42	{'transaction_details': [	00008c6fff6ce0c5a6ad37ccaf5b867e10a837aac0aad39c54ba955ef4d0c173	11897	00008358fcee771f799761de97751d3b4648f863e954bfb77f59b2deec75ecb5
8	6	26/3/2023 17:42	{'transaction_details': [	00008c6fff6ce0c5a6ad37ccaf5b867e10a837aac0aad39c54ba955ef4d0c173	11897	00008358fcee771f799761de97751d3b4648f863e954bfb77f59b2deec75ecb5

Fig. 4: View blockchain in csv file

To enhance clarity and ease of understanding, a simplified flowchart depicting the program's sequence is provided below, considering that executing several python scripts could be complicated to comprehend. This flowchart, seen in Fig. 5, is based on the condition that the transaction was deemed valid.



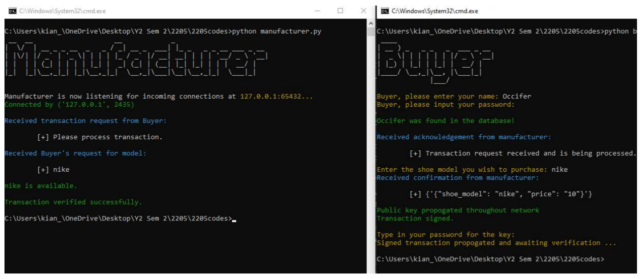
START
Transaction requested by buyer?
N
Y
Details of purchase entered in 'buyer.py'
Transaction acknowledged in 'manufacturer.py'
1
View blockchain
Block mined successfully
Y
EXIT
4
'miner.py' options
2
Mine a block
Hash value starts with leading number of zeroes?
N
Verify blockchain
3
Display confirmation message that all blocks were verified
N
Any erroneous block detected?
Y
Remove erroneous block from blockchain

Fig. 5: Flowchart of our program

IV. Results and Analysis

As we have three python files, it is recommended to run manufacturer.py first, followed by buyer.py and miner.py. This is analogous to how a store owner would first open their store to allow customers to enter and make purchases.

A. Communication between Manufacturer and Buyer


```
C:\Windows\System32\cmd.exe
C:\Users\kian_\OneDrive\Desktop\Y2 Sem 2\2205\2205codes>python manufacturer.py

Manufacturer is now listening for incoming connections at 127.0.0.1:65432...
Connected by ('127.0.0.1', 2435)

Received transaction request from Buyer:

        [+] Please process transaction.

Received Buyer's request for model:

        [+] nike

nike is available.

Transaction verified successfully.

C:\Users\kian_\OneDrive\Desktop\Y2 Sem 2\2205\2205codes>
```

```
C:\Windows\System32\cmd.exe
C:\Users\kian_\OneDrive\Desktop\Y2 Sem 2\2205\2205codes>python b

Buyer, please enter your name: Occifer
Buyer, please input your password:

Occifer was found in the database!

Received acknowledgement from manufacturer:

        [+] Transaction request received and is being processed.

Enter the shoe model you wish to purchase: nike
Received confirmation from manufacturer:

        [+] {'{"shoe_model": "nike", "price": "10"}'}

Public key propogated throughout network
Transaction signed.

Type in your password for the key:
Signed transaction propogated and awaiting verification ...

C:\Users\kian_\OneDrive\Desktop\Y2 Sem 2\2205\2205codes>
```

Fig. 6: manufacturer.py and buyer.py screens upon transaction completion

In our example, a buyer named 'Occifer' wants to buy a shoe. As Occifer's details have already been saved in our MySQL database, Occifer just needs to enter his name and password, which is also Occifer (the terminal masks the password for privacy). While Occifer is entering their credentials, the manufacturer.py terminal has already started running and is waiting for connections at the local host. Upon Occifer logging in, a blue confirmation message is shown to the manufactuer.py terminal, and the manufacturer only knows that a random buyer has started a possible transaction without knowing the buyer's name, similar to how a store owner is informed that a random customer has walked into the store.

On the buyer.py screen, Occifer chooses to buy a Nike shoe, and confirms the purchase. The manufacturer is then updated with the shoe brand which the buyer desires to purchase, and verifies the transaction. Doing this will successfully complete the transaction and the function ends for both manufacturer.py and buyer.py. The transaction will thereafter we verified and appended to newly mined blocks which will then be added into the blockchain.

B. Miner.py

Upon running *miner.py*, the user will be presented with a list of options to undertake, as seen in Fig. 7.

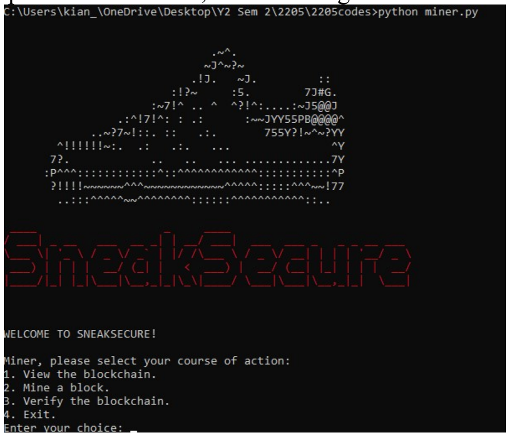

```
C:\Users\kian_\OneDrive\Desktop\Y2 Sem 2\2205\2205codes>python miner.py

WELCOME TO SNEAKSECURE!

Miner, please select your course of action:
1. View the blockchain.
2. Mine a block.
3. Verify the blockchain.
4. Exit.
Enter your choice:
```

Fig. 7: Miner's menu page

When users select the second option to mine a block, the program will start mining based on the consensus algorithm established. As seen in Fig. 8, upon successfully mining a block, the transaction will take place instantaneously.

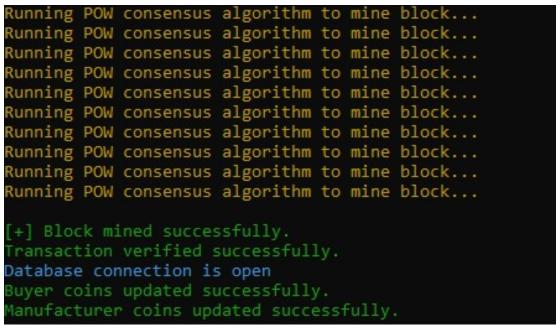
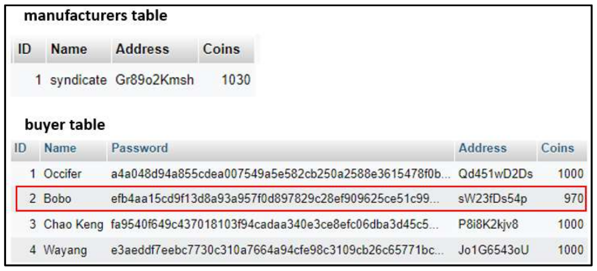

```
Running POW consensus algorithm to mine block...
Running POW consensus algorithm to mine block...
Running POW consensus algorithm to mine block...
Running POW consensus algorithm to mine block...
Running POW consensus algorithm to mine block...
Running POW consensus algorithm to mine block...
Running POW consensus algorithm to mine block...
Running POW consensus algorithm to mine block...
Running POW consensus algorithm to mine block...
Running POW consensus algorithm to mine block...

[+] Block mined successfully.
Transaction verified successfully.
Database connection is open
Buyer coins updated successfully.
Manufacturer coins updated successfully.
```

Fig. 8: Block mined successfully, and transaction carried out.

As seen in Fig. 9, the coins amount of manufacturer and buyer are updated in the SQL database.


manufacturers table

ID	Name	Address	Coins
1	syndicate	Gr89o2Kmsh	1030

buyer table

ID	Name	Password	Address	Coins
1	Occifer	a4a048d94a855cdea007549a5e582cb250a2588e3615478f0b...	Qd451wD2Ds	1000
2	Bobo	efb4aa15cd9f13d8a93a957f0d897829c28ef909625ce51c99...	sW23fDs54p	970
3	Chao Keng	fa9540f649c437018103f94cadaa340e3ce8efc06dba3d45c5...	P8i8K2kjv8	1000
4	Wayang	e3aeddf7eebc7730c310a7664a94cfe98c3109cb26c65771bc...	Jo1G6543oU	1000

Fig. 9: Transaction carried out and coins value updated in SQL database

C. Wireshark captures

As seen in Fig. 10, the data sent from the manufacturer to the buyer is encrypted.

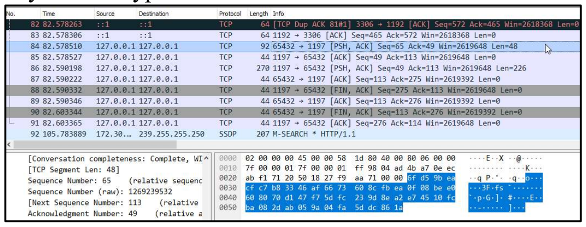

```
No.  Time        Source     Destination     Protocol Length Info
  82 82.578263   ::1        ::1             TCP       64 [TCP Dup ACK 81#1] 3306 → 1192 [ACK] Seq=572 Ack=465 Win=2618368 Len=0
  83 82.578306   ::1        ::1             TCP       64 1192 → 3306 [ACK] Seq=465 Ack=572 Win=2618368 Len=0
  84 82.578510   127.0.0.1 127.0.0.1        TCP       92 65432 → 1197 [PSH, ACK] Seq=65 Ack=49 Win=2619648 Len=48
  85 82.578527   127.0.0.1 127.0.0.1        TCP       44 1197 → 65432 [ACK] Seq=49 Ack=113 Win=2619648 Len=0
  86 82.590198   127.0.0.1 127.0.0.1        TCP      270 1197 → 65432 [PSH, ACK] Seq=49 Ack=113 Win=2619648 Len=226
  87 82.590222   127.0.0.1 127.0.0.1        TCP       44 65432 → 1197 [ACK] Seq=113 Ack=275 Win=2619392 Len=0
  88 82.590332   127.0.0.1 127.0.0.1        TCP       44 1197 → 65432 [FIN, ACK] Seq=275 Ack=113 Win=2619648 Len=0
  89 82.590346   127.0.0.1 127.0.0.1        TCP       44 65432 → 1197 [ACK] Seq=113 Ack=276 Win=2619392 Len=0
  90 82.603344   127.0.0.1 127.0.0.1        TCP       44 65432 → 1197 [FIN, ACK] Seq=113 Ack=276 Win=2619392 Len=0
  91 82.603365   127.0.0.1 127.0.0.1        TCP       44 1197 → 65432 [ACK] Seq=276 Ack=114 Win=2619648 Len=0
  92 105.783889  172.30…   239.255.255.250  SSDP     207 M-SEARCH * HTTP/1.1

[Conversation completeness: Complete, WI     0000  02 00 00 00 45 00 00 58  1d 80 40 00 80 06 00 00   ····E··X ··@·····
[TCP Segment Len: 48]                        0010  7f 00 00 01 7f 00 00 01  ff 98 04 ad 4b a7 0e ec   ········ ····K···
Sequence Number: 65    (relative sequenc     0020  ab f1 71 20 50 18 27 f9  aa 71 00 00 6f d5 9b ea   ··q P·'· ·q··o···
Sequence Number (raw): 1269239532            0030  cf c7 b8 33 46 af 66 73  60 8c fb ea 0f 08 be e0   ···3F·fs `·······
[Next Sequence Number: 113    (relative      0040  60 80 70 d1 47 f7 5d fc  23 9d 8e a2 e7 45 10 fc   `·p·G·]· #····E··
Acknowledgment Number: 49    (relative a     0050  ba 08 2d ab 05 9a 04 fa  5d dc 86 1a               ········ ]···
```

Fig. 10: *Encrypted* data sent from the manufacturer to the buyer

Similarly, as seen in Fig. 11, the data sent from the buyer to the manufacturer is also encrypted.

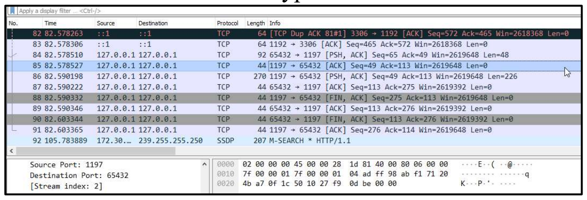
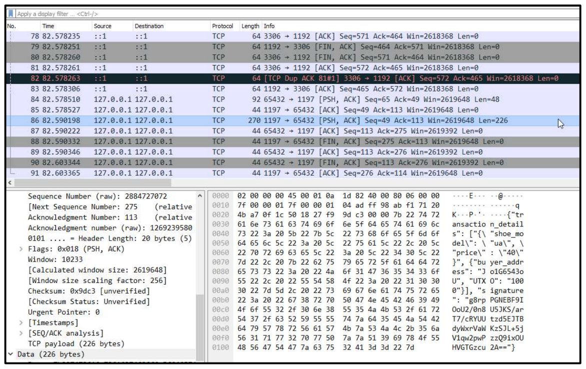

```
Apply a display filter ... <Ctrl-/>
No.  Time        Source     Destination     Protocol Length Info
  82 82.578263   ::1        ::1             TCP       64 [TCP Dup ACK 81#1] 3306 → 1192 [ACK] Seq=572 Ack=465 Win=2618368 Len=0
  83 82.578306   ::1        ::1             TCP       64 1192 → 3306 [ACK] Seq=465 Ack=572 Win=2618368 Len=0
  84 82.578510   127.0.0.1 127.0.0.1        TCP       92 65432 → 1197 [PSH, ACK] Seq=65 Ack=49 Win=2619648 Len=48
  85 82.578527   127.0.0.1 127.0.0.1        TCP       44 1197 → 65432 [ACK] Seq=49 Ack=113 Win=2619648 Len=0
  86 82.590198   127.0.0.1 127.0.0.1        TCP      270 1197 → 65432 [PSH, ACK] Seq=49 Ack=113 Win=2619648 Len=226
  87 82.590222   127.0.0.1 127.0.0.1        TCP       44 65432 → 1197 [ACK] Seq=113 Ack=275 Win=2619392 Len=0
  88 82.590332   127.0.0.1 127.0.0.1        TCP       44 1197 → 65432 [FIN, ACK] Seq=275 Ack=113 Win=2619648 Len=0
  89 82.590346   127.0.0.1 127.0.0.1        TCP       44 65432 → 1197 [ACK] Seq=113 Ack=276 Win=2619392 Len=0
  90 82.603344   127.0.0.1 127.0.0.1        TCP       44 65432 → 1197 [FIN, ACK] Seq=113 Ack=276 Win=2619392 Len=0
  91 82.603365   127.0.0.1 127.0.0.1        TCP       44 1197 → 65432 [ACK] Seq=276 Ack=114 Win=2619648 Len=0
  92 105.783889  172.30…   239.255.255.250  SSDP     207 M-SEARCH * HTTP/1.1

Source Port: 1197              0000  02 00 00 00 45 00 00 28  1d 81 40 00 80 06 00 00   ····E··( ··@·····
Destination Port: 65432        0010  7f 00 00 01 7f 00 00 01  04 ad ff 98 ab f1 71 20   ········ ······q
[Stream index: 2]              0020  4b a7 0f 1c 50 10 27 f9  0d be 00 00               K···P·'· ····
```

Fig. 11: *Encrypted* data sent from the buyer to the manufacturer

As seen in Fig. 12, the transaction details sent from the buyer to the manufacturer are visible in Wireshark as TCP packets. Transaction details are not encrypted in blockchain because they need to be transparent and visible to all participants in the network.



```
Apply a display filter ... <Ctrl-/>
No.  Time       Source    Destination Protocol Length Info
  78 82.578235  ::1       ::1         TCP       64 3306 → 1192 [ACK] Seq=571 Ack=464 Win=2618368 Len=0
  79 82.578251  ::1       ::1         TCP       64 1192 → 3306 [FIN, ACK] Seq=464 Ack=571 Win=2618368 Len=0
  80 82.578260  ::1       ::1         TCP       64 3306 → 1192 [FIN, ACK] Seq=571 Ack=464 Win=2618368 Len=0
  81 82.578261  ::1       ::1         TCP       64 3306 → 1192 [ACK] Seq=572 Ack=465 Win=2618368 Len=0
  82 82.578263  ::1       ::1         TCP       64 [TCP Dup ACK 81#1] 3306 → 1192 [ACK] Seq=572 Ack=465 Win=2618368 Len=0
  83 82.578306  ::1       ::1         TCP       64 1192 → 3306 [ACK] Seq=465 Ack=572 Win=2618368 Len=0
  84 82.578510  127.0.0.1 127.0.0.1   TCP       92 65432 → 1197 [PSH, ACK] Seq=65 Ack=49 Win=2619648 Len=48
  85 82.578527  127.0.0.1 127.0.0.1   TCP       44 1197 → 65432 [ACK] Seq=49 Ack=113 Win=2619648 Len=0
  86 82.590198  127.0.0.1 127.0.0.1   TCP      270 1197 → 65432 [PSH, ACK] Seq=49 Ack=113 Win=2619648 Len=226
  87 82.590222  127.0.0.1 127.0.0.1   TCP       44 65432 → 1197 [ACK] Seq=113 Ack=275 Win=2619392 Len=0
  88 82.590332  127.0.0.1 127.0.0.1   TCP       44 1197 → 65432 [FIN, ACK] Seq=275 Ack=113 Win=2619648 Len=0
  89 82.590346  127.0.0.1 127.0.0.1   TCP       44 65432 → 1197 [ACK] Seq=113 Ack=276 Win=2619392 Len=0
  90 82.603344  127.0.0.1 127.0.0.1   TCP       44 65432 → 1197 [FIN, ACK] Seq=113 Ack=276 Win=2619392 Len=0
  91 82.603365  127.0.0.1 127.0.0.1   TCP       44 1197 → 65432 [ACK] Seq=276 Ack=114 Win=2619648 Len=0
```

```
    Sequence Number (raw): 2884727072
    [Next Sequence Number: 275    (relative
    Acknowledgment Number: 113    (relative
    Acknowledgment number (raw): 1269239580
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 10233
    [Calculated window size: 2619648]
    [Window size scaling factor: 256]
    Checksum: 0x9dc3 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
    TCP payload (226 bytes)
v Data (226 bytes)
```

```
0000  02 00 00 00 45 00 01 0a  1d 82 40 00 80 06 00 00   ····E··· ··@·····
0010  7f 00 00 01 7f 00 00 01  04 ad ff 98 ab f1 71 20   ········ ······q
0020  4b a7 0f 1c 50 18 27 f9  9d c3 00 00 7b 22 74 72   K···P·'· ····{"tr
0030  61 6e 73 61 63 74 69 6f  6e 5f 64 65 74 61 69 6c   ansactio n_detail
0040  73 22 3a 20 5b 22 7b 5c  22 73 68 6f 65 5f 6d 6f   s": ["{\ "shoe_mo
0050  64 65 6c 5c 22 3a 20 5c  22 75 61 5c 22 2c 20 5c   del\": \ "ua\", \
0060  22 70 72 69 63 65 5c 22  3a 20 5c 22 34 30 5c 22   "price\" : \"40\"
0070  7d 22 2c 20 7b 22 62 75  79 65 72 5f 61 64 64 72   }", {"bu yer_addr
0080  65 73 73 22 3a 20 22 4a  6f 31 47 36 35 34 33 6f   ess": "J o1G6543o
0090  55 22 2c 20 22 55 54 58  4f 22 3a 20 22 31 30 30   U", "UTX O": "100
00a0  30 22 7d 5d 2c 20 22 73  69 67 6e 61 74 75 72 65   0"}], "s ignature
00b0  22 3a 20 22 67 38 72 70  50 47 4e 45 42 46 39 49   ": "g8rp PGNEBF9I
00c0  4f 6f 55 32 2f 30 6e 38  55 35 4a 4b 53 2f 61 72   OoU2/0n8 U5JKS/ar
00d0  54 37 2f 63 52 59 55 55  74 7a 64 35 45 4a 54 42   T7/cRYUU tzd5EJTB
00e0  64 79 57 78 72 56 61 57  4b 7a 53 4a 4c 2b 35 6a   dyWxrVaW KzSJL+5j
00f0  56 31 71 77 32 70 77 50  7a 7a 51 39 69 78 4f 55   V1qw2pwP zzQ9ixOU
0100  48 56 47 54 47 7a 63 75  32 41 3d 3d 22 7d         HVGTGzcu 2A=="}
```

Fig. 12: TCP packet capture of transaction in Wireshark

The transparency of the blockchain is one of its most key features, as it allows all parties to verify and validate the transactions taking place on the network without the need for a trusted third party. All transactions must be visible to everyone in the network to ensure that the blockchain remains decentralized and secure. This transparency also helps to prevent fraud and corruption by allowing all participants to see the details of all transactions. However, while transaction details are visible to all publicly, the buyer can remain anonymous as only their blockchain address is included in the transaction. This feature helps to ensure the privacy of the buyer while still allowing for transparency and accountability in the blockchain.

V. COMPILING AND RUNNING THE CODE

Listed below are the steps to compile and run our program:

1. Download the zip file from LMS.
2. Unzip the sneaksecure-BlockChain file to get to its folder.
3. *cd sneaksecure-BlockChain Folder*
4. *pip install –r requirements.txt*
5. python {manufactuer.py | buyer.py | miner.py}

The above-mentioned steps will ensure that all your dependencies are downloaded to your environment. In the sneaksecure_BlockChain Folder, there are 3 executable python files. They are *manufacturer.py*, *buyer.py* and *miner.py*.

VI. CONCLUSION

In comparison to other solutions, our proposed solution offers a more comprehensive and flexible simulation of a transaction between a manufacturer and a buyer. Our program utilizes three python files to manage the different actors involved in the transaction and a MySQL database to store data, providing a scalable and customizable platform for simulating transactions. Additionally, our solution allows for the configuration of the number of buyers and manufacturers, as well as the initial number of coins available to each, making it more adaptable to different scenarios. By allowing buyers to purchase shoes from four different brands at varying prices, our solution also provides a more realistic simulation of real-world transactions. Overall, our proposed solution stands out from other solutions in terms of its functionality, flexibility, and realism, making it a more viable option for simulating transactions between a manufacturer and a buyer.

ACKNOWLEDGMENT

ICT2205 Group 13 wishes to acknowledge Professors Spiridon Bakiras and Woo Wing Keong for their unwavering help and guidance throughout the lectures and labs which enhanced our knowledge about Applied Cryptography and allowed us to understand how and why this module is vital for our future knowledge in our course of study.

REFERENCES

[1] Z. Zheng, S. Xie, H. Dai, X. Chen, H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," [Online]. Available: https://www.researchgate.net/publication/318131748_An_Overview_of_Blockchain_Technology_Architecture_Consensus_and_Future_Trends. [Accessed: 22-Mar-2023].

[2] K. Christidis, M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7467408. [Accessed: 22-Mar-2023].

[3] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed: 25-Mar-2023].

[4] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," [Online]. Available: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf. [Accessed: 25-Mar-2023].

[5] Ethereum, "Welcome to Ethereum," [Online]. Available: https://ethereum.org/en/. [Accessed: 25-Mar-2023].