Blockchain for Provenance in a Repo to Auction System

Mason Harlan

CS 491

Prof. Badsha

12/2/2021

Abstract

In the current state of the repossession to auction market, there is a lack of provenance and accountability from the repossession to the sale. In this paper, I explore an application of blockchain in order to create provenance in a repo to auction system that maintains the integrity of the transactions and operations performed on a vehicle in the system. The system uses python to demonstrate some key features such as consensus and traceability. Files for the simulation can be found at (to request access email harlangmason@gmail.com):

https://github.com/mgharlan/project491

1 Introduction

The repossession industry in the United States recovers about 1.5 million vehicles every year. These cars often end up actioned off to car dealers or to the public. Knowing what actions have been taken on the car between recovery and auction is critically important to all parties involved in the repo to auction process. Car buyers are also interested in a vehicles history report and all of the operations that have taken place on a car (sundevilauto). In the current state of the market companies often work together to smooth out the process. Repossession companies partner with banks, locksmiths, mechanics, auctions, and transport companies in order to prepare the car for a sale. Some companies, such as KAR Global, are vertically integrated and own business units in all of these areas. These business units, all owned by the same parent company, do not have internal communication. As a result, it can be difficult and unreliable to figure out what operations have taken place on a car. For example, if a recovery company sends the car to locksmith company to get new keys cut and then to an auction company for the final sale and the auction company wants to know who cut the keys it would be difficult to obtain this information. This is where the proposed blockchain system comes in. A private blockchain system centered around provenance and data integrity would allow any users in the system to view what actions have been taken on the car and when. The transaction record would be validated before being written into the blockchain and could reliably inform the auction company who cut the keys for the car. This smooths out tension points between different business units and allows for inter-business unit communication. More information and transparency can significantly improve business operations and provide valuable insight. Similarly, future iterations could allow customers to view a report of a car that they are interested in purchasing which provides transparency and creates provenance.

2 System Description

The proposed system is simulated in four main components: Nodes, Blocks, the Blockchain, and a Transaction Pool. Each component is used by the main driver of the simulation to demonstrate how provenance is created in a repo to auction system.

Nodes in the system are represented by an abstract base class that must implement a name, an id, mining status, and an operation. The name and id correspond to the company that the node represents. The mining status attribute is a Boolean that is True if the node can perform mining and is false if it cannot. There is a Transaction Pool that is shared between all of the nodes in the network. The transaction pool will be discussed in more detail later.

```
class Node(ABC):
  transactionPool = TransactionPool()

  def __init__(self, id = 0, name = ''):
    self.id = id
    self.name = name
    print(f"{type(self).__name__} node created: {self.name}({self.id})")

  @property
  @abstractmethod
  def miner(self):
    pass

  @abstractmethod
  def performOperation(self):
    pass
```

In this system, there are five node types that implement the abstract base class: Recovery, Auction, Locksmith, Transport, and Mechanic. These node types correspond to the types of companies demonstrated in the simulation. Here is one example of these nodes (the other nodes look similar):

```
class Recovery(Node):
  miner = True

  def performOperation(self, VIN):
    Node.transactionPool.add(f"{Operation.RECOVER.value}|{VIN}|{self.id}")
    print(f"{self.name}({self.id}) recovered VIN: {VIN}")
```

The different node types all perform operations based on their company types. The following enumerations describe the company types and there corresponding operations:

```
class Operation(Enum):
  RECOVER = 1
  TRANSPORT = 2
  REPAIR = 3
  CUT_KEYS = 4
  AUCTION = 5

class Company(Enum):
  RECOVERY = 1
  TRANSPORT = 2
  MECHANIC = 3
  LOCKSMITH = 4
  AUCTION = 5
```

As mentioned above, the nodes submit their performed operations to the transaction pool which stores them until they are ready to be validated, mined, and add to the blockchain. The Transaction Pool is shared across all nodes and implements two important methods: add and pop. Add appends a transaction to the transaction_pool list and pop removes and returns the oldest transaction from the list. The Transaction Pool is essentially a FIFO queue. The transaction pool is needed to help synchronize the mining and consensus processes by providing a centralized place for pending transactions to be stored. Transactions have the general form of “operation | VIN | ID of performer | additional info1 | additional info 2| …”

```
class TransactionPool:
  transaction_pool = []
  is_empty = True

  def add(self, transaction : str) -> None:
    self.is_empty = False
    self.transaction_pool.append(transaction)
    # print('add: ', self.transaction_pool)

  def pop(self) -> str:
    if(not self.is_empty):
      value = self.transaction_pool.pop(0)
      if(len(self.transaction_pool) == 0):
        self.is_empty = True

      # print('pop: ', self.transaction_pool)
      return value
    else:
      # print('pop: ', self.transaction_pool)
      return ''
```

Blocks are the essential unit for the blockchain. Each block in the system stores information about the transaction that the block contains such as what the transaction was and when the transaction took place. The block also holds information about the consensus process and its position in the Blockchain. This information includes the previous block's hash, the nonce, the miner who validated the block, and the hash of the block.

```
class Block():
  def __init__(self, data, timestamp, previous_hash):
    self.data = data
    self.timestamp = timestamp
    self.previous_hash = previous_hash
    self.nonce = 0
    self.miner = None
    self.hash_block()

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

Finally, the Blockchain component was created by inheriting from the python dictionary class. This implementation has several advantages over the python list implementation. For example, a dictionary approach allows the system to look up a transaction by its hash by simply calling blockchain[hash]. This has a lookup time of O(1) and is much faster than the potential O(N) lookup time that comes from scanning through a list looking for a hash value. Additionally, accessing the blocks by hash makes the program logic simpler by allowing the user to look up a previous block based on the current block’s previous hash record. This removes any need for calculating indexes.

```
class Blockchain(dict):
  currentBlock = None

  def __init__(self):
    dict.__init__(self)
    self['0'] = self.create_genesis_block()
    self.currentBlock = '0'

  # Generate genesis block
  def create_genesis_block(self):
    # Manually construct a block with index zero and arbitrary previous hash
    genesis = Block("", date.datetime.now(), "")
    genesis.hash = '0'
    return genesis
```

From above, the genesis block contains an empty string as the transaction value and for the previous hash. The hash value of the genesis block is zero. The Blockchain has some helper functions to make working with the blocks easier. The getLastBlockWithVin function returns the most recently recorded block with the VIN being queried. The tip function returns the most recent block on the blockchain and the add block adds a new block to the blockchain. The writeToFile function writes the entire blockchain to a file for parsing and/or storage later on.

```
  def getLastBlockWithVin(self, VIN):
    block = self.tip()
    while(block.hash != '0'):
      if(block.data.split('|')[1] == VIN):
        return block
      else:
        block = self[block.previous_hash]
    return None

  def tip(self):
    if(self.currentBlock != None):
      return self[self.currentBlock]
    else:
      return None

  def add(self, newBlock):
    self[newBlock.hash] = newBlock
    self.currentBlock = newBlock.hash

  def writeToFile(self):
    file = open("blockchain.txt", 'w')
    block = self.tip()
    while block.hash != '0':
      file.write("Block Data: " + str(block.data) + "\n")
      file.write("Hash: " + str(block.hash) + "\n")
      file.write("Miner Who Discovered Block: " + str(block.miner) + "\n")
      file.write("--------------------------------\n")
      block = self[block.previous_hash]
    file.write("Genesis Block \n")
    file.write("Hash: 0 \n")
    file.write("Miner Who Discovered Block: NA \n")
    file.write("----------------------------------\n")

    file.close()
```

Next the consensus process will be described using the components from above. Periodically, nodes that have the mining privilege will pop transactions from the mining pool. The transactions are validated by the mining nodes. If the operation is a recovery, then this means the VIN is being added to the system and no validation is necessary. Otherwise, the mining node checks to make sure that the VIN is in the possession of the company performing the transaction in question. This can be achieved by making sure the last transaction was performed by that company or if the last transaction was a transport to that company. Once this information has been validated, the proof of work puzzle begins. If the validation fails, mining does not begin, and the block is rejected. In proof of work, each mining enabled node gets a chance to solve the hash problem. After each attempt the nonce is increased. The difficulty of the problem is set by the global DIFFICULTY_HASH. If the mining node is successful, the block is added to the chain and the miner who successfully mined the block is marked on that block.

```
def runMining(self):
    random.shuffle(self.miners)
    while(not Node.transactionPool.is_empty):
      newBlock = Block(Node.transactionPool.pop(), date.datetime.now(),
self.blockchain.tip().hash)
      miner = None
      accepted = False

      #check if transaction is valid before mining
      dataList = newBlock.data.split('|')
      lastBlockWithVin = self.blockchain.getLastBlockWithVin(dataList[1])
      if(Operation(int(dataList[0])) == Operation.RECOVER):
        accepted = True
      elif(Operation(int(dataList[0])) == Operation.TRANSPORT):
        if(dataList[3] == lastBlockWithVin.data.split('|')[-1]):
          accepted = True
      else:
        if(lastBlockWithVin != None and dataList[2] ==
lastBlockWithVin.data.split('|')[-1]):
          accepted = True

      #if transaction is valid
      if(accepted):
        # While the hash is bigger than or equal to the difficulty continue
to iterate the nonce
        while int(newBlock.hash_block(), 16) >= DIFFICULTY_HASH:
          newBlock.nonce += 1
          miner = self.miners[newBlock.nonce % len(self.miners)]
          newBlock.miner = miner
        self.blockchain.add(newBlock)
      else:
        print(f"{newBlock.data} rejected")
```

In the next section I will describe how these main components and processes are used to simulate a repo to auction system and how they create provenance.

3 Simulation

The simulation starts by defining some high-level parameters and global variables. VIN length is set to 11, the blockchain is instantiated, and the company ledger is defined. The company ledger gives the simulation the company ids, names, and types in order to setup the nodes to reflect the data given. Additionally, nodes that are mining enabled have a parameter for how much mining power they have. For example, the first entry tells the simulation that the company RecoveryInc is a recovery company with the id 100 and has a mining power of 2.

```
class Main:
  VIN_length = 11
  companies = dict()
  miners = []
  blockchain = Blockchain()
  company_ledger = [[100, 'RecoveryInc', Company.RECOVERY, 2],
   [200, 'TransportCo', Company.TRANSPORT], [201, 'TransportInc',
Company.TRANSPORT],
    [300, 'MechanicLtd', Company.MECHANIC], [400, 'LocksmithCo',
Company.LOCKSMITH],
     [500, 'AuctionCo', Company.AUCTION, 2], [501, 'AuctionLLC',
Company.AUCTION, 3]]
```

The company ledger is read, and the nodes are created with the setupNodes function:

```
  def setupNodes(self):
    for company in self.company_ledger:
      if(company[2] == Company.RECOVERY):
        Main.companies[company[0]] = Recovery(company[0], company[1])
        self.miners.extend(company[3] * [company[0]])
      elif(company[2] == Company.TRANSPORT):
        Main.companies[company[0]] = Transport(company[0], company[1])
      elif(company[2] == Company.LOCKSMITH):
        Main.companies[company[0]] = Locksmith(company[0], company[1])
      elif(company[2] == Company.MECHANIC):
        Main.companies[company[0]] = Mechanic(company[0], company[1])
      elif(company[2] == Company.AUCTION):
        Main.companies[company[0]] = Auction(company[0], company[1])
        self.miners.extend(company[3] * [company[0]])
```

Finally, the simulation is run. In this simulation, random VINs are generated with the generateVIN function which produces a 11-digit alphanumeric unique vehicle identification number which is used to identify a car in the blockchain and in the automotive industry as a whole.

```
  def generateVIN(self):
    return ''.join(random.choices(string.ascii_uppercase + string.digits, k =
self.VIN_length))
```

There are two main demonstrations in the simulation: running transactions successfully with two different VINS and running a transaction that is invalid and gets rejected. For the first section RecoveryInc recovers a car with the first randomly generated VIN and sends it to MechanicLtd through the transportation company TransportCo. After this operation the miners have some time to run some mining and the consensus model above is executed. Next, MechanicLtd performs the repairs and sends the car to AuctionCo through TransportInc. AuctionCo auctions the car, and the first VIN has completed its run through the system. Mining is run again, and the transactions are added to the blockchain.

```
    firstVIN = self.generateVIN()
    #RecoveryInc recovers a car
    Main.companies[100].performOperation(firstVIN)
    #TransportCo takes the car from RecoveryInc and delivers it to
MechanicLtd
    Main.companies[200].performOperation(firstVIN, 100, 300)

    #Run mining simulation
    self.runMining()

    #MechanicLtd repairs the car
    Main.companies[300].performOperation(firstVIN)
    #TransportInc takes the car from MechanicLtd and delivers it to AuctionCo
    Main.companies[201].performOperation(firstVIN, 300, 500)
    #AuctionCo auctions the car
    Main.companies[500].performOperation(firstVIN)


    #Run mining simulation
    self.runMining()

    print("---------------------")
```

Next, the second VIN is recovered by RecoveryInc and transported to LocksmithCo by TransportInc. LocksmithCo cuts new keys for the car and then sends the car to AuctionLLC through TransportCo. AuctionLLC auctions the car, and the second VIN has completed its run through the system. Mining is run intermittently after a few transactions and the transactions are validated and added to the blockchain.

```
    secondVIN = self.generateVIN()

    #RecoveryInc recovers a car and logs the operation
    Main.companies[100].performOperation(secondVIN)
    #TransportInc takes the car from RecoveryCO and delivers it to
LocksmithCo
    Main.companies[201].performOperation(secondVIN, 100, 400)
    #Run mining simulation
    self.runMining()

    #LocksmithCo cuts keys for the car
    Main.companies[400].performOperation(secondVIN)
    #TransportCo takes the car from LocksmithCo and delivers it to AuctionLLC
    Main.companies[200].performOperation(secondVIN, 400, 501)
    #AuctionLLC auctions the car
    Main.companies[501].performOperation(secondVIN)

    #Run mining simulation
    self.runMining()

    print("---------------------")
```

Next, the rejected transaction workflow is shown. The third VIN is generated and recovered by RecoveryInc. A transaction is submitted to the system that shows TransportInc taking the car from RecoveryCo and delivering it to LocksmithCo. RecoveryCo never had possession of the car and the transaction is rejected. Mining is run and only the one valid transaction is added to the blockchain.

```
    thirdVIN = self.generateVIN()

    #RecoveryInc recovers a car and logs the operation
    Main.companies[100].performOperation(thirdVIN)
    #TransportInc takes the car from RecoveryCO and delivers it to
LocksmithCo
    Main.companies[201].performOperation(thirdVIN, 200, 400)

    #Run mining simulation
    self.runMining()

    print("---------------------")
```

Finally, the trace function is shown which is used to create provenance. Anyone that has access to the blockchain can submit a trace request and a VIN. They will get back a printout of all of the transactions that have been performed on that VIN. The current system only describes the transactions that were performed but additional information such as the timestamp and who validated the block can be easily added.

```
    print("---------------------")
    self.trace(firstVIN)
    print("---------------------")
    self.trace(secondVIN)
    print("---------------------")
    self.trace(thirdVIN)
```

At the end of the simulation, the blockchain is written out a file. We can inspect the log of operations during the simulations run, the output of the trace function, and the output file to verify that everything matches, and the application is preserving integrity. Here is the output of the execution log:

```
---------------------
RecoveryInc(100) recovered VIN: 1MB58U3A5EA
TransportCo(200) transported VIN: 1MB58U3A5EA from RecoveryInc(100) to
MechanicLtd(300)
MechanicLtd(300) repaired VIN: 1MB58U3A5EA
TransportInc(201) transported VIN: 1MB58U3A5EA from MechanicLtd(300) to
AuctionCo(500)
AuctionCo(500) auctioned VIN: 1MB58U3A5EA
---------------------
RecoveryInc(100) recovered VIN: 5KOZOUEZZ6F
TransportInc(201) transported VIN: 5KOZOUEZZ6F from RecoveryInc(100) to
LocksmithCo(400)
LocksmithCo(400) cut keys for VIN: 5KOZOUEZZ6F
TransportCo(200) transported VIN: 5KOZOUEZZ6F from LocksmithCo(400) to
AuctionLLC(501)
AuctionLLC(501) auctioned VIN: 5KOZOUEZZ6F
---------------------
RecoveryInc(100) recovered VIN: R6G7JHB6PLI
TransportInc(201) transported VIN: R6G7JHB6PLI from TransportCo(200) to
LocksmithCo(400)
2|R6G7JHB6PLI|201|200|400 rejected
---------------------
```

We can see that the execution log matches the trace function exactly:

```
---------------------
RecoveryInc(100) recovered VIN: 1MB58U3A5EA
TransportCo(200) transported VIN: 1MB58U3A5EA from RecoveryInc(100) to
MechanicLtd(300)
MechanicLtd(300) repaired VIN: 1MB58U3A5EA
TransportInc(201) transported VIN: 1MB58U3A5EA from MechanicLtd(300) to
AuctionCo(500)
AuctionCo(500) auctioned VIN: 1MB58U3A5EA
---------------------
RecoveryInc(100) recovered VIN: 5KOZOUEZZ6F
TransportInc(201) transported VIN: 5KOZOUEZZ6F from RecoveryInc(100) to
LocksmithCo(400)
LocksmithCo(400) cut keys for VIN: 5KOZOUEZZ6F
TransportCo(200) transported VIN: 5KOZOUEZZ6F from LocksmithCo(400) to
AuctionLLC(501)
AuctionLLC(501) auctioned VIN: 5KOZOUEZZ6F
---------------------
RecoveryInc(100) recovered VIN: R6G7JHB6PLI
---------------------
```

And that the trace function matches the blockchain output files exactly as well:

```
Block Data: 1|R6G7JHB6PLI|100
Hash: 000073eb5dd11655b136cde449e7f06c5b912475c36ebc50d2b30d7b4fbd208d
Miner Who Discovered Block: 500
----------------------------------
```

```
Block Data: 5|5KOZOUEZZ6F|501
Hash: 0000238c757347c31579e8a486d8b60d299881adccce9a256889b928a948dca3
Miner Who Discovered Block: 100
----------------------------------
Block Data: 2|5KOZOUEZZ6F|200|400|501
Hash: 000050d378d13b37b4bf7155c9652b2cdfedd28e7e5b9b1d763eb6bcbc1ad059
Miner Who Discovered Block: 100
----------------------------------
Block Data: 4|5KOZOUEZZ6F|400
Hash: 0000597dce939bd88d037025ceaa048a9c062342cdb5827382d1928c888ba7a8
Miner Who Discovered Block: 100
----------------------------------
Block Data: 2|5KOZOUEZZ6F|201|100|400
Hash: 000093ab5cb2bed84c9b5c8f496f4b30b5c50986b05a19736dd59f7c25119964
Miner Who Discovered Block: 501
----------------------------------
Block Data: 1|5KOZOUEZZ6F|100
Hash: 0000f86b9db52cce893d4b5eab926dff30f67df9cebcacb2ab64422c0bcde360
Miner Who Discovered Block: 501
----------------------------------
Block Data: 5|1MB58U3A5EA|500
Hash: 0000cab921e01e68984d5ef4ff07304656fc18f2f698242274676ab9b8b6c46a
Miner Who Discovered Block: 501
----------------------------------
Block Data: 2|1MB58U3A5EA|201|300|500
Hash: 00006f37bf02159c8d11f5193422725d2485ee2a918654364bd27ad1dadf3f7a
Miner Who Discovered Block: 100
----------------------------------
Block Data: 3|1MB58U3A5EA|300
Hash: 0000b61d57425d8ff49135ebd656a512c46b0cd6a54b1b45ad7692d4d648095a
Miner Who Discovered Block: 100
----------------------------------
Block Data: 2|1MB58U3A5EA|200|100|300
Hash: 0000965e6e5bc306370d816db250d236d45893a7351ba5cd8b529ea9c2ce258f
Miner Who Discovered Block: 100
----------------------------------
Block Data: 1|1MB58U3A5EA|100
Hash: 00009f396d60c2b1d65f812da2d942375a1b44f32761fab84e49e60c492cabfa
Miner Who Discovered Block: 500
----------------------------------
Genesis Block
Hash: 0
Miner Who Discovered Block: NA
----------------------------------
```

In conclusion, the proposed blockchain system has added valuable traceability and communication between business units in the repo to auction system. With the proposed blockchain system, transactions performed on a vehicle are closely monitored and validated with a proof of work consensus model. The application demonstrated how provenance can be added and maintained to an existing workflow. The proposed private blockchain system makes it easier for customers and businesses to have insights on what has actually been done to a vehicle and when. It becomes trivial to track down who performed specific operations on a car and the overall accountability within the repo to auction process is increased. The reliability of this data is critical and a blockchain system promotes high integrity.

4 Future Work

In future iterations of the system additional node types could be added to better reflect the industry in real life. For example, storage lots, banks, and detailing service companies were left out of this simulation but are also important businesses involved in the recovery to auction process. Adding these types of nodes should be easy thanks to the Node abstract base class which encapsulates what it means to be a node fairly well. Another valuable addition to the system would be allowing multiple operation types per node type. For example, locksmith companies and mechanic companies are able to perform more operations than cutting keys and repairing a vehicle. Additional operations such as replacing locks or performing a tune up could be added for those types of companies respectively. More fined grained operations and text descriptions could also be added to the data of a block. For example, the main operation might be repair but a sub operation could be open as a text field that could describe what repairs were completed. A mechanic might set the operation to repair and then fill in the suboperation text field to say that the brakes were changed, and the oil filled. Similarly, the current system has one transaction per block which becomes spatially expensive at scale. One transaction per block could add up to gigabytes of data. Allowing more transactions per block could help decrease the size and increase the speed of mining but could potentially take away from the speed of traceability. Traceability suffers in terms of speed because the block that contains the previous VIN transaction must be scanned to find that transaction. It would be interesting to see how this system operates at scale and these parameters could be tuned accordingly. Another improvement to this system could be adding a previous hash with vin field. This would be calculated when the transaction is being validated and add to the block. Having the previous hash for the last seen block with the vin in question would make the validation process much faster since you could skip all the transactions between VIN transactions. This essentially creates a traceable chain for every VIN on the blockchain and significantly reduces the amount of data that needs to be sorted through to create provenance. However, this approach would suffer in terms of space required since it would increase the size of a block. Along this same line of thinking, a caching mechanism could be added to cache which blocks have been seen most recently with a specific VIN. Most of the operations check for the previous transaction with a VIN and having this information stored in a cache could significantly speed up the validation process. Finally, proof of stake methods could be explored but these methods are costly and can be limiting in terms of security.

5 References

https://www.coxautoinc.com/market-insights/repossessions-are-down-thanks-to-accommodations-and-stimulus/

https://www.sundevilauto.com/why-is-a-vehicle-history-report-important/