6-18-2021

A Secure Proof of Delivery Scheme for Crowdsourced Last Mile Delivery Using Blockchain

Vipul Malhotra
University of Windsor

A Secure Proof of Delivery Scheme for Crowdsourced Last Mile Delivery Using Blockchain

By

Vipul Malhotra

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2021

A Secure Proof of Delivery Scheme for Crowdsourced Last Mile Delivery Using Blockchain

by

Vipul Malhotra

APPROVED BY:

B. Anderson
Department of Political Science

A. Jaekel
School of Computer Science

S. Saad, Advisor
School of Computer Science

April 19, 2021

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Shipping a product from a warehouse shelf to a customer doorstep might look simple initially, but it is a complicated and expensive operation in reality. Moving the product from the last distribution hub in the distribution channel to the final destination (residential or business address), also known as Last-Mile Delivery (LMD), is the costliest and time-consuming phase. Studies show that LMD accounts for 53% to 70% of the total cost of transportation.

One of the main challenges in the LMD operations is proof-of-delivery (PoD). It is the responsibility of any LMD solution to provide Proof of Delivery (POD) to confirm that the purchased item is delivered to the customer (buyer). It makes both sender and receiver accountable because POD acts as proof that the sender sends the package to the correct destination, and the buyer receives the product in good condition. Most of today, PoD solutions uses paper and pen to collect customer's signature as PoD and even when handheld devices are used, it only to collect buyer signature. Developing a secure and trusted PoD solution will increase logistics accuracy, improve customer experience, enhance packages visibility and tracking and reduce operational cost.

Recently, digital signature and new technologies such as blockchain introduced an opportunity to improve logistics and LMD operations. Several works in the literature proposed PoD using blockchain and other cryptographic schemes. In our research, we investigated existing work; we selected the most promising solutions in the literature and implemented those solutions to test them against several scenarios. Based on our analysis, we identified several limitations in the existing work; some of these limitations are related to security, trust, and scalability. Finally, we propose a new PoD scheme using blockchain, blind signatures and a reputation-based trust model to overcome existing limitations in state-of-the-art PoD solutions.

DEDICATION

I would like to dedicate this thesis to my mom for her incredible love and support. Because I believe that she was the real backbone of our family, this is to appreciate her selfless hard work and efforts towards the family.

Furthermore, I dedicate it to my dad to raise me like a son and give me wings to fly. And to my entire family for their unconditional affection towards me.

AKNOWLEDGEMENTS

I would like to sincerely express my most profound gratitude towards my supervisor Dr. Sherif Saad Ahmed, whose input helped me immensely. With his input, I was able to look at my research with a different perspective and a more critical eye.

Secondly, I would like to express my gratitude to my thesis committee members for their beneficial advice and suggestions for my thesis.I would also like to thank my brother Rahul for always encouraging and supporting me.

This project was supported in part by collaborative research funding from the National Research Council of Canada's Artificial Intelligence for Logistics Program.

I humbly extend my thanks to the School of Computer Science and all concerned people who helped me in this regard.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CHAPTER 1

Introduction

1.1 Overview

Last Mile Delivery (LMD) as the term suggests is the movement of an item from its last distribution hub to the final destination. It is considered to be the costliest and time-consuming phase in the whole delivery scenario. The rapid growth of e-commerce and online retailer's and customer's expectations of same-day-delivery and on-demand delivery made LMD an urgent logistic challenge. Several factors increase the operational cost of last-mile delivery, such as limited delivery services in urban and rural areas, lack of visibility which result in failed delivery, lack of secure proof-of-delivery to prevent common last-mile scams and fraudulent activities.

The global cost of LMD in 2018 was \$1.99 billion and is expected to reach \$7.69 billion by 2027 [52]. As an example, Canada offers online retailers significant growth opportunities with a population of over 35 million. However, with a landmass that exceeds 9.98 million square kilometres, many find it challenging to get their products into their customers' hands. Hackers commit different types of frauds during the last-mile journey of a product, including reshipping fraud, stolen or compromised shipping accounts and interception fraud [51]. Based on Transunion findings [51] , shipping fraud were among the most significant fraud trends in 2019 which saw 391% increase year over year.

There are many stakeholders in LMD, and it is important to design an operational last-mile delivery scheme that satisfies the expectations of those stakeholders. The common LMD stockholders are well described [50] by Taniguchi et al. Those key stakeholders are:

- **Shippers:** responsible to maximize their profit while optimizing the level of service under the time window constraint.
- **Freight carriers:** aim to minimize the transportation costs while optimizing the level of service under the time window constraint.
- **Residents or consumers:** responsible to maximize their gain by the timely purchase of necessary goods.
- **Administrator :** responsible to maximize the economic prosperity of the city and to align all conflicting interests together to achieve a sustainable transport system.

Our research focuses on developing a decentralized last-mile delivery scheme that improves delivery time, visibility, and reduces potential scams and fraudulent activities.

1.2 Last Mile Logistics Challenges

Challenges associated with last-mile logistics are many, like the number of failed delivery attempts, insecure proof of delivery, choice of the last-mile delivery mode, how delivery attributes affecting consumer participation in buying the product and effects of the delivery window etc. In this thesis, we will focus on a subset of the LMD challenges. In particular, we focus on security, visibility, delivery mode and delivery time. We do not focus on optimization challenges, such as optimizing the number of carriers or vehicle routing.

The impact of failed first-time home deliveries on other carrier journeys, i.e. repeat deliveries and customer trips to retrieve their carrier depots' assets, increases concern to e-retailers. Liying Song et al. [49] try to address this last-mile problem (delivering a package from the last distribution hub to the customer's house) through a concept of collection and delivery points Collection and Delivery Points (CDP). With a large proportion of failed deliveries, many different delivery solutions emerged, allowing carriers to drop off consignments without the actual need of obtaining the signatures Proof of Delivery (POD). These solutions limit several desirable delivery modes, such as same-day delivery, and on-demand delivery, and other custom delivery modes. Moreover, they have security concerns and skip the most important part in LMD i.e. obtaining proof of delivery which later resulted in the correct customer delivery problem, hence incurring unnecessary refund costs for sellers. Despite major limitations, their proposed solution results show that carrier processing costs associated with home delivery failures are reduced significantly by diverting failed deliveries to CDP.

In [56] Xuping Wang et al. compare the competitiveness of three commonly used last-mile delivery modes, namely, Attended Home Delivery (AHD) or in-person delivery, Reception Boxes (RB) and CDP under different scenarios exceptionally high population density areas. AHD couriers send goods to customers' doorsteps, receive their signatures via hand-held devices and leave for the next order - this is the most widespread last-mile delivery model. Though AHD provides the opportunity to interact face-to-face with customers, its low operation efficiency makes it undesirable for massive orders, and insecure POD aggravates the concerns. There are various kinds of reception boxes such as independent reception boxes installed at the garage or home yard, delivery boxes equipped with docking mechanisms retrieved after goods inside taken away and shared reception boxes installed near customers for shared usage. However, the reception boxes investment is costly, and there is no available evidence of the product received to correct hands. CDP refer to convenience stores and other institutions belonging to cooperates with express companies where customers pick up their goods. Customer satisfaction will not be affected as CDP are the places they commonly visit, i.e. grocery stores, but the verifiability of the correct customer receiving the product POD is always a concern.

To overcome these challenges of obtaining proof of delivery either by hand-held devices or via insecure mechanisms and eliminating the need for buyer to be present at the time of delivery, the concept of blind signatures and proof of delivery via immutable blockchain ledger are investigated in this thesis. A secure POD scheme should provide flexibility to the buyer to nominate another person on his behalf to receive the product, makes each party accountable and builds transparency in the system.

Customer satisfaction in the last-mile delivery always a critical factor from the seller's perspective. Dung H Nguyen et al. [38] conducted a study on consumer preferences for online retailing delivery options. Their study identified five main factors related to product delivery that affect the customers' buying decisions. These factors are the accuracy of estimated delivery time, delivery speed, delivery cost, the flexibility of delivery day (e.g. weekday, weekend), and delivery time slot (e.g. morning, evening, or night). Of course, some of those factors are more important in some online shopping cases, for example, Chen et al. [9] showed that the speed of delivery is a crucial concern to consumers shopping online for specialty foods, where short delivery time directly contributes to the quality of food quality. In addition, Dung H Nguyen et al. [38] analysis showed significant differences between gender and income group. Certain customer segments indicate distinct preference structures they name it as a price-oriented community, time-and-convenience-oriented value-for-money-oriented community. In addition to the previously mentioned factors, visibility in LMD is another essential factor; visibility refers to tracking and tracking delivery items in real-time [11] .

To achieve high customer satisfaction and accommodate the above-mentioned delivery attributes in my proposed last-mile delivery scheme, I provided buyers flexibility in changing location and superior product handling instructions even after the order is placed. This flexibility is provided via the concept of crowd-sourcing, which empowers sellers to assign carriers from the available pool according to the customer's customized delivery attributes. Furthermore, to eliminate the need for buyers present at home at the time of the delivery concept of blind signatures introduced that provides the ability to sign on the document on behalf of the buyer as well. To ensure each party in a last-mile delivery get their share of data visible, we propose a new method to obtain POD using an immutable permission-ed blockchain network. It empowers the party to see their share of data and builds visibility to the system.

1.3 Motivation

One potential solution to the ever-growing cost of last-mile delivery to help online retailers fulfill customers' requirements and successfully meet their expectations is crowdsourcing or crowd-shipping. The use of crowd-shipping will reduce LMD delivery costs and enable small and middle-size online retailers to compete with large retailers that use expensive commercials carriers or have their fleets.

Using crowdsourcing for LMD requires a new LMD scheme that enables trust between the key stakeholders. The requirements of this new LMD delivery scheme are decentralization, security, trust, and flexibility. In the literature, very few works proposed LMD schemes that could satisfy these requirements. The majority of the literature's work focuses on optimizing LMD for fixed-size fleet and mainly on routing and minimizing carriers' number.

1.4 Problem Statement

There is an urgent need for designing a flexible LMD scheme that leverages crowdsourcing. We need to investigate the use of blockchain and smart contracts to develop a secure LMD scheme. The scheme should provide a secure approach to obtain a trusted and verifiable POD and encourage fairness between the LMD stakeholders. The requirements of such scheme based on current LMD challenges are:

1. Build immutable proof of a delivery system that brings visibility for each party involved in a delivery.

2. Verification of agents via secure cryptographic methods.

3. Leverage benefits of crowd-sourcing to build trust in the last-mile system.

4. Solution must be auditable to handle disputes.

5. Ensure each participating entity in delivering a product have enough incentive and equal opportunity to participate without reserving collateral.
6. Penalized mechanism to discourage malicious activities.
7. Eliminate the need for receiving entity to be present at the time of delivery.

1.5 Thesis Contribution

The contribution of this thesis is summarized in the following points:

- Reviewed literature on last-mile delivery challenges, studied various design patterns used by the previous researchers in building a proof of delivery systems to overcome these challenges and the limitations involved in their research work.
- Reverse engineering the state-of-the-art proof of delivery scheme for delivering physical assets.
- Further investigated how using blockchain one can enable different delivery modes such as crowd-sourcing with secure POD. Accommodated crowdscouring benefits in our designed solution, especially in selecting the carrier from the available pool based on their reputation metric.
- Design a new scheme for LMD using blockchain and smart contract that supports secure and trusted POD using the blind-signature in the crowdsourcing environment.

1.6 Thesis Organization

The rest of the thesis is organized as follows:

- Chapter 2 discussed how using blockchain contributes to last-mile delivery and various proof of delivery solutions that use blockchain as a means of traceability and delivering physical assets. Additionally, it presents an analysis and limitations of these works and discussed how crowdsourcing platforms along with blockchain's immutability contribute to last-mile delivery operations.

- Chapter 3, present the design of a new crowdsourcing scheme for LMD by levering blockchain and smart contracts. The proposed scheme uses a blind signature, a lightweight reputation system. Provide algorithmic details of different smart-contract modules which are used to execute the whole last-mile delivery process.

- Chapter 4, we evaluate the proposed scheme using a real-life delivery orders dataset. We provide an implementation of the proposed scheme using Ethereum [13] as a proof-of-concept. In addition, we compare the performance and the cost of the proposed scheme with existing work in the literature.

- Finally, Chapter 5 concludes the thesis and discusses the potential future work.

CHAPTER 2

Related Works

This chapter focus on the existing literature that uses blockchain for obtaining POD, blockchain for crowd-sourcing and last-mile delivery in general. The primary focus is on literature that contributes towards using blockchain as means of delivering physical assets and crowd-sourcing via reputation-based systems.

2.1 Blockchain

Blockchain refers to a distributed ledger of records that act as proof of transactions between various entities. A typical blockchain transaction road-map involves the following stages. First, a transaction is requested (sending money from one person to another). Second, the transaction is broadcasted over a peer-to-peer network. Third, the nodes in the peer-to-peer validates the transaction. Fourth, the transaction is represented online as a block. Fifth, blocks are chained using cryptographic techniques. Sixth, blocks are added to the existing chain and seventh, the transaction is said to complete.

2.1.1 Blockchain Types & Consensus Algorithms

A consensus algorithm referred to a protocol through which all peers in the blockchain network reach out on a standard agreement about the distributed ledger's state. Thus, the consensus protocol builds data consistency and establishes a coordination pathway among the peers. The blockchain consensus protocol comprises some specific objectives, such as building an agreement, cooperation, co-activity, equivalent rights

Table 2.1.1: Difference between various types of Blockchain

Public (Permission-less) Blockchain	1. Zero trust network of peers who competes against each other to confirm a transaction, arrange the transaction into blocks and get rewarded. 2. The main working principles are a complicated mathematical puzzle and a possibility to easily prove the solution. 3. Open to any party who is ready to join the network. 4. High Transaction cost. 5. Examples: Proof of Work(POW)
Private (Permission-ed) Blockchain	1. Only certain parties are allowed to control the transaction that is written to the block. On top of the blockchain, a control layer runs which governs action performed by allowed parties. 2. Super peers are trusted to some degree. 3. Low Transaction cost. 4. Examples: Hyperledger Fabric
Hybrid Blockchain	1. Blend of the private and public blockchain. It utilizes the features of both types of blockchains that is one can have a private authorization based framework as well as public consent less framework. 2. Opt to incentivize users if they want to. 3. Low Transaction cost.

to each node, and compulsory participation of every node in the consensus process. Hence, an agreement calculation targets finding a typical understanding that is a success for the entire network.

Many consensus algorithms exist in the literature, but I will discuss the commonly used consensus algorithms in Blockchain-based applications—Proof Of Work (POW) consensus algorithm utilized to choose a miner for the next block. POW calculation's focal thought is to tackle a tricky numerical riddle and effectively give out an answer. This numerical riddle requires a ton of computational power, and consequently, the node who tackles the puzzle at the earliest opportunity will mine the following block. For a more detailed understanding of POW refer [22].

Byzantine Fault Tolerance (BFT) consensus algorithm is the element of a distributed network to arrive at consensus(agreement on an equal worth) in any event when a portion of the nodes in the network neglect to react or react with erroneous data. A BFT component aims to protect against the framework disappointments by utilizing aggregate choice-making (both – right and flawed nodes), which plans to decrease the impact of defective nodes. For a more detailed understanding of BFT refer [20].

Proof of Stake (PoS) is a sort of agreement component utilized by blockchain organizations to accomplish circulated agreement. It expects clients to stake their ether's(ETH) to turn into a validator in the organization. Validators are liable for exactly the same thing as excavators in POW [22] requesting exchanges and making new squares so everything hubs can concede to the condition of the organization. The number of upgrades in contrast to POW is better energy effectiveness, lower passage boundaries, decreased equipment necessities, more grounded invulnerability to centralization, and more grounded support for shard chains. For a more detailed understanding of PoS refer[14].

In the Proof of Burn (PoB) consensus algorithm, instead of placing into expensive gear equipment, validators 'burn-through' coins by sending them to the location from where they are unrecoverable. By presenting the coins to an inaccessible location, validators get a benefit to mine on the framework subject to a random selection process. Hence, consuming coins here infers that validators have long-term commitment as a trade-off for their short-term loss. For a more detailed understanding of PoB [5]

In the Proof of Capacity (POC) consensus algorithm, validators ought to contribute their hard drive space as opposed to placing assets into exorbitant hardware or burning-through coins. The more hard-drive space validators contribute, the better are their chances of getting picked for mining the following block and gaining the block reward. For more detailed understanding of POC refer [46].

Some other consensus algorithms are Proof of Elapsed Time, Proof of Activity, Proof of Weight, Proof of Importance, Leased Proof of Stake, etc. [30].

2.1.2 The Role of Blockchain in LMD

A blockchain ledger can maintain and store all information pertaining to a transaction and acts as a single source of truth for customers, sellers, custom officials, delivery services and so forth. Each entity involved in the delivery gets access to their share of records thus, restricting the sensitive information to be withheld with accountable entities. Million of goods trade daily across the globe that range from small shipments to massive containers of ocean cargo thus providing many points of failure in a product journey where fraud or theft might occur. The adoption of blockchain-based last-mile delivery solutions will enable accurate documentation and fast transfer of original documents. This makes the supply chain more efficient, builds trust between parties and reduces unnecessary transaction costs.

Shipchain [45], is one of the blockchain-based solutions existing in the market to provide the final delivery on the customer's doorstep. Key functionalities in their business application are as follows: pick up and delivery with full transparency provided by the blockchain from the convenience of your smartphone, monitor in-real-time, tracking updates simultaneously between web and mobile for any moving shipments, upload signed documents, handling procedures, inventory lists, and shipment images with just a few taps and share key shipment information and updates with the recipient through the life of the shipment. However, there are certain limitations in terms of secure verification of delivery agents and obtaining proof of delivery using handheld devices which need to be addressed, and those issues are addressed via my newly proposed POD platform explained in chapter 3 of this thesis.

2.2 Supply Chain and Traceability

Mehmet Demir et al. [12] focus specifically on a subsection of the supply chain industry called "the last mile." When parties with clashing interests team up in a business climate, they need to construct trust to smooth the deals. It is average that when the business goes as arranged, there is no explicit requirement for intermediation; all gatherings direct and proceed with their organizations inside their decent edges. Trust is essential amid conflict; when things don't go true to form, parties need verification. They need a dependable, untampered, and verifiable record of information identified with the exchange in question. Blockchain innovation gives this trust.

Other than its exemplary advantages, blockchain innovation offers answers for two principle issues in the conveyance business. These are *Chain of custody throughout the handover of packages* and *Continuous monitoring*. Chain of custody is an issue when multiple parties conduct business indirectly through their delegates, intermediaries or specialists. These interactions, fundamentally the handover packages between independent parties, have a security and trust issue. The absence of ordered documentation or paper trail recording the succession of custody and handovers with adequate physical or electronic proof feeds the problem. This trust issue cost organizations as business misfortune or costs, for example, insurance fees due to difficulties in finding responsible entities for harm that occurs at an unknown time.

To defeat the information flow issues, help other supporting business processes and assemble trust, they proposed a novel blockchain-based structure Blockchain and IoT Delivery Assurance on Supply-chain (BIDAS) for making solutions for delivery assets that record and offer information on business members' interaction. This structure makes arrangements that incorporate handover and monitoring aspects of the delivery businesses and leverages few benefits of hyper-ledger blockchain technology. BIDAS models the interaction flow between the initiator and administration delegates. The price sensitivity of the clients towards the delivery fees would be a test for this sort of system as the advantages of the blockchain may not legitimize the expense of each execution. Security and accessibility issues are additionally drilled down. However, they guaranteed that their system carries trust-capable monitoring even with restricted data.

Mohamed Awwad et al. [3] examines the case studies on early execution of blockchain technology with Internet of Things (IoT) with uncommon significance on the degree of deployment of blockchain technology for validation, transparency, and traceability purpose at different enterprises, for example, online business, food, and warehousing etc. Traceability in supply chain research fundamentally required because of expanding transportation and shipping items throughout the world. They featured different use-cases that are leveraging blockchain technology to tracing products. For instance, IBM teamed up with Capgemini at the Watson IoT centre and effectively built up a Smart Container Management blockchain technology prototype, tracking tuna fish over the blockchain framework, British airways use-case on maintaining flight information on blockchain and many more.

Shiaofang Liang et al. [32] explores the utilization of traceability systems and blockchain technology in domestic and foreign industry fields. The conventional product traceability system has the issues of product label copying, spamming and product quality issues and the difficulty of positioning the problem link. Consolidating blockchain technology and supply chain qualities, they build blockchain and supply chain logistics information ecosystem model from three dimensions: object, attribute, and function. The primary layer is the base layer made out of the infrastructure and information. The infrastructure layer gathers the fundamental information created by each connection of coordination exercises through sensors, standardized identification and other information procurement hardware. The basic information incorporates logistics information, products, stockpiling, and so on. It likewise includes the enterprise information of third-party logistics enterprises. After the necessary information is transferred to the data layer through the transfer mechanism it is encrypted asymmetrically and timestamped to create information blocks, which are then connected into a blockchain. The subsequent layer is the core technology layer, comprised of the network, the consensus and the contract layers, respectively. The network layer spreads and stores the information blocks to each node through a specific network transmission protocol and authentication mechanism and permits the confirmed nodes to participate in the consensus and record blocks. The third part is fundamentally the interaction layer, where the individuals from the chain conduct various business operations. The major limitations of these systems are that they consider traceability of an item as a major issue, not consider accountability of entities and they build a Business to Consumer (B2C) platform instead direct Peer to Peer (P2P) platform where consumers interact externally via interaction layer without their direct participation in the blockchain system.

Recent work from Ju Myung Song et al.[48] highlights the challenges like a complex chain of actors, high-cost entry barriers, challenges in tracing physical items and above all culture adoption of decentralization in the last-mile logistics industry. To address a portion of these difficulties, Johannes Kretzschmar et al. [31] introduced a strategy to execute cargo sharing in the last mile setting utilizing a P2P network arrangement dependent on blockchain and smart contracts. Their work's objective is to execute a platform functionality in logistics via a combination of an adapted blockchain and smart contract technology as advancement to a sharing platform. The attention is on the aversion of a proprietary provider with related costs because of infrastructure, service provisions and eventually the danger of reliance on a market monopoly position. Plus, there ought to be full authority over information distributing, stockpiling and a straightforward interaction model through open smart contracts. Contrasted with sharing platform solution, their P2P approach evades the danger of infrastructure failure with increased performance and scalability, as claimed by researchers. Their proposed execution tried to store demands over blockchain, and the offer stage carried out through Whisper channels (P2P clients discuss straightforwardly with one another utilizing the Ethereum direct correspondence convention- Whisper). Every customer manages and receives offers autonomously locally. After the lapse of the solicitation cutoff time, the acknowledged bids calculated by matching algorithm are put away on the blockchain. The contractors notified via a corresponding event trigger or Whisper message. Their P2P approach guarantees simple specially appointed admittance for sharing e-assets, and distributed application suggests high unwavering quality and adaptability.

A portion of the difficulties demonstrated in their work are: heuristics about the accommodation of offers, calculation of the profit share and offer acceptance criteria, a requester genuinely searching for the least expensive arrangement or the smallest possible number of service providers to improve on scheduling and uniform information model that covers all potential parts of last-mile delivery, like time, size or weight limitations or extra prerequisites, like cooling or tracking of the carrier vehicle. Work from both of these researchers contributes towards sharing resources for last-mile logistics via e-marketplace use-case and helps in e-procurement.

2.3 Blockchain in Delivering Physical Assets

Riham AlTawy et al. [1] address the research problem of anonymous delivery of the purchased physical goods via a blockchain-based platform called- Lelantos. Their system is inspired by onion routing techniques which used to achieve anonymous message delivery. Lelantos combines a blockchain smart contract interface to reasonably and secretly a transitional delivery process without a trusted third party. It is a web service to promote and enlist delivery companies that offer the requested service and contractual party-side applications to monitor the state of the smart contract and communicate with it based on the contractual party's role. This platform utilizes a lightweight execution of the on-chain operations to limit the on-chain code execution and consequently gas use. The primary highlights of this stage include:

1. Fair Exchange: The package delivery directed by a decentralized smart contract guarantees the reasonable exchange of assets to both vendors and delivery companies and that package delivered to the intended customer.

2. Customer anonymity: No private data identified with the client revealed to any

of the contractual parties.

3. Customer-merchant unlinkability: the inability of an adversary to trace a given client back to the shipper or the opposite way around by inactively observing the state of the contract or by being working together with appropriate chosen subsets of delivery companies.

However, certain assumptions are made which otherwise breach their systems, like no external attacks via GPS device, appropriate packing of goods, and delivery companies paying regular shipping fees etc. In the end, they defined the security properties of their system and provide evidence to support their security claim. They contrasted their model and other existing recommendations and guaranteed their convention disposes of the requirement for a confided in outsider and guarantees reasonable trade between contractual parties and the offered obscurity by means of both the blockchain and onion routing protocols.

The most promised solutions that tackle the problem of proof of delivery in delivering physical assets are provided by Khaled Salah et al. [24]. Researchers provided two versions of POD, one with a single transporter involved in the delivery scenario and the other with multiple transporters involvement. The proposed solution is sufficiently conventional and can be applied practically to almost all shipped physical items and assets. Their solution boosts each participating entity, including the vendor, carrier, and purchaser, to act honestly, and it dispenses with the requirement for an outsider as an escrow. The proposed POD solution guarantees responsibility, timeliness, trustworthiness and makes it auditable. Additionally, the proposed solution utilizes a Smart Contract Attestation Authority (SCAA) to guarantee that the code follows the terms and conditions endorsed by the participating entities. In the real world, courier and delivery companies use trackers and proof of delivery frameworks to guarantee that their client's necessities are met on schedule and without delays. This solution uses equivalent settled upon security to boost every one of participating entities to act genuinely. It guarantees the integrity of signed terms and conditions structure by utilizing Interplanetary File System (IPFS) hash stored in the smart contract. It exhibits responsibility by using keys and hashes for the check of the genuine real beneficiary. Refund and cancellation are likewise dealt with to safeguard the privileges of the vendor, purchaser and carrier. Each participating entity signs the terms and conditions agreement and consent to its content by depositing collateral that is double the delivery item's cost. At that point, the vendor prepares the item and hands it over to the carrier. Each item has two keys that the vendor gives, for example, a key assigned to the carrier referred to as KeyT and a key given over to the purchaser referred to as KeyB. The carrier delivers the item to the purchaser, and the two of them trade their keys. It guarantees that the carrier has arrived at the planned purchaser. Both the carrier and purchaser enter the keys to the smart contract, and confirmation happens. The smart contract registers the hashes of the keys entered, and if the hashes match, the delivery payment is settled. The purchaser is refunded one item price; the carrier gets its deposited collateral with additional 10% of the item price paid for the delivery service. Finally, the vendor receives the remainder of the stored item price, including the vendor's collateral and 90% of item price paid by the purchaser.

In its second version[42], it gave a proof of concept for multiple carriers' inclusion by collaborating with a chain of smart contracts. Like the previous one, the framework's principle participating entities are vendor, purchaser, carrier/(s), arbitrator and SCAA. However, the difference is that as the item gets handed over between two entities, a chain of contracts is made dependent on the number of carriers and at least two contracts needed between a vendor and a purchaser. In any case, the terms and conditions segment of the contract between the vendor, purchaser and carrier(s) stays as before. Three types of contracts intended to deliver the item between the vendor and the purchaser and make it versatile to the number of carriers required per delivery. The contract is made dependent on the need, and together they make a chain of contracts. Each contract points to the next contract. Hence, every parent contract has the location of its child contract, and each child has the location of its parent contract. Additionally, all contracts have the location of the primary main contract that began the chain. The main contract has an additional location, which is the last contract's location in the chain. Delivery consistently starts with the contract of the type POD and finishes with contract Buyer Transporter Contract (BT). Consequently, POD is the main contract, and BT is the finish of the chain contract. In the centre, if the number of carriers is more than one, contracts of the type Courier Service contract (CS) are created as per delivery needs.

In the POD contract, the vendor, purchaser and first carrier sign the terms and conditions and store the agreed-upon collateral. Afterwards, the vendor creates the package and physically hand it over to the carrier with a key. The carrier would then make the following CS contract and Carrier 2 consents to the terms and conditions and deposits collateral held by the CS contract. Consequently, every contract goes about as an escrow to the Ether kept to it. Carrier 2 would then receive the packaged item and notify everyone that Carrier 1 has shown up. It will permit Carrier 1 to affirm that it has reached and that the key is currently with Carrier 2. Carrier 2 enters the key, which hashed and contrasted with the key hash effectively accessible in the contract. If the confirmation is successful, the following CS contract is made by Carrier 2, and the chain goes on until the objective location is the purchaser's location. When the objective is equivalent to the purchaser's location, a BT contract executes, and the last key verification stage completes.

2.4 Crowdsourcing

Katarzyna Gdowska et al. [19] consider a concept of occasional carriers which act as free agents to accept or reject the assignments. The objective is to utilize crowdshipping to decrease the total delivery cost in the same-day last-mile delivery framework by permitting Occasional Courier (OC) opportunity to accept or dismiss the allocated delivery. They introduced probability functions to represent OC's readiness to deliver the item to the final customer. The OC's readiness to acknowledge or dismiss the delivery task that is appointed to them and the impact of their choice on the total delivery cost assumes a significant part in LMD. They urged OC's to participate in the framework by offering them exclusively determined expenses - based on historical information concerning the connection between OC's readiness to accept delivery and the given remuneration charge. Subsequently, researchers recognized and advocated the requirement for another system to ascertain a satisfactory pay rate for OC's by considering the dynamics involved in the delivery process and consequently lowering down the cost of last-mile delivery.

Simone Serafini et al. [44] conducted a sustainable crowd shipping study in Rome using public transport for last-mile delivery. It mainly focuses on crowd shipping services deployed using the public transport network wherein passengers act as crowd-shippers that are already moving for some other reasons. Researchers utilized expressed inclination to distinguish the main highlights related with the decision of acting as crowd-shipper and discrete decision models to understand underlying behaviour. The research covers the metro areas of Rome, Italy, to quantify the effect of this freight transport strategy in the urban context. Their methodology assumes the packages can be picked and dropped off in Automated Parcel Lockers (APL) located either inside the metro station or in surroundings. Their survey identified socio-demographic characteristics based on green attitude, i.e. their belief in adopting sustainable transport modes or using organic products. The green attitude is estimated in four levels(no GA, low, medium, high) partner a load to explicit supportive of ecological, social viewpoints. Results showed 74% of respondents self-expressed a medium degree of a green mentality. Locally situated outings expressed by respondents and including the metro are mostly occurring in the first part of the day top (79%). A comparative recurrence for the work-to-home tours in the early evening is primarily due to the suburbanite trips during non-weekend days (84%). Admittance to/from metro stations is typically performed by strolling (52.9%) and the average travel time related with the whole is roughly 50 minutes. Results demonstrated that if delivery points are situated inside the metro stations, a worker can go about as a carrier with a possibility of 54.8%, notwithstanding the low compensation. Along these lines, researchers with this work help to recognize the most pertinent instruments/service characteristics to work with to build up a crowd-shipping service that can depend on an adequately enormous base of potential crowd-shippers to provide a reliable solution to a substantial number of delivery requests.

Ming Ni et al. [39] inspects the issue of operations in Same-Day Delivery with Crowd-shipping and Store Fulfilment (SDD-CSF). The principle objective is to close the gap between nearby stores and clients. SDD-CSF makes order fulfilment plan from two viewpoints: request souring choice and delivery method selection to limit the expense related with order fulfilment plan. They embraced the new idea of last-mile delivery from nearby stores utilizing publicly supported transportation using two explicit delivery techniques dependent on distinct characteristics of crowd-sourced shippers: Information Sharing Driver (ISD)'s and OC's. They developed a dynamic programming model for request satisfaction which numerically approximated into a linear programming model. The proposed model considers both currently received orders and predicted future demand to settle on order assignment decision that minimizes direct delivery cost and the future, expected delivery cost. They claimed that under perfect information, the proposed model could converge to the global optimum and used the Instacart dataset to quantify their results. The results indicate that their proposed SDD-CSF model reduces delivery cost up to 15% - 18% approx in comparison to traditional mixed linear programming models like Conservative, Myopic and global-optimal etc used for delivery prediction. Also, researchers accommodated the feedback control system in their model to cope with the inaccurate forecast of demand and ensure timely delivery of items in comparison to other models.

Yuan Wang et al. [57] proposed a compelling large-scale mobile crowd-sourcing where a massive pool of crowd-workers performs last-mile delivery. To proficiently settle the model, they formulated it as a network min-cost flow problem and proposed different pruning methods that drastically decrease the network size. They directed various experiments with Singapore and Bejing datasets. Their outcomes demonstrate that their solution can uphold real-time delivery optimization in the large-scale mobile crowd-sourcing problem. In their crowd-delivery model, some pop-stations circulated the city and an enormous pool of workers prepared to acknowledge the delivery assignments from pop-stations to the purchasers' locations. The logistic companies just should be centred around the scheduling optimization of delivering parcels to the pop stations. They contrasted their model against UberRush. The critical distinction is that UberRush centres around on-demand delivery and sends them to close carriers. However, their model aims to use an enormous crowd-workers pool to complete the last leg of delivery with any possible transportation means. All in all, they treat the transportation means as a black box, and these crowd-workers can walk, take a transport/train, or drive a vehicle to finish the delivery task. A portion of the pruning methodologies proposed by them is cost-based pruning, the limit based pruning and recurrence based pruning. Aftereffects of their investigations show that running time improved by two significant degrees from the benchmark solution and can uphold real-time delivery optimization in the large-scale mobile crowd-sourcing problem.

2.4.1 Crowd-Shipping effects on LMD

Brian Odongo et al. [7] investigated the methodology utilized by various crowd-sourcing platforms to understand the network effects. An orchestrated system introduced to investigate how the blend of technological, networking, pricing and motivational strategies applied to crowd-sourcing platforms that concentrate on last-mile delivery. The companies don't deliver the items without help from anyone else; all things considered, they provide a platform that matches clients who need any products, carriers who showed readiness and capacity to deliver items. Their fundamental discoveries address questions like: Can successful technology and business model in passenger mobility and hospitality industries be applied to the delivery of goods? Over the most recent few years, numerous startups leverage crowd-logistics benefits and raised millions in funds. Beyond that, huge players are trying different things with this new delivery service. Amazon, DHL, UPS are to give some examples. The logistics provider is optimistic about crowd-logistics to answer the clients' developing assumptions for quicker, more customized and cost-productive delivery service. The second colossal finding is technologies like mobile devices, digital payment infrastructure, location services, verified user profiles and online reviews, communication application programming interfaces, and platform-specific algorithms enable crowd logistics. Third, the effect of crowd-logistics on LMD is that the fossil fuel emissions estimated at 94% for deliveries in metropolitan regions and 82% in rural territories, as demonstrated in certain studies. Some different investigators did a few trials and found that it helps save 149 km and subsequent decrease of the material impression of 55% and air utilization of 60%. Fourth, to urge individuals to participate in crowd-logistics, they should be granted some momentary remuneration.

A report from Barclays [4] proposes that most logistics providers are optimistic about the future of delivery goods. Barclays tracked down that 92% of suppliers accept that proceeded with development in internet shopping will give future growth opportunities. Nonetheless, more than 50% expressed that the critical area they see as a danger to future gains is adapting to the expanded capacity requirements. Almost 35% picture growth potential through better innovation and expanded global trade and believed that it will reinforce by investments in additional innovative modes of delivery. 28.3% are concerned about increased delivery by retailers' delivery services, and 33.8% are concerned by increasingly price-sensitive consumers, resulting in online shoppers opting for cheaper, retailer-led services.

Hon-Yin Mak et al. [34] designed an omnichannel retail network's pricing strategy for its on the web and in-store channels dependent on shoppers' choices on the purchase, channel selection and participation in crowd shipping. They considered a difference between two repayment modes for crowd shipping, i.e., repaying customer deliverers their incurred delivery costs or providing an additional premium as a (cross-channel) subsidy, against a store-worked delivery model. In the cost-based reimbursement model, since in-store shoppers gain zero surpluses through reimbursements, the crowd shipping option does not alter consumers' incentives to shop in-store or online. In this case, the only effect of crowd shipping is through (potential) efficiency gains in the delivery process, i.e., it benefits the retailer if (and only if) it is cheaper to use consumers for delivery than its delivery fleet. Through cross-subsidy, the retailer can price discriminate between the online and in-store channels more effectively and improve profits and consumer surplus. Such extra gains are more salient for products whose margin is high and the association between consumer's travel costs and product valuation is strong (e.g., high-end products).Accordingly their overall investigation uncovers that the company's productivity impact relies upon shoppers' availability with the low estimation of time, though the pricing impact benefits firms selling high-end items. Peer-to-peer crowd shipping can prompt critical mutually beneficial results in dense metropolitan business sectors for high-end products with everything taken into account.

Vincent E Castillo et al. [8] contributes towards building up an understanding of how Crowd-sourced Logistics (CSL) contributes towards logistics effectiveness by mimicking same-day delivery services from a distribution centre to 1,000 client areas all through New York City under dynamic market conditions and by contrasting the outcomes with those of a conventional devoted fleet of delivery drivers. The discoveries investigated to propose how firms may discover vital advantages utilizing CSL. They featured future key research areas for crowd-shipping like crowd-sourced dedicated mixed fleet size optimization, crowd-sourced driver supply elasticity and supply management strategies, same-day delivery request management strategies, the suitability of CSL for various urban communities, impact on omnichannel dispersion systems, and motivations for crowd-sourced drivers.

2.4.2 Reputation Based Systems

Reputation refers to building trust between two parties based on their behaviour within the system. Let's discuss an example of UberEats [55] reputation/rating system to allow carriers proper functioning. These ratings help carriers understand what their customers think of them and build trust in the system. Factors for calculation of these ratings are:

1. Once the transporter finishes the pickup, and the application offers the chance to rate the restaurant. They additionally rate their client once they drop off their request.

2. Restaurant staff and clients have the choice to leave you criticism, as well. If somebody gives a thumbs down(negative rating), the application may ask that

individual for somewhat more input concerning why that rating was picked.

3. The application adds the evaluations a transporter gets into an average shown to the clients previously and during the delivery.

4. Carrier can see average fulfilment rating in their application.

5. Generally evaluating comes from their last 100 appraisals from the restaurant staff and delivery clients, and a carrier gets evaluations solely after ten effective deliveries.

If carriers reliably get negative appraisals, warning issued, and Uber share resources, including tips from other delivery individuals. If their rating didn't improve, they might lose admittance to the application.

2.4.2.1 Reputation System Design Patterns

Sergio Marti et al. [36] contributed towards a useful taxonomy of the field of peer-to-peer reputation design. The reputation system's framework usefulness essentially partitioned into three-phase segments: Data Sharing, Scoring, and Response. The main thrust behind reputation system design is offering assistance that seriously mitigates misbehaviour while imposing a minimal cost on well-behaved users. Subsequently, it is fundamental to the behaviour and expectations of typical right users, the objectives and attacks of adversaries, and technological limitations resulting from the system's environment. There are two essential kinds of adversaries in peer-to-peer networks: selfish peers and others are malicious peers. Self-centred peers wish to utilize framework services while contributing negligible or no assets themselves.

On the other hand, malicious peers' objective is to cause harm, either explicitly focused on individuals from the network or the framework overall. The central part of a reputation framework is liable for gathering data on peer's conduct, which will be utilized to decide how dependable they are on the absolute scale and comparative with different peers. Associating a history of behaviour with a particular agent requires a sufficiently persistent identifier; in this manner, a few properties in an identity scheme must be considered. These properties incorporate anonymity, spoof-resistant and unforgettable. Utilizing these network identities, a reputation framework protocol accumulates information on peer conduct in past exchanges to decide the ratings.

The amount and nature of this information are entirely against one another in such distributed frameworks. Most careful peers may need to depend on their insight and utilize just neighbourhood information while deciding to transact with a given peer. Hence as the quantity of information assembled expands, the credibility of each snippet of information diminishes. To increase the information sources a cautious agent primarily relies on information collected from one or multiple sources. The primary sources of information are personal experience, one-hop trusted peers (reputable neighbours having an a-prior relationship), multi-hop trusted peers (reputable peers that can trust other reputable neighbours) and the global system(all reputable peers).

Most reputation systems don't attempt to verify the integrity of the information collected, which plays an essential role in reputation calculation; instead, they assume the majority of users are honest and well-behaved. This problem can be solved using reputation-based weighting similar to Page Rank. To manage novices (outsiders) with no exchange history, an outsider adaptive strategy, i.e., all exchange data on first-time connections with any outsider, are aggregated together. Then utilizing a generosity metric dependent on recent transactions, an agent assesses the possible likelihood of being cheated by the next stranger and concludes whether to trust the next stranger dependent on that likelihood. When a companion's exchange history has gathered, the weighting is done then a reputation score computed. This reputation score calculation occurs through a reputation score function.

In the end, Sergio et al. [36] discussed the incentive schemes and punishment strategies that come under the response component of a reputation system that plays a pivotal role in selecting honest peers. These incentive schemes fundamentally offer one of two incentives: improve service or cash. Service can additionally be improved in terms of speed, amount and quality. Suppose the reputation framework can distinguish effectively malicious peers. In that case, it might fight back in various ways like caution other peers, detach from the adversary, kick malicious peer from the network for a specific period or restrict it. The adversary would have to get another identifier that might be expensive or difficult to get to re-enter the framework.

Andrew Whitby et al. [58] provides a filtering technique to showcase how unfair ratings are different and can be easily detected through statistical patterns. The direct impact of reputation systems is incentivizing legitimate conduct and discouraging unscrupulous parties from participating. Researchers comprehensively characterize the evaluations into two fundamental classes: Endogenous discounting or Exogenous discounting of unfair ratings. Endogenous: covers strategies that bar or give weight to presumed unfair ratings dependent on analyzing and comparing rating values themselves. However, Exogenous covers external factors, for example, the reputation of raters etc. The assumption here is that low reputation raters will give unfair ratings, which requires external evidence apart from reputation values. Researchers primarily focus on the first category (Endogenous), and simulation scenarios are tested to validate the proposed technique. Bayesian reputation framework used to filter out unfair positive or negative ratings, and these frameworks depend on computing reputation scores by statistical updating of beta Probability Density Functions (PDF). The beta-family disseminations are a continuous family of distribution functions ordered by two boundaries, alpha and beta. By differing these alpha and beta boundaries, we can accomplish a uniform PDF. Non-uniform PDF can be interpreted as saying that the general recurrence of a positive result in future is to some degree uncertain. Researchers attempt to address the collection, age and aggregation of ratings as vectors. When aggregated ratings for a specific rating for a particular agent are known, it is feasible to ascertain the agent's reputation probability distribution.

The principal issue with unfair ratings is that these ratings are subjective and unverifiable. Simultaneously, the reputation framework may keep the seller rates honest. What assurance is there that purchaser will be legitimate in the assessment of sellers through ratings? Two types of unfair ratings are unfair positive ratings (ballot-stuffing) and unfair negative ratings (badmouthing). The danger of unfair ratings is most elevated when they can control the reputation frameworks to an agent's benefit. For Instance, a purchaser may collude with a seller to abuse the seller's rivals, bringing about gains to the seller. This issue turns out to be more convoluted in binary ratings where negative ratings (i.e., 0) can't be dismissed essentially because the ratee (an individual who gets appraisals) has received mostly positive ratings (i.e., 1). The essential principle of filtering algorithm introduced here dependent on beta distribution is to check that the score R(Z) of agent Z falls between q quantile(lower) and 1-q quantile (upper) for each rater X.Their simulated results curiously show when unfair raters prevail with regards to making seller's reputations diverging from their honesty levels, purchasers react by turning out to be more risk-averse or more risk-seeking.

Zheng Zhao et al.[59] proposed a reputation model based on attribute reputation , making user identity credible. In designing a system model, two concepts play a major role one is an attribute, and the other is services. For authentication, it need not check each attribute; instead, it needs vital verifiable attribute information to provide anonymous service to the owner of the attribute. Services are further classified into authentication and authorization. An authentication service is an interaction of approving the attribute's identity by confirming whether the set of attributes fulfils the conditions given by the smart contract. Then again, authorized service is authorizers created through smart contract must be able to sign attributes. Clients can provide others with the option to utilize a specific attribute through the authorizer or reuse the privilege after the trust between entities finished. Researchers used the Ethereum account address to represent the identity of the user. When an identity is made, they utilized attributes to represent the information contained in an identity. Fields of an attribute incorporate hash, value, reputation, grantee and certificate. After attribute creation is done, it's related to two services: authentication and authorization. The proprietor of the identity, for the most part, gives the read right to different identities by permitting them to access attribute values. For instance, a customer can enable a bank to access his name and ID information to create an account for him. In the proposed framework to achieve this situation, the operation of attribute citation is required. A reference request should have been dispatched from the reference or caller to the smart contract. The smart contract, at that point, checks the caller's identity and sends the reference request to the attribute owner. The attribute owner at that point concludes whether to allow the attribute to the referencer. If the attribute owner consents to enable the attribute to, attribute authorization finished. A client can make his own attributes himself. However, it doesn't have to pass the check.

As an example, let suppose Bob creates an attribute containing education qualification. When the job seeker needs to confirm his academic record, Bob needs to allow this attribute. Such attribute can't be checked unless an educational institution confirms that this attribute is legit. The proposed attribute signature can resolve this problem or issue. When attribute requires to be certified by one or multiple parties, their signatures are required. The certificate field of attribute records signature information whenever the signer needs to get verified one can look for the signer address stored in this field. However, there is the problem with this part, the user can create multiple identities over blockchain at no cost, resulting in false identities hence a risk of less trustworthiness because of the system's decentralized nature. Therefore, to solve this problem identity must be associated with reputation to construct a reputation-based identity management model. To solve the effect of Sybil identities, Attribute Reputation Value (ARV) is used. It changes dynamically as soon as the number of interactions increases. . To avoid attribute value getting frequently changed every time attribute is modified ARV value reduced. Also, the limit is set for the number of times attributes can modify. When an identity authorizes an attribute to another identity trustworthiness is set up between them. To improve Rep, the identity should be reliable in interactions with various identities. Subsequently, it takes care of the issue of brush reputation and in this manner makes ARV really persuading.

Marcela T. de Oliveira et al. [54] designed Blockchain Reputation-Based Consensus (BRBC) system in which a node should have a reputation score higher than a given network trust limit before being permitted to embed another block in the chain. A randomly chosen set of judges will notice the node's conduct and appropriately update the node reputation score. BRBC embraces network maturity as a criterion of initial reputation; the longer the node stays in the network, the more possibility of turning into a miner. The consensus issue is officially characterized by three properties: validity, agreement and termination. Validity guarantees that if all the correct processes propose the same value, then any valid process converges to this value. The key challenges for a reputation based blockchain are to find a way to achieve the following:

1. Make distributed reputation score scheme for miners dependent on a trust limit
2. Monitor activity of miners.
3. Respond to malicious measures, including provisioning assets to assess and withdraw the authorization of miners.
4. Keep a list of authorized miners that can change in time and arrange a rotation mining scheme.
5. Perform network access control and self-organization for the number of authorized miners as per the network's size.

BRBC consensus mechanism is divided into the particular number of steps these incorporate Access control (further grouped into three phases: generation of asymmetric keys public for node ID and private key to finish paperwork for node activities, a transaction in progress should be approved by the miner who signs it and advances it to the network through another node exchange when effectively mined it become part of the blockchain and all nodes will add the public key to the list of identification keys.), Miner selection (occurs automatically when miners are needed self-selection happens, i.e., Me >M, e is the average number of miners more prominent than current miners. The number of judges expected to guarantee legitimate network activity and screen miners relies upon expected resilience thinking about the number of malicious nodes, expected time to expel malicious node and accuracy of trust deposited in the monitored node. To mitigate judge selection manipulation, a node should not have the option to distinguish its appointed judges ahead of time. Researchers embrace a non-reversible pseudo-random choice of judges dependent on the Bloom filter utilizing the public key of the miner node). After jury selection, each judge keeps up data about its observed miners. The miner public key added to the trust table, and the underlying reputation score is appointed to it. In a naive approach, a new miner gets a high reputation score significantly higher than the threshold on the other hand in suspicious approach judges don't anticipate that new miners should be trustworthy consequently allot a reputation score equivalent to the threshold. A miner should have a good reputation to take part in the network. The voting favouring the expulsion of malicious nodes from a network happens if the reputation score drops down to a specific threshold value.

The mechanism for expelling a malicious node has two phases voting and expulsion. All judges independently issue voting transactions for the nodes. At that point, every one of these transactions is approved and mined into a block. To expel a node for a network, more than half of the judges' expected number should concur that the node has fallen under a reputation threshold. For expulsion, the appointed authority who sent the oldest vote should send expulsion transaction. The expulsion transaction is then approved and mined into a block. The validation process incorporates verification of the signatures of judges and the number of signatures. When a block is mined, all nodes need to refresh their view and eliminate the address of the expelled node, disposing of its public key. The significant thing to note is that if a removed node re-joins the network, it must re-join with another key pair, and it will join as a participant, not as a miner.

Sidra Malik et al. [35] highlight trust-related problems in the supply chain industry and try to overcome this with the help of reputation assignment to the parties to inculcate trust in the system. They proposed a TrustChain system, a three-layered trust framework: Data, Blockchain, and Application layer. The goal is to achieve the following:

1. Reputation model that assesses the quality of commodities.

2. Assignment of product-specific reputations for same participants.

3. Utilization of smart contracts for transparent, effective, secure and automated calculation of reputation scores.

4. Minimal overhead in terms of latency and throughput contrasted with other straightforward blockchain-based supply chain models.

The proposed framework's information layer encompasses supply chain information delivered by the sensor gadget, trade events between entities, and regular endorsements. The raw data can be stored in the database at the application layer (i.e., off-the-chain), while the message digest of data is shipped off the blockchain layer as transactions. At the blockchain layer, transactions stored on the ledger and the processed following set of access rules define by Access Control List (ACL). The entrance rules indicate who can peruse or compose the information on the record. The transactions invoke smart contracts, which create a reputation and trust module. The reputation and trust values put away on the digital profiles of supply chain entities and items on the blockchain. Ultimately, the application layer associates with the blockchain layer through queries. Additionally, administrators and regulators can question the trust and quality scores of entities and commodities. In light of the retrieved scores, they action rewards and penalties, which reimburse the entities with high scores by publishing scores, punishes the entities with a low score by revocation through the network and distribute the eventual product ratings for clients.

The proposed framework TrustChain is permissioned blockchain network uses hyperledger fabric for deployment. The smart contract further classified into the quality contract and rating contract. Quality Agreement enlists the quality rating measures, for example, temperature thresholds, boundary thresholds (maximum upper and lower bounds of required temperatures in which commodity is safe) and damage thresholds (temperature thresholds exceeding which result in complete spoilage of item). The quality contract generates two outputs: warning notifications and reputations score of the commodity. The commodity reputation scores over and again refreshed with each exchange event as the commodity moves through the item chain till it arrives at the last retailer. Rating Contract: For exchange between purchaser or vendor, the rating contract is invoked to compute the vendor's reputation with three inputs: reputation score of traded commodities, regulator's rating for vendor and purchaser's rating for the vendor—the reputation of the vendor at that point registered as a weighted aggregate. When the smart contract calculates the reputation, a reputation model is picked dependent on the aggregation function(mean, middle or beta-reputation). Researchers adopt the time-differing and amnesic trust score estimation that adjusts to supply chain events, where the recent events given higher weights than older events. If another trader joins the network, starting trust score allocated to him is the base score needed to participate in the network. The trust score estimation includes two stages: one is computing the trader's general reputation score dependent on the current and past reputation scores, and the other is calculating the trust score dependent on the overall reputation score and other application-specific features. Results of their simulation indicated minimal overhead in terms of latency and throughput is required for this system in comparison to other simple blockchain-based supply chain models.

CHAPTER 3

Methodology

In this chapter, we describe our new crowdsourcing LMD scheme that leverages blockchain and smart contracts. First, we describe the system design and architecture. Then, provide detailed descriptions for the main operational scenarios of the proposed scheme. Finally, we compare our scheme to existing work in the literature and provide an informal security analysis for our scheme to demonstrate the scheme is resilient to known security attacks and satisfies the security and trust requirements.

3.1 System Overview

The system objective here is to achieve a last-mile decentralized system that leverages the benefits of blockchain immutability and reputation based crowd-sourcing. Each party in the system contains a certain level of trust based on which the system's user can choose the consensus algorithm. For instance, if the system's parties have zero-trust in each other, then the POW algorithm is most commonly accepted algorithm; however, if most of parties in the system trust each other, then Proof of Authority (POA) consensus algorithm is more acceptable. I made certain assumptions while designing this system: each party is working in a trust-less environment, hence POW used as a consensus algorithm, and reputation of parties are reliable, meaning no reputation based attacks are possible in such a system.

The proposed solution uses the Ethereum blockchain to create a decentralized system, builds trust, and uses immutable logs and events. This type of solution reduces or eliminates the need for an intermediate agent or broker to sell physical items with proof of delivery. The roles of Ethereum entities in the smart contract are as follows:

- **Seller**: It initiates the delivery process by asking for carrier assignment. It is responsible for handing over the item to carrier, assignment of a carrier and provide ratings to the carriers.
- **Buyer:** It is the entity that shows interest in buying an item.It provides the details for specific carrier handling conditions required for purchased item and responsible for providing ratings to the carriers.
- **Carrier:** It is responsible for delivery of an item to its correct buyer. Its ratings and selection policy are handled by *rating contract*.
- **Arbitrator**: It is a trusted third party agreed by each entity in the delivery process who is responsible for resolving disputes off-chain, reaching consensus on weights assigned to carrier and to conduct few last-mile operations .

The delivery of a product is achieved with multiple smart contracts, facilitating the automation of process and support in saving all transactions' history without alterations. A carrier assigned by a *rating contract* is based on carriers' selection policy 3.2.3. All other parties get notified about the status of the item during delivery. An agreed upon and trusted arbitrator by each party is also part of the main POD contract and can only step in in-case of dispute. Each of the mentioned participants possesses an Ethereum address, and out of these seller and buyer entity contributes to the carriers' rating.

Figure 3.1.1 shows overall system overview depicting main participating entities in a successful transaction.The main delivery contract follows a chain of events that each participating entity should follow to preserve everyone's right. All actions that take place off the chain are accompanied by functions in the contract that trigger logged events and notifications. The main delivery contract invokes the rating contract, which holds the pool of carriers' rating profile. The rating contract is responsible for selecting a carrier by calculating reputation metric (see section 3.2.2) based on the

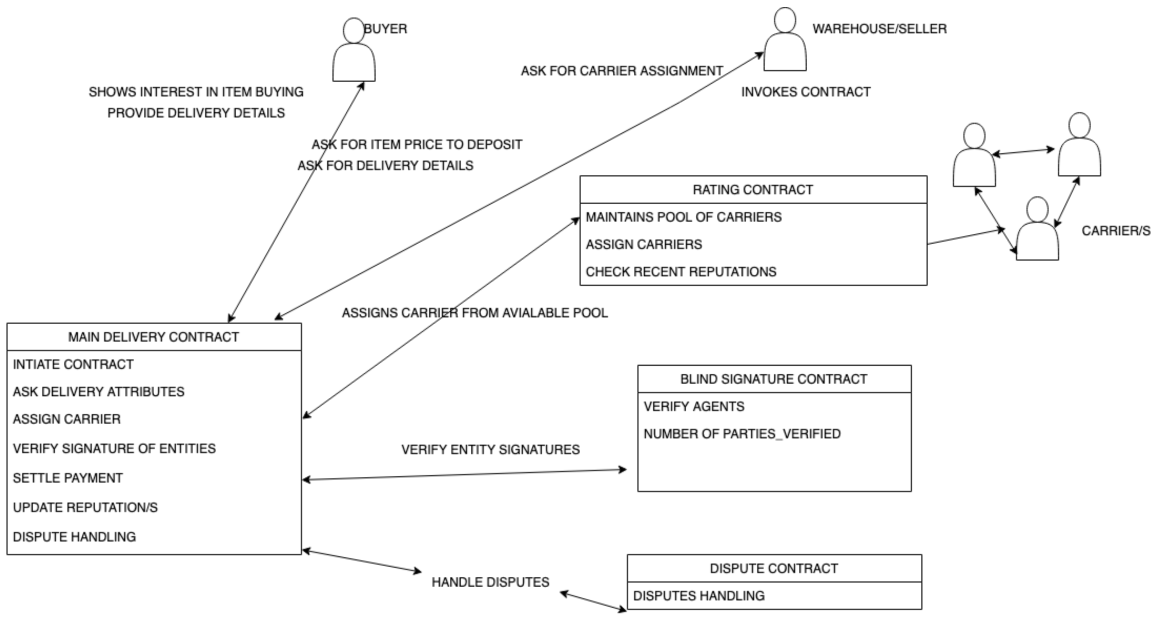


Fig. 3.1.1: System Overview of the POD solution showing the main participating entities participating in a successful transaction

previous delivery outcomes. In case of dispute, the main delivery contract invokes a dispute contract that contains all the commonly occurred conflicts. For verification of signatures, a blind signature contract deployed, which provides the seller's ability to send a blinded message to the buyer to obtain its signature and later verify the buyer's authenticity.

3.2 System Design

There are certain limitations discovered while replicating and analyzing the state-of-the-art blockchain scheme for delivering physical assets are [24]. First, scalability challenge i.e. deposit twice a collateral is not feasible for carrier/s because according to some articles in New York Times in US alone 1.5 million parcels deliver per day [23]. It is estimated that on average a carrier delivers 800+ deliveries in a day [41]. Second, verifiability of agents challenge which involves major flaw in verification of the keys, either seller or carrier can be a fraudster i.e., there is a high chance if any of the party readies to breach the system and share keys hash of each other bashes the trust of the current system. Third, arbitrator role not explicitly defined in the existing POD work and every-time dispute occurs it goes to arbitrator and resolves off-chain which incurs unnecessary cost on overall delivery process. To overcome this set of challenges POD system proposed that leverages blockchain immutability and crowd-sourcing techniques.The execution of a delivery is divided between set of contracts and role of each contract is defined as follows:

- **Rating Contract**: The smart contract that ensures selection of carriers under various different possible scenarios and assigns the one best suited for a specific delivery. It also maintains the profile of carriers, update reputation scores,penalizes or reward carriers based on their successful deliveries.
- **Dispute Contract**: The smart contract that handles the dispute under predefined dispute scenarios that may likely to happen, if still dispute exists then it is solved off-chain by the arbitrator.
- **Main Delivery Contract**: The smart contract which acts as the starting point of the process flow and executes the last mile delivery operations like creating a package, verification of entities, assigning carrier and invoking dispute contract etc.
- **Blind Signature Contract**: The smart contract responsible for generation and verification of blind signatures between two parties(seller and buyer). It has function like verify signer function and parties verified function details of which are attached in appendix 5.2 and explained in section 3.2.4.

3.2.1 System Initialization

A minimum reputation score $\boldsymbol{s}$ is assigned for each new carrier that joins the system. Apart from that, a carrier pool with reputation scores are maintained by rating contract to ensure n carriers with varied reputation score exist in the pool before receiving a delivery request. This is done to ensure every-time a delivery request comes the selection policy must be met (section 3.2.3) and carrier must be assigned to execute delivery.

3.2.2 Reputation

Reputation plays an essential role in building trust in the system because each participating entity has a reputation value which reflects the trustworthiness of the attribute(item) exchange. To overcome challenge of reserving collateral from each party in terms of crypto-currency and to establish trust between two parties, it is essential to have a reputation based network that leverages the benefits of blockchain immutability and provides a confidence score to accomplish a deal between two parties. The details about the calculation of this metric are as follows:

Reputation Score: For the carrier's reputation score calculation, take the summation of reputation score provided by buyer and reputation score provided by seller for that very carrier. This can be expressed mathematically as:

$$Rep_{carrier} = Rep_{buyer} * W_1 + Rep_{seller} * W_2$$

Note: Rep stands for reputation. W_1, W_2 are weights set by the arbitrator.

The proposed system also provides liberty to customize delivery by allowing the buyer to provide some special delivery handling conditions, for instance: guarantee delivery temperature conditions between (5 - 10) Fahrenheit. In that case, the sensor rating can be beneficial. The sensor rating is optional here depending on the type of physical asset to be delivered. This can be expressed mathematically as:

$$Rep_{carrier} = Rep_{buyer} * W_1 + Rep_{seller} * W_2 + Rep_{sensor} * W_3$$

Note: Rep stands for reputation. W_1, W_2, W_3 are weights set by the arbitrator. The overall reputation score is computed as summation of all successful transactions over a period of time :

$$Rep_t = \sum_{t=t_0}^{t=t_n} Rep_{carrier(t)}$$

Note: Rep_t refers to the total reputation of a party over a period of time.

If an agent is the initial user to the system then every initial user is assigned a Rep_{min} score to participate in the system. This score is set by arbitrator within the delivery scenario. The main significance of Rep_{min} is to assign a carrier and to provide equality to each participating carrier. To understand how this Rep_{min} is used in carrier selection policy see section 3.2.3.

3.2.3 Carrier Selection Policy

Selection of carriers also plays an important role in building trust to the system. It not only avoids the possibility of greedy selection of carriers(i.e the carriers having a greater trust score is always selected) but also provides a chance for low reputation score carriers to equally participate in the system, I hereby follow a biased selective approach. In order to understand how selective biasness works let us consider following scenarios:

3.2.3.1 SCENARIO 1

If an item requested by buyer contains custom delivery requirements for example change in delivery location, seller will first check the pool of available carriers reputation, the priority is giving to lower reputation score carrier but **reputed carriers** (meeting minimum reputation score criteria). In case all carriers are below the minimum threshold(Rep_{min}) then wait for reputed carriers to join the system.

3.2.3.2 SCENARIO 2

If an item requested by buyer has no custom delivery requirements, check for the reputation of carriers from the available pool. If any carrier is meeting the minimum reputation score then that is eligible for delivery. However, if system has multiple carriers that are eligible for delivery, ask for the carriers to bid for delivery.

If the bidding amount is above minimum reputation score then carrier having lowest bidding score is assigned the delivery. If in case multiple carriers are having same bidding scores then the carrier who bid first is considered.

3.2.3.3 SCENARIO 3

If the bidding score didn't solve carrier selection dispute and there reputation is also same, then choose the carrier with highest priority but biased selection for example 70% times the carrier with high reputation score is selected and 30% times the carrier with low reputation score is selected. In this way low reputation score carriers are encouraged to participate and it will avoid the fraud or monoply of reputed carriers. Using a proportional selection strategy (aka roulette wheel selection) or tournament selection based on the carrier reputation score enable our approach select one ore more candidate carriers with the minimum required reputation score.

Thus, the carrier's assignment largely depends on customer delivery attributes and the available pool of carriers. It not only provides equality but also brings trust to the system.

3.2.4 Signature Verification of Agents

The existing POD systems are incapable to cope with the defect in the verification of agents.To verify whether a last mile carrier reaches to a correct destination they use simple blockchain key hashes without taking into account who enters that hash.To overcome this limitation and to escort accountability blind signatures concept are introduced on top of blockchain layer.

Figure 3.2.1 depicted the typical process flow used to verify two parties thus it is executed when buyer sent the message to the smart contract that 'package has been received' then seller sent a blinded message for buyer to sign so that seller verifies weather the buyer is genuine or not. Contract listening script executes and listens to verification events invoked by blind signature contract functions on-chain and as soon as event invoked it in turn calls blind signature script containing details of verification

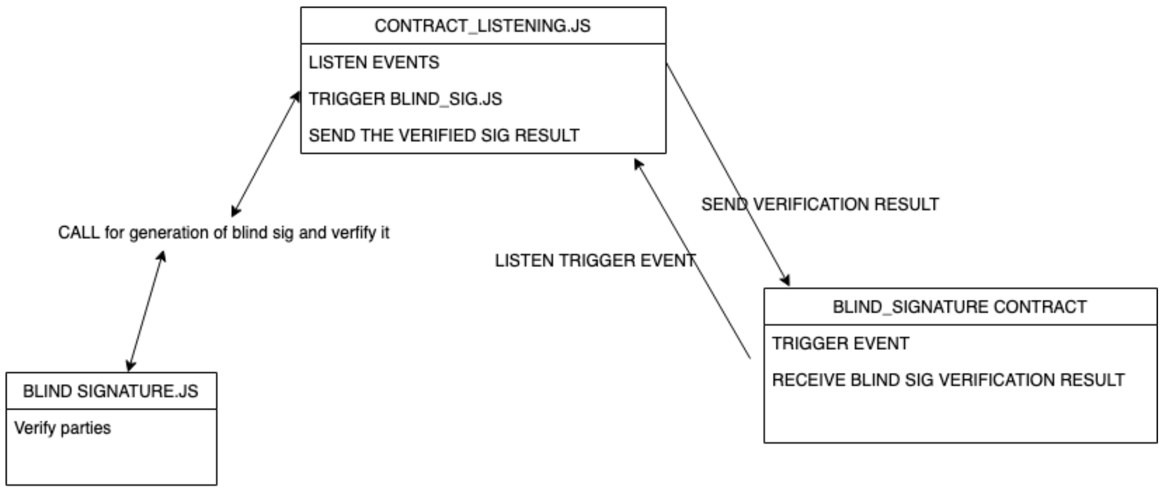


Fig. 3.2.1: Generation and verification of blind signatures in proposed system

steps.

Steps involved in verification process of agents in proposed model are:

1. Generate private key for buyer and seller using already implemented RSA based blind-signatures packages API. [29].
2. Buyer wants seller to sign a message without revealing its content.
3. Buyer gets n and e variables from seller's public key. For significance of n and e public values in key generation refer [47]
4. Buyer blinds message using these n and e values.
5. Buyer sends blinded message to seller.
6. Seller signs blinded message.
7. Seller sends signed message to buyer.
8. Buyer un-blinds the message using seller's signed message, n and r values and sent to main delivery contract for verification.
9. Main Delivery Contract verifies this signature by taking into account n, e, un-blinded signature (prev step) and buyer's message sent to seller.

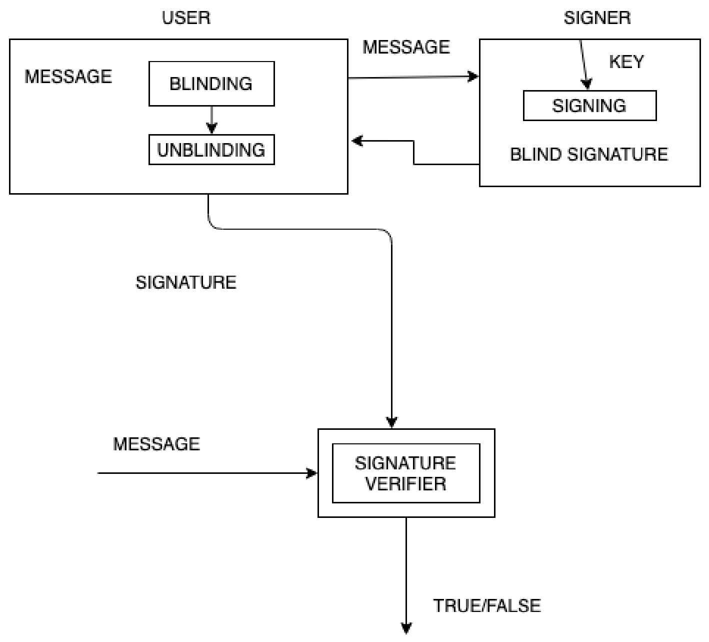


Fig. 3.2.2: General Blind Signature Process

10. Main Delivery contract also verifies weather seller did sign it or not via unblinded signature , seller's public key and message values.

11. Boolean results are generated- i.e both parties verification resulted true then settle payment executed and obtain proof of delivery.

This type of verification process eliminates the need of actual buyer to be present at the time of delivery because anyone on behalf of buyer can show the signed token from seller to carrier and obtain the item from carrier. For understanding of math involved in blind signature algorithm or steps refer [29]. Details of contract listening script, blind_signature contract functions and blind signature script are added to appendix 5.2.

3.2.5 Delivery Execution

Steps involved in delivery execution are:

1. Buyer shows interest in buying an item.

2. Main Delivery contract sent request to buyer to pay item's price.

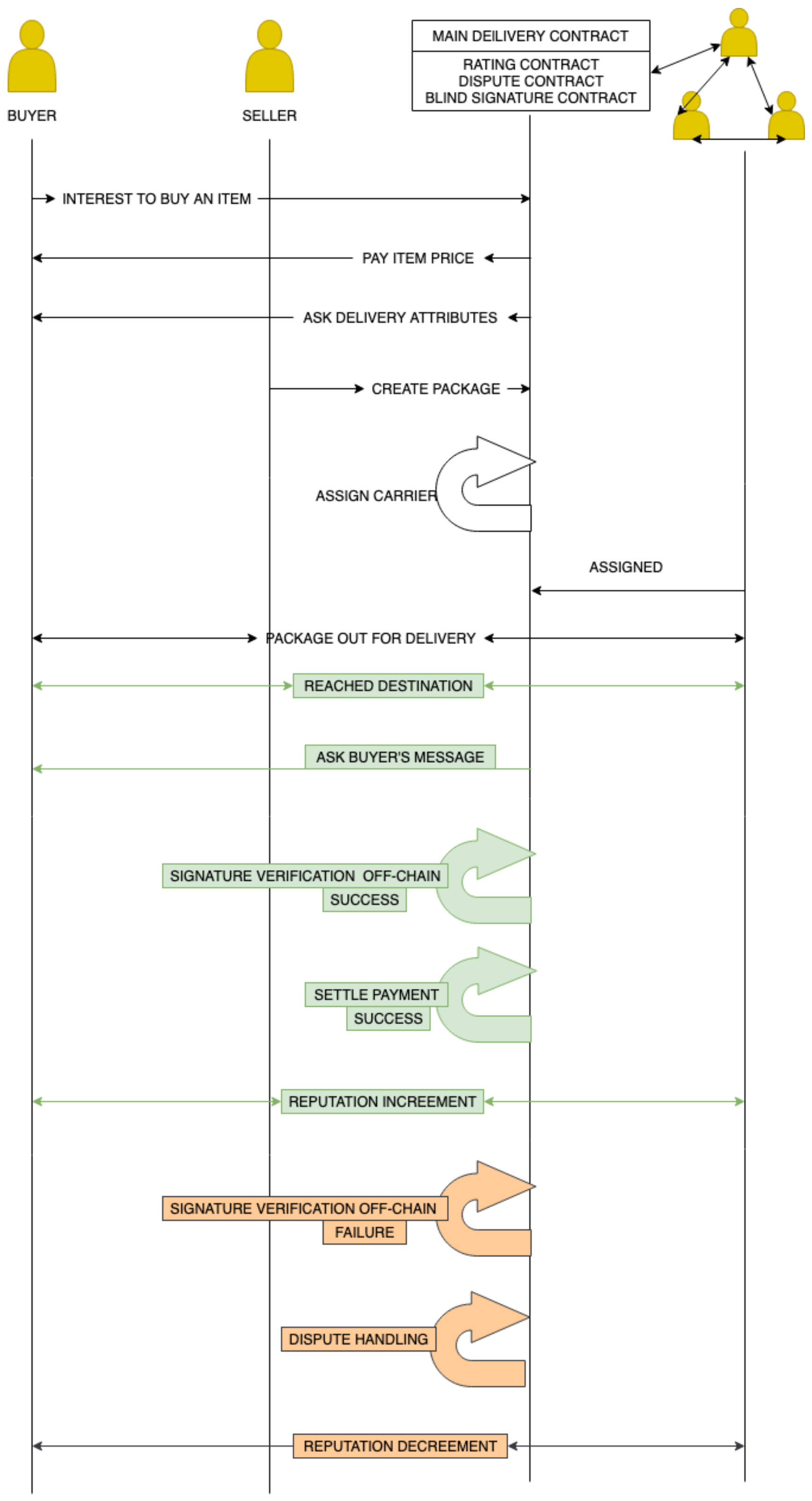


Fig. 3.2.3: Sequence diagram of the smart contract code that shows the flow for a successful and un-successful delivery execution

3. Main Delivery contract ask for any customize delivery attributes request from buyer like special package handling conditions or location.
4. Seller receives delivery request, create package, request to assign carrier and sent the notification to the contract.
5. Main Delivery contract assigns carrier with the help of rating contract's carrier pool and carrier selection policy mentioned in section 3.2.3 .
6. Assigned carrier notifies each party that package is out for delivery meaning item is on the way to its destination.
7. Assigned carrier notifies each party that he/she reached destination and delivered package.
8. Main Delivery contract ask for buyer's message that needs to be blinded before sending to seller for signature.
9. Message blinded,sent to seller and signatures verified via off-chain interactions refer section 3.2.4
10. Settle payment between various parties executed.
11. Reputation score(see section 3.2.2)increment after successful delivery and it can be checked via some helper functions defined within the contract.
12. In case signature verification failure happens then smart-contract dispute handling function invokes.
13. Dispute handling function consists of set of possible conflicts and steps that are executed under each conflict.
14. Reputation score decrements after un-successful delivery and it can be checked via some helper functions defined within the contract.

Note : The steps shown in green are indicative of successful delivery execution and steps shown in light red color are indicative of un-successful delivery due to possible disputes. Also, there can be many possible disputes that can happen (see algorithm 3.3.6). Figure 3.2.3 only reflects no signature verification match dispute.

3.3 Operational Scenarios

In this section, algorithmic details related to key last-mile delivery operations such as carrier assignment, verification of receiving entity's signatures, payment settlement , reputation building and handling of disputes in proposed POD system are elucidated.

3.3.1 Carrier Assignment

Algorithm 3.3.1 shows the algorithm for assigning carrier via a crowd-sourced pool of carriers. The process of carrier assignment is two-fold, algorithm 3.3.1 is the caller algorithm which calls algorithm 3.3.2, checks the state of the contract, sets the cancellation state and creates notification after successful assignment whereas algorithm 3.3.2 is callee algorithm which implements the carrier selection policy, calculates the reputation score metric (described in section 3.2.2) of selected carrier and returns the address of selected carrier. If no available carrier is assigned , it returns no carrier assigned and notifies each party that no carrier is available to satisfy the carrier selection policy. In that case, caller algorithm 3.3.1 sets the type of dispute variable to 1, changes contract state to dispute and executes dispute algorithm 3.3.6. Within dispute algorithm, carrier selection dispute function defined that takes care of carrier selection policy and assigns a new carrier to a delivery from available pool of carriers in case no carrier is assigned.

3.3.2 Verify Signatures

Signature verification happens every time the delivery item handed over to another participant. The receiving entity (buyer) first confirms that the delivery carrier has arrived at the destination.Then, buyer send the message off-chain to blind the message and executes chain of events by invoking blind signature contract's verification

Algorithm 3.3.1 Main Delivery Contract: Carrier Assignment

```
Require: E, item price,contractState, cancellation state, carrier, type_of_dispute
 1: E: the set of Ethereum addresses/entities participating in this contract.
 2: carrier: the address variable set by this algorithm
 3: Restrict the access to only seller address s ϵ E in the contract
 4: procedure assignCarrier()
 5:     if contractState ==packageCreated then
 6:         carrier ←assigned by rating contract's assignCarrier function    ▷ see algorithm 3.3.2
 7:         if carrier!=null then
 8:             set the cancellation state for s ϵ E as false
 9:             set the cancellation state for carrier ϵ E as false
10:             create a notification for package hand over to carrier after successful assignment.
11:             contractState ←package handover and carrier assigned
12:         else
13:             Set type_of_dispute to 1
14:             Revert contractState to dispute.
15:             Execute dispute.    ▷ see algorithm 3.3.6
16:         end if
17:     end if
18:     return carrier
19: end procedure
```

Algorithm 3.3.2 Rating Contract: Carrier Assignment

```
Require: pool of carrier's previous ratings, CSP(carrier selection policy)
 1: carrier: the carrier address set by this algorithm
 2: Access restriction is handled by contract calling this procedure
 3: procedure assignCarrier()
 4:     i ← assign carrier location    ▷ returned by rating contract's carrierSelector function
 5:     reputation_score ←calculate reputation score    ▷ see section 3.2.2 for reputation score calculation
 6:     carrier ←assign address of carrier at location i
 7:     if i!=no carrier assigned then
 8:         create a notification that CSP criteria met
 9:         check carrier assigned at address i has met minimum reputation_score criteria
10:     else
11:         create a notification that CSP criteria not met
12:     end if
13:     return carrier
14: end procedure
```

function.These events include sending blinded token to seller who then signs it and send back signed token to buyer. The buyer give signed token to to blind signature script which verifies and returns the number of parties verified through off-chain interaction. In case the number of parties is not equal to two, it sets the type of dispute variable to 2, change contract state to dispute and execute dispute algorithm 3.3.6. Algorithm 3.3.3 shows full details of how the function that does signature verification works in code.

Algorithm 3.3.3 Main Delivery Contract: Verification of Signatures

```
Require: E, message_to_sign, contractState, VerifiedParties, type_of_dispute,
    BS_Contract
 1: $E$: is the set of Ethereum addresses/entities participating in this contract.
 2: VerifiedParties: a variable used to store number of parties verified.
 3: Restrict the access to only buyer address s ϵ E in the contract
 4: procedure AskForBuyerMessage(message_to_sign)
 5:     if contractState ==ArrivedToDestination then
 6:         call BS_contract's verify(message_to_sign) function
                                    ▷ returns verification event logs initiated off-chain
 7:         $VerifiedParties$ ←assign the number of parties
                    ▷ Parties returned via BS_Contract's parties_verified(message_to_sign)
    function
 8:     else if VerifiedParties!=2 then
 9:         Set type_of_dispute to 2
10:         Revert contractState to dispute.
11:         Execute dispute                          ▷ see dispute algorithm 3.3.6
12:     else
13:         Revert contractState and show error.
14:     end if
15:     return VerifiedParties
16: end procedure
```

3.3.3 Payment Settlement & Update Reputation Score

When the signature verification process is successful, the delivery item reaches its correct buyer by a carrier. Algorithm 3.3.4 shows the process of payment settlement wherein contract state is verified first then n amount of the item price store in smart-contract transferred to seller's account and m amount of stored item price transferred to carrier's account for its services. Once payment settlement happens, the carrier's reputation score increments, and this procedure returns the updated reputation score for future deliveries. Algorithm 3.3.5 use this updated reputation score and send it back to the rating contract for updating the carrier's current ratings. Whenever this specific carrier picked next time by carrier selection policy, its new reputation score is considered.

Algorithm 3.3.4 Main Delivery Contract: Settle Payment

```
Require: E, seller, buyer, carrier, item price, contractState , RT_Contract,M,N
 1: $E$: the set of Ethereum addresses/entities participating in this contract.
 2: item price: the price stored in contract deposited by buyer for purchasing an item.
 3: M: Carrier fee
 4: N: Seller fee
 5: Restrict the access to only arbitrator address a $\epsilon$ E in the contract
 6: procedure SETTLE_PAYMENT()
 7:     if contractState ==BothPartiesVerifiedSuccessfully then
 8:         $M$ ←transfer m amount of item price to carrier's account.
 9:         $N$ ←transfer n amount of item price to seller's account.
10:         $contractState$ ←PaymentSuccess
11:         check for Rep_score of assigned carrier and increment it.
                        ▷ it is done via RT_Contract's reputationCheck(carrier) function
12:     else
13:         Revert contractState and show an error
14:     end if
15:     return carrier's updated reputation score
16: end procedure
```

3.3.4 Dispute Handling

Disputes are always challenging to handle and incur an unnecessary cost to the whole delivery process. Solving a dispute off-chain is still a challenge for arbitrator because many factors need to be considered: the delivery time, the number of deliveries allocated to a carrier, carrier selection policy breach, etc. A hybrid approach can be beneficial to avoid this challenge, i.e. both smart-contract-based dispute handling and off-chain dispute handling. For smart-contract-based disputes handling all commonly occurred disputes, conflicts function defined in a contract(see appendix 5.2).Algorithm 3.3.6 shows full details of how dispute handling works in code. For instance, if the type of dispute is level 1 then this algorithm calls dispute contract's carrier selection

Algorithm 3.3.5 Main Delivery Contract: Update Reputation Score

```
Require: E, rep_score, carrier, contractState , RT_Contract
 1: E: is the set of Ethereum addresses/entities participating in this contract.
 2: carrier: variable containing address of carrier c ϵ E.
 3: rep_score: variable used to store reputation score in the contract.
 4: Restrict the access to only arbitrator address a ϵ E in the contract
 5: procedure updateReputation()
 6:     if contractState ==PaymentSuccess then
 7:         send updated rep_score to RT_Contract's setTarget(rep_score,carrier) function
 8:     else
 9:         Revert contractState and show an error
10:     end if
11: end procedure
```

function which contains the steps of assigning a new carrier by taking into account the carrier selection policy. If in case dispute contract unable to solve a dispute, the off-chain approach adopted where arbitrator audits the transaction logs to see a point of failure and party responsible for it. Thus, this approach saves time in solving a dispute and avoids atleast some cost incurring on the last-mile delivery process.

3.3.5 Cancel Delivery and Refund

The seller, buyer and or carrier can cancel delivery only under specific conditions. Algorithm 3.3.7 shows the cancellation algorithm that happens through the main delivery contract. Input needed for the algorithm is the address of the caller, cancellation state and item price. The cancellation state is a boolean false if the item handed over to the carrier. Any of the mentioned entities in the smart contract can decide on cancelling the delivery request. For example, the buyer and carrier can cancel delivery as long as it isn't on its way to the destination. The seller can cancel as long as the delivery item not handed over to the carrier. If the item is getting delivered, nobody can cancel the exchange. Once, cancellation state checked refund is initiated for the buyer and notification is sent to each party about delivery cancellation.

Algorithm 3.3.6 Main Delivery Contract: Dispute Handling

```
Require: E, caller,buyer, carrier,arbitrator, contractState, type_of_dispute,
    DS_Contract
 1: E: is the set of Ethereum addresses/entities participating in this contract.
 2: procedure disputeHandling()
 3:     check for contractState as dispute
 4:     check for type of dispute.
 5:     if type_of_dispute==1 then
 6:         call for DS_Contract's carrierSelection() function
              ▷ contains details of resolving carrier selection dispute using pool of carriers.
 7:     else if type_of_dispute==2 then
 8:         call for DS_Contract's NoSignatureMatchFound() function
              ▷ decrements reputation of carrier and hold the item until correct proof of
    buyer is provided.
 9:     else if type_of_dispute==3 then
10:         call for DS_Contract's CarrierTimeExceeded() function
              ▷ decrements the carrier reputation for not delivering item on time and
    initiates refund for buyer.
11:     else
12:         Create notification that dispute can be resolved off-chain by arbitrator.
13:         Revert contractState and show an error.
14:     end if
15: end procedure
```

Algorithm 3.3.7 Main Delivery Contract: Cancel delivery

```
Require: E, caller, cancelation state , item price
 1: E: is the set of Ethereum addresses/entities participating in this contract.
 2: procedure cancelDelivery(reason)
 3:     if caller $\epsilon$ E then
 4:         Check the cancellation state of caller.
 5:         if cancelation state== true then
 6:             contractState ←cancelation and refund
 7:             Return the item price to buyer
 8:             Create the notification about the cancelation and refund of deposited
    item price.
 9:             contractstate ←Aborted
10:         end if
11:     else
12:         Cancelation of delivery denied
13:         Revert contractState and show an error
14:     end if
15: end procedure
```

3.3.6 Exceeded Delivery Time

Punctuality is crucial when dealing with delivery services. Therefore, delivery time contributes to having a decent impression on the buyer and providing good customer service. The buyer has the privilege to decline to take the item if the delivery surpassed the expected delivery time. Algorithm 3.3.8 shows details of the exceed delivery time calculation that permits the buyer to call the function if the current delivery time surpassed the expected delivery time. In the event that delivery time surpassed, it sets the contract state to dispute, sets the type of dispute variable to 3 and executes dispute algorithm 3.3.6

Algorithm 3.3.8 Main Delivery Contract: Exceeded Delivery Time

```
Require: E, caller,buyer, contractState ,current delivery time, type_of_dispute,
    del_window(Delivery Window)
 1: E: is the set of Ethereum addresses/entities participating in this contract.
 2: procedure exceedDeliveryTime()
 3:     if caller ==buyer && contractState== Item on the way to buyer then
 4:         if current time > del_window then
 5:             contractState ←dispute due to exceeding delivery time
 6:             Create notification about cancelation by buyer
 7:             Set type_of_dispute to 3
 8:             Execute dispute.                    ▷ see dispute algorithm 3.3.6
 9:         end if
10:     else
11:         Revert contractState and show an error
12:     end if
13: end procedure
```

3.4 Security Analysis

Every day millions of delivery packages go missing in the United States. An estimate of 1.7 million packages worth $25million has either stolen or lost due to some kind of fraud [37]. 36% of Americans have suffered from package theft [17], and 90% of thieves are never caught [40]. Some recent studies conducted by US Foods indicated that 28% of deliverers admitted they had taken food from customer's delivery[16]. Retailers or sellers face increasingly sophisticated shipping fraud tactics, including re-shipping frauds [51] i.e. once a package is en-route, the fraudster logs in to modify the delivery address and reships it to their address. Stolen or compromised shipping accounts are well-known fraud these days wherein fraudsters steal shipping account numbers from consumers or businesses to make unauthorized shipments. Once accounts are compromised, packages are routed to a central location and then distributed to fraudulent sites. The prominent distributor may be a part of the fraud ring or be a fraud victim and unaware that they are shipping stolen goods—such type of fraud categorized as interception fraud.These types of fraud hurt customer loyalty and incur an unnecessary cost to retailers. These scams and fraud tactics used highlight the urgent need and importance of obtaining secure proof of delivery from receiving entity to ensure the delivery done with the correct customer; else, it can result in any of the fraud mentioned above.

By design, our proposed POD system inherits vital security features of blockchain. These features include decentralized trust, integrity, non-repudiation, and availability. Our framework likewise handles authentication and access control through smart contract by utilizing restrict modifiers that permit certain actors to execute functions. Also, by design, the proposed POD framework ensures against Man-In-The-Middle (MITM) attacks and replay attacks as every message exchange is cryptographically signed and time-stamped. Integrity plays a significant part in the no-alteration of crucial information. The proposed POD framework gives the capacity to trace back the history events using transaction logs. Non-repudiation guarantees verification for the identity of the sender. The initiator's Ethereum address for all function calls recorded and part of the transaction logs. In the delivery interaction, all participating entities sign any delivery operation utilizing their public keys. This guarantees that no entity can deny its actions later. Data accessibility isn't a worry as contracts are deployed on the public blockchain and consistently accessible to participating entities for performing various delivery operations.

To ensure the verifiability of the POD token secure blind signatures concept incorporated, it allows the signer to sign a token attached to the delivery item, thus establishing a link between the signed token and deliverable item.Further, to make carriers accountable and discourage fraudulent activities in the proposed system, the concept of minimum reputation score introduced. Each carrier interested in joining the system must maintain a minimum reputation score, which later can be extended into incentive mechanism to motivate the honest carriers in the form of conversion of reputation score into ether. They will remain in the system as long as they meet the minimum reputation criteria. The analysis of various security attributes presented in different POD systems in contrast to proposed POD system is presented in table 3.4.1.

Table values ✓indicates system possess that feature and ✗indicates system does not have that particular feature. *NOT PRACTICAL* value assigned to those features where schemes are proposed but these schemes are not practically feasible. *NOT SCALABLE* assigned to those where features are proposoed but they are not scalable. Each system is analyzed based on common security parameters:

1. Accountability: each participating entity must be accountable for each delivery operation performed by them.
2. Auditability: systematic and independent examination of a delivery state with the goal of determining whether delivery operation is correct (according to the consistency rules) and was continuously correct in the past.
3. Anonymity: the identity of a client who takes part in a smart contract is guaranteed to be kept private. Besides, all the information regarding the shipping is also kept hidden.
4. Fair exchange of services: retailers are paid for the services only when the package is dropped off and approved by a security verification mechanism. The buyer ensured to receive a package if the seller is paid.
5. Protection against customer's early aborts: whenever during the package delivery, parties who tackled their job are ensured to get paid regardless of whether the buyer chooses to cancel the delivery.

6. Authorized pickup: the package is delivered to the expected buyer, and no other entity can successfully claim it.

7. Traceability: solution must be traceable enough to determine the point of failure in delivery process.

System Name	ACCOUNTABILITY	AUDITABILITY	ANONYMITY	FAIR EXCHANGE OF SERVICES	PROTECTION AGAINST CUSTOMER EARLY ABORTS	AUTHORIZED PICKUP	TRACEABILITY
Proposed POD System	✓	✓	✗	✓	✓	✓	✓
Single Carrier System	✓	✓	✗	NOT SCALABLE	✓	NOT PRACTICAL	✓
Multiple Carrier System	✓	✓	✗	NOT SCALABLE	✓	NOT PRACTICAL	✓
Lelantos	✗	✗	✓	NOT PRACTICAL	✓	✓	✗

Table 3.4.1: Security Features of various POD Systems

CHAPTER 4

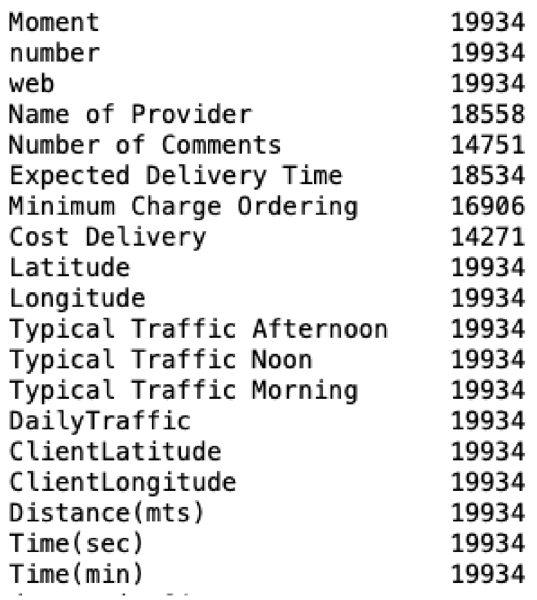
Experiments and Results

4.1 Dataset Used For Experiments

To validate our proposed last-mile delivery scheme, we implemented a proof-of-concept using the Ethereum blockchain. For testing the performance of our proof-of-concept, we used a dataset of online food orders. [43] . The dataset could be downloaded the data repository in [10].

The main attributes of the dataset are shown in Figure 4.1.1. The definition of the different attributes used from the dataset are as follows:

1. Number: indicates serial number of orders per restaurant. For example if order A is from KFC location 1 then value in this column is 1 until the location of order changes.
2. Expected Delivery Time: indicates amount of time expected to deliver an item.
3. Minimum Charge Ordering: indicates minimum charges required to order an item from a specific restaurant. Note some columns have NA values.
4. Cost Delivery: indicates cost to deliver an item.
5. Latitude: indicates latitude at which restaurant located.
6. Longitude: indicates longitude at which restaurant located.
7. Client Latitude: indicates latitude at which client located.
8. Client Longitude: indicates longitude at which client located.



```
Moment                        19934
number                        19934
web                           19934
Name of Provider              18558
Number of Comments            14751
Expected Delivery Time        18534
Minimum Charge Ordering       16906
Cost Delivery                 14271
Latitude                      19934
Longitude                     19934
Typical Traffic Afternoon     19934
Typical Traffic Noon          19934
Typical Traffic Morning       19934
DailyTraffic                  19934
ClientLatitude                19934
ClientLongitude               19934
Distance(mts)                 19934
Time(sec)                     19934
Time(min)                     19934
```

Fig. 4.1.1: Delivery Dataset Description

9. Distance: indicates distance in metres between client and restaurant located. It is calculated as difference of client latitude and longitude vs restaurant's latitude and longitude.

10. Time: indicates the amount of time taken to deliver an item. It is in seconds.

11. web: indicates website from which order took place.

12. Number of Comments: indicates the feedback by clients after their delivery or in case of dispute.

13. Typical traffic: indicates rush hours in the following three categories: free or green traffic (G), average or orange traffic (O), and heavy or red traffic (R). For example, the sequence *R-O-G* means that the typical traffic changes from "red" in the morning to "orange" at noon and "green" in the afternoon, describing a place where traffic conditions improve as time passes.

14. The traffic moment or Moment: is a categorical variable with three possible values: Morning, Noon, and Afternoon. Moment helps to identify the typical traffic, as captured by Google Maps API, around each restaurant during rush hours on Saturdays.

15. Name of provider: indicates the to the commercial name of each restaurant

4.2 Experiments Conducted & Results

To analyze the proposed POD system based on a reputation network, I need to test this system against a real-life dataset (mentioned above). Execute multiple delivery request simultaneously to observe how crowd-sourcing contributes to building trust in the design and how blockchain immutable logs and blind signatures obtain proof that the item delivered to the correct customer. I divide this section into two parts one is experiment design, and the other one is experiment results

4.2.1 Experimentation Design

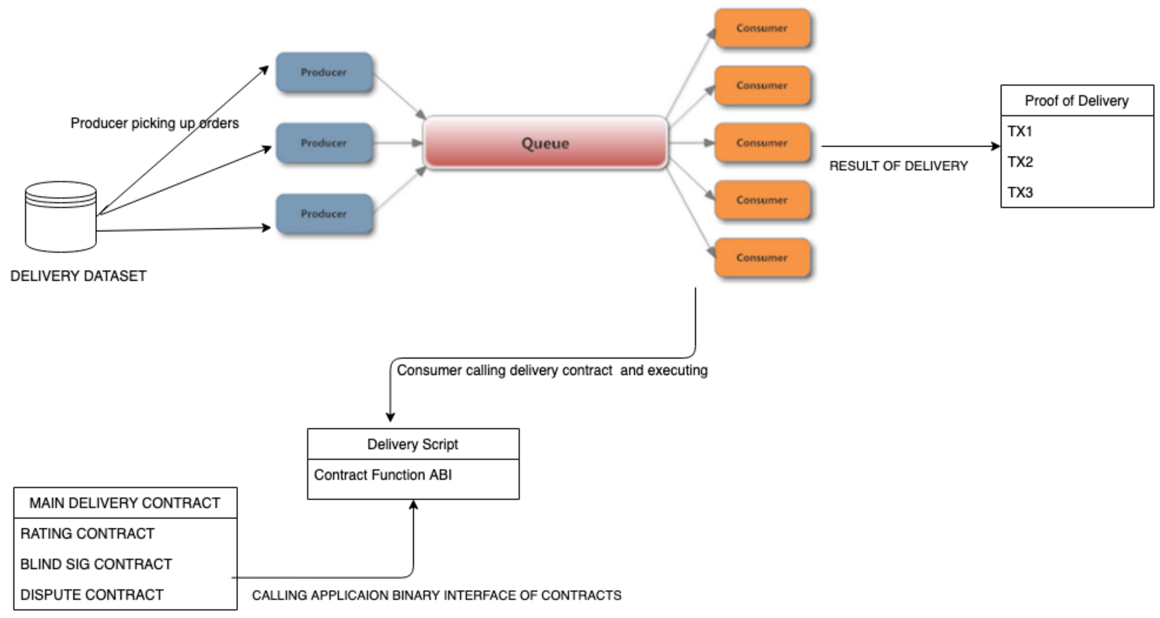


Fig. 4.2.1: Experimentation Design

Figure 4.2.1 explains the experimentation flow adopted to test the proposed system. Steps involved in experimentation are:

1. Create a subset of the dataset mentioned above-containing orders from the top 5 frequently ordered restaurants because I want to test how multiple orders

from the restaurant at a single location executed and helps grow the network of carriers.

2. Create a producer-consumer queue where the producer picks up the order from the dataset and adds it to the queue to consume it.

3. Consumer picks the item from a queue executes the delivery request.

4. The delivery script contains the application binary interface of smart contract code and helps run the deployed contract functions through off-chain interactions.

5. The delivery script is responsible for the carrier's assignment, delivers the item to the buyer, settle payment and increment of reputation score after successful delivery.

6. Now once delivery executed successfully, one can retrieve POD for that delivery and check the updated reputation score of the carrier assigned.

7. The delivery execution process continues until the consumer consumes all the orders in the queue.

Note that producer here refers to restaurants or sellers receiving the orders, and consumers are carriers who are executing deliveries. For a detailed understanding of the producer-consumer problem refer [21]. By performing the steps mentioned above, POD for all deliveries in the dataset are obtained. I analyze carriers' reputation scores in a pool and observe how it varies concerning the number of deliveries. All the findings and results are reported in section 4.2.2 of this book.

4.2.2 Experiments Results

4.2.2.1 Delivery Execution Results:

The variance in reputation scores under the different number of carrier pool N (n set to 5,10,15,20) who are responsible for delivering 258 delivery orders from top

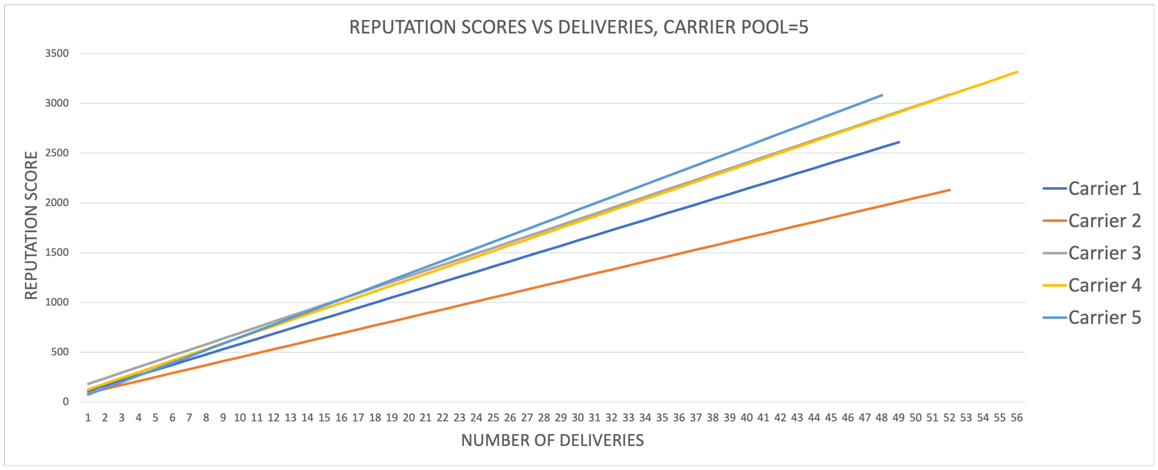


Fig. 4.2.2: Reputation Scores Vs Number Of Deliveries where N (number of carriers) equals 5

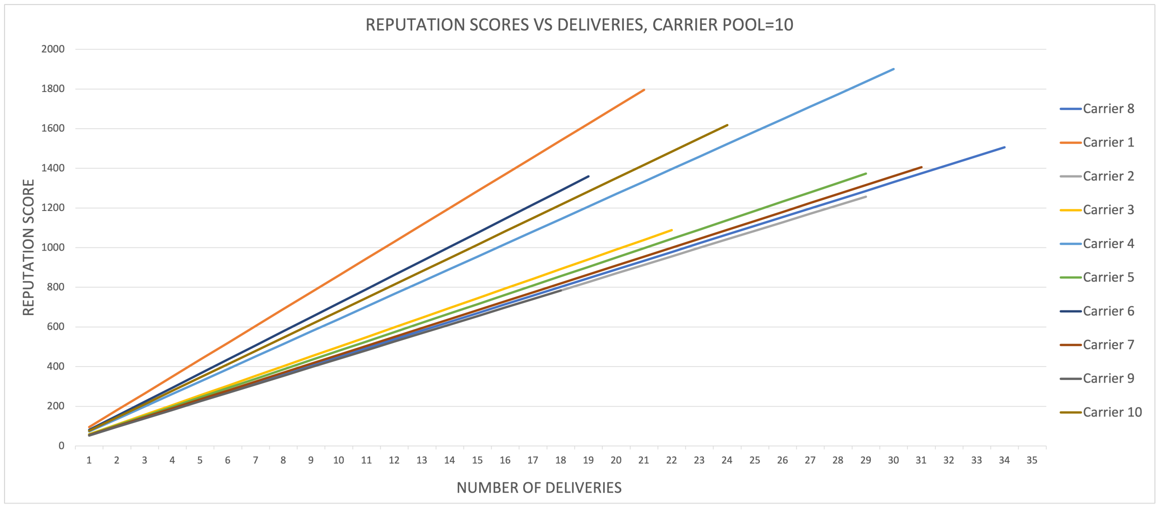


Fig. 4.2.3: Reputation Scores Vs Number Of Deliveries where N (number of carriers) equals 10

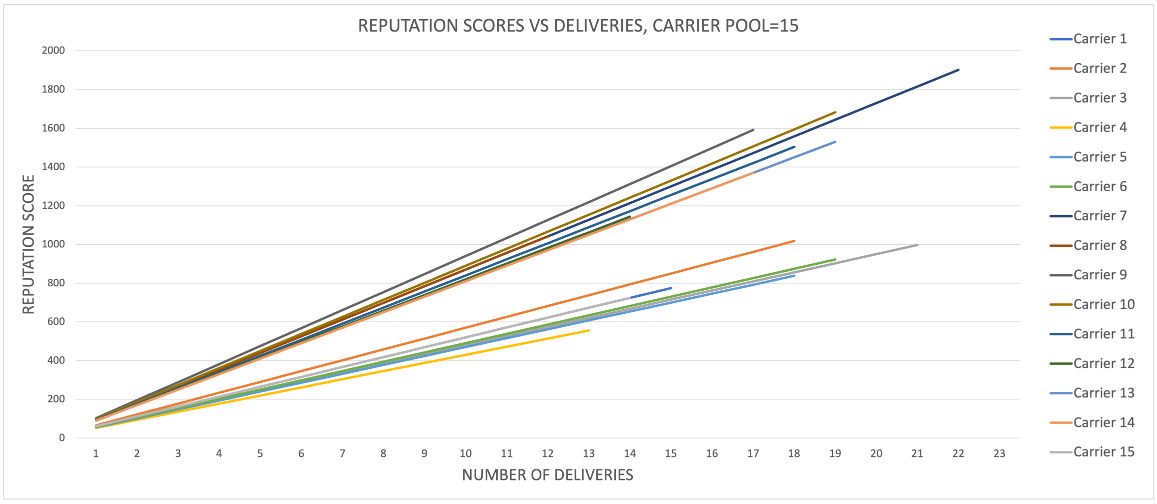


Fig. 4.2.4: Reputation Scores Vs Number Of Deliveries where N (number of carriers) equals 15

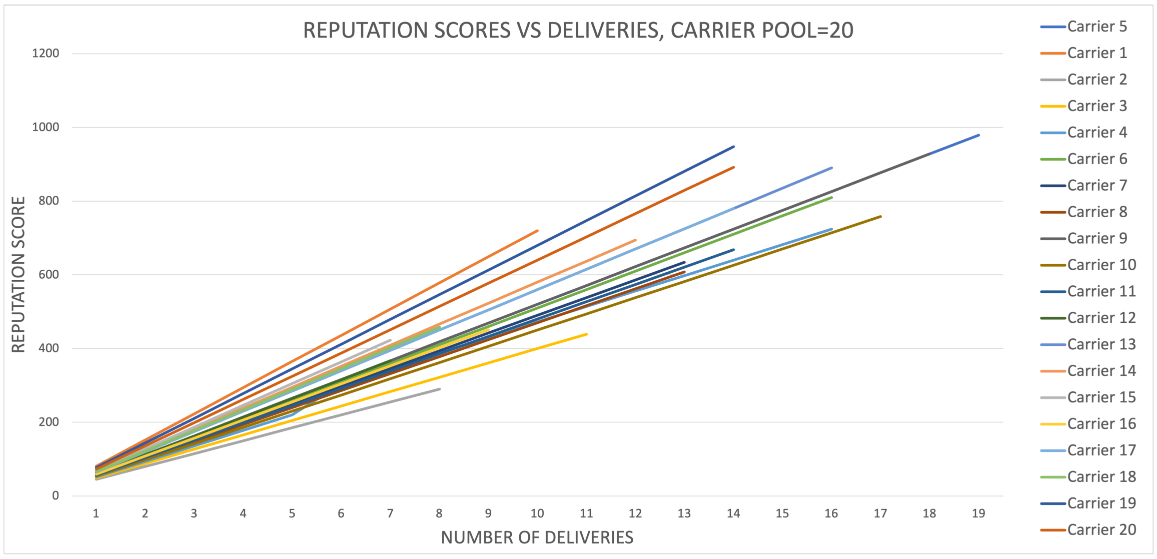
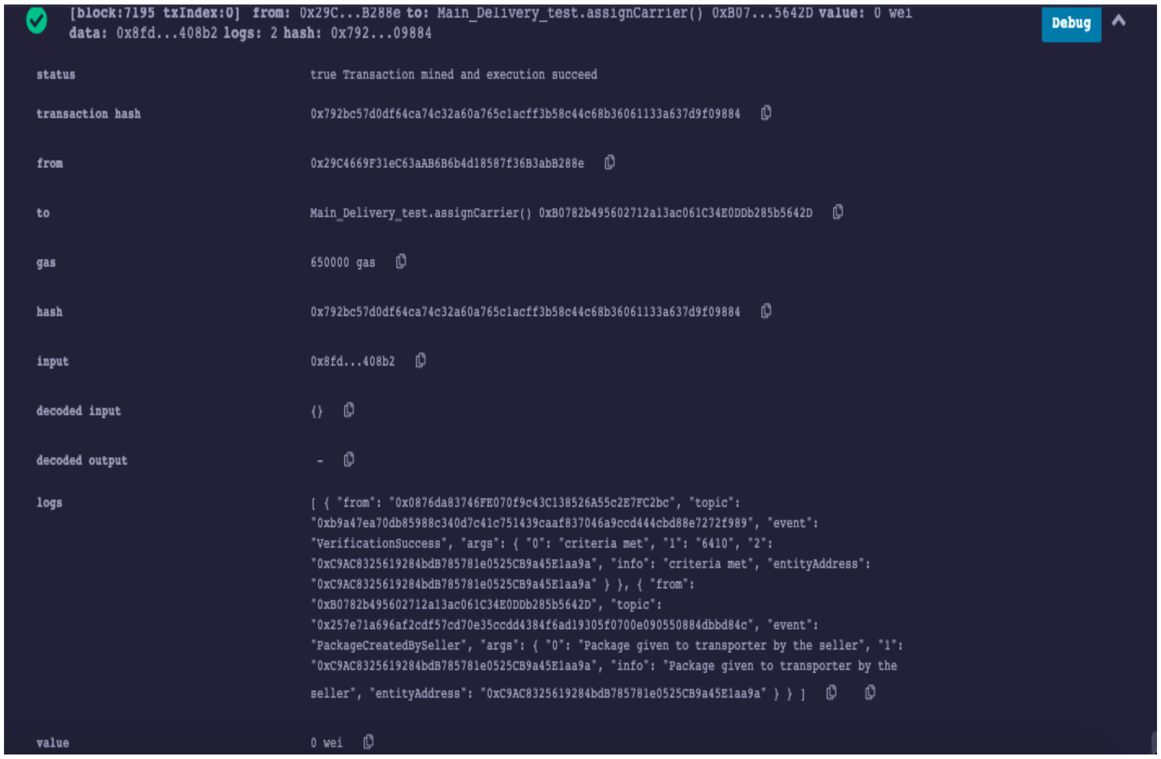


Fig. 4.2.5: Reputation Scores Vs Number Of Deliveries where N (number of carriers) equals 20

restaurants of Bogota City, Colombia are shown in figure 4.2.2, figure 4.2.3, figure 4.2.4 and figure 4.2.5 respectively. Carrier selection plays a vital role in delivery execution. For testing purposes, a random selection process adopted to provide a fair chance to each carrier within the pool. Results indicate that carriers with a low reputation score in the proposed POD system get equal opportunities to deliver an item and grow over time. For instance, in figure 4.2.2 carrier 5, who joined the system with the lowest reputation score of 74, surpassing almost all other carriers in the pool at the end of all 258 delivery orders. All this provide evidence that even low reputation score carriers given the fair chance of opportunities to deliver items in a proposed system maintaining appropriate selection, i.e. those carriers with high reputation scores likely preferred over low reputation score carriers. Also, as the number of carriers in the pool increases, the number of delivery orders per carrier decreases, and most carriers' have similar reputation scores over time, whether they started late or early in the system. Hence, infers larger pool of carriers eliminates gap between highly reputed carriers and less reputed carriers to some extent and motivate newcomers to join the scheme.

4.2.2.2 Proof of Delivery Transaction Logs:

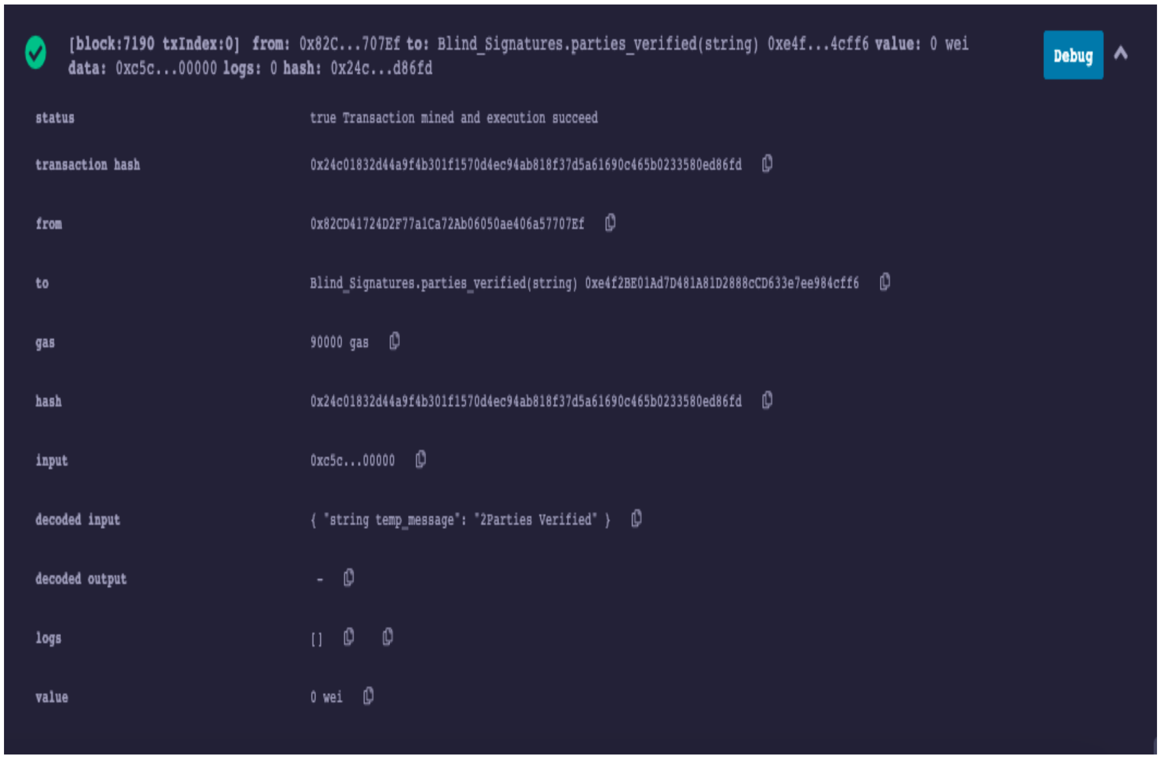
All the delivery transactions are stored in the form of an immutable ledger on the blockchain.Figure 4.2.6 depicts the transaction logs of carrier assignment for a delivery item which returns the address of selected carrier from the carrier pool, figure 4.2.7 shows transaction logs of obtaining a signature verified token indicating the number of parties verified and figure 4.2.8 illustrates payment settlement execution after successful delivery which shows that stored item price in the contract is transferred to seller's and carrier's account and reputation increment happens. All these figures infer how a transaction logs and proof of delivery are stored over blockchain when a successful delivery of an item happens. These logs are also helpful and retrieved easily if in case dispute happens to better understand reasons for point of failure.
Note: These transaction logs are from one of the 257 deliveries executed using the experimental setup mentioned in section 4.2.1.



```
[block:7195 txIndex:0] from: 0x29C...B288e to: Main_Delivery_test.assignCarrier() 0xB07...5642D value: 0 wei
data: 0x8fd...408b2 logs: 2 hash: 0x792...09884

status              true Transaction mined and execution succeed
transaction hash    0x792bc57d0df64ca74c32a60a765c1acff3b58c44c68b36061133a637d9f09884
from                0x29C4669F31eC63aAB6B6b4d18587f36B3abB288e
to                  Main_Delivery_test.assignCarrier() 0xB0782b495602712a13ac061C34E0DDb285b5642D
gas                 650000 gas
hash                0x792bc57d0df64ca74c32a60a765c1acff3b58c44c68b36061133a637d9f09884
input               0x8fd...408b2
decoded input       {}
decoded output      -
logs                [ { "from": "0x0876da83746FE070f9c43C138526A55c2E7FC2bc", "topic":
                    "0xb9a47ea70db85988c340d7c41c751439caaf837046a9ccd444cbd88e7272f989", "event":
                    "VerificationSuccess", "args": { "0": "criteria met", "1": "6410", "2":
                    "0xC9AC8325619284bdB785781e0525CB9a45E1aa9a", "info": "criteria met", "entityAddress":
                    "0xC9AC8325619284bdB785781e0525CB9a45E1aa9a" } }, { "from":
                    "0xB0782b495602712a13ac061C34E0DDb285b5642D", "topic":
                    "0x257e71a696af2cdf57cd70e35ccdd4384f6ad19305f0700e090550884dbbd84c", "event":
                    "PackageCreatedBySeller", "args": { "0": "Package given to transporter by the seller", "1":
                    "0xC9AC8325619284bdB785781e0525CB9a45E1aa9a", "info": "Package given to transporter by the
                    seller", "entityAddress": "0xC9AC8325619284bdB785781e0525CB9a45E1aa9a" } } ]
value               0 wei
```

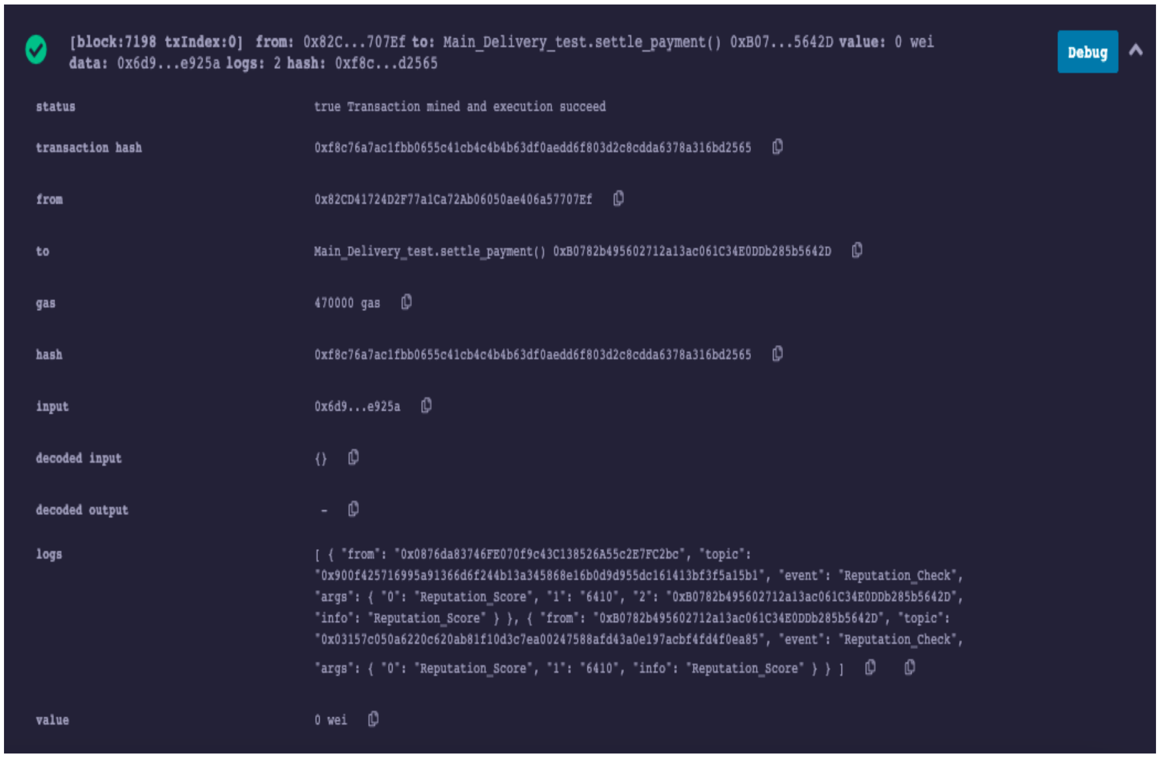
Fig. 4.2.6: POD: Assign Carrier Transaction Logs



```
[block:7190 txIndex:0] from: 0x82C...707Ef to: Blind_Signatures.parties_verified(string) 0xe4f...4cff6 value: 0 wei
data: 0xc5c...00000 logs: 0 hash: 0x24c...d86fd

status              true Transaction mined and execution succeed
transaction hash    0x24c01832d44a9f4b301f1570d4ec94ab818f37d5a61690c465b0233580ed86fd
from                0x82CD41724D2F77a1Ca72Ab06050ae406a57707Ef
to                  Blind_Signatures.parties_verified(string) 0xe4f2BE01Ad7D481A81D2888cCD633e7ee984cff6
gas                 90000 gas
hash                0x24c01832d44a9f4b301f1570d4ec94ab818f37d5a61690c465b0233580ed86fd
input               0xc5c...00000
decoded input       { "string temp_message": "2Parties Verified" }
decoded output      -
logs                []
value               0 wei
```

Fig. 4.2.7: POD: Signature Verified Token Transaction Logs



```
[block:7198 txIndex:0] from: 0x82C...707Ef to: Main_Delivery_test.settle_payment() 0xB07...5642D value: 0 wei
data: 0x6d9...e925a logs: 2 hash: 0xf8c...d2565

status              true Transaction mined and execution succeed
transaction hash    0xf8c76a7ac1fbb0655c41cb4c4b4b63df0aedd6f803d2c8cdda6378a316bd2565
from                0x82CD41724D2F77a1Ca72Ab06050ae406a57707Ef
to                  Main_Delivery_test.settle_payment() 0xB0782b495602712a13ac061C34E0DDb285b5642D
gas                 470000 gas
hash                0xf8c76a7ac1fbb0655c41cb4c4b4b63df0aedd6f803d2c8cdda6378a316bd2565
input               0x6d9...e925a
decoded input       {}
decoded output      -
logs                [ { "from": "0x0876da83746FE070f9c43C138526A55c2E7FC2bc", "topic":
                    "0x900f425716995a91366d6f244b13a345868e16b0d9d955dc161413bf3f5a15b1", "event": "Reputation_Check",
                    "args": { "0": "Reputation_Score", "1": "6410", "2": "0xB0782b495602712a13ac061C34E0DDb285b5642D",
                    "info": "Reputation_Score" } }, { "from": "0xB0782b495602712a13ac061C34E0DDb285b5642D", "topic":
                    "0x03157c050a6220c620ab81f10d3c7ea00247588afd43a0e197acbf4fd4f0ea85", "event": "Reputation_Check",
                    "args": { "0": "Reputation_Score", "1": "6410", "info": "Reputation_Score" } } ]
value               0 wei
```

Fig. 4.2.8: POD: Settle Payment Transaction Logs

4.3 Cost Analysis

Gas utilized in the Ethereum blockchain as a measure of computational work . Various transactions require a different amount of gas, and the transaction fee is calculated in terms of gas, yet the actual cost paid is in Ether. Thus, the gas cost referred to as the amount of gas needed for a transaction and the gas cost is the cost of gas in Ether. Moreover, every transaction's gas limit can be set to avoid running out of gas if there are any bugs in the code. This gas limit gives a security component. It is likewise conceivable to accelerate the transaction depending upon Ether's amount spent per unit of 'Gwei'. Therefore there is a trade-off between priority and cost. Choosing the right gas price is a bit difficult as it constantly requires monitoring of the network. This decision choice has been made simpler by ETH Service station by giving three distinct classifications on their site [15].

1. Safe-Low: This is a gas value that one can decide to go for a modest and safe alternative. It is affordable. Simultaneously, the transaction would be mined

quickly. This price is determined after at least 50 transactions being executed at this price in the last 24 hours.

2. Normal: This gas cost is acknowledged by the top miners and is generally near the default wallet price.

3. Quick: This cost is the most negligible value needed to be picked to be favoured by all the top miners. Choosing a price higher than this cost will most probably not yield to a better speed.

In the cost analysis, a gas price of 58 Gwei was used, which is the latest Average price based on ETH Gas Station [15]. A higher gas price means a higher cost of transaction as well. The new proposed POD system transactional cost in ETH(ether) required for each last-mile operation is reflected in table 4.3.1. To draw a comparison between existing POD systems in literature vs the proposed POD system, the overall cost for single and multi-carrier systems calculated.

Single carrier POD systems overall costs are (refer table 4.3.2) relatively less than the new proposed system because of the less computational extensive steps performed. However, multiple carrier POD system cost indicated that the new proposed POD system has the advantage over these systems because of its customized delivery options and secured verification mechanisms leveraging crowd-sourcing benefits (see table 4.3.3). The overall cost of system comparison in USD indicated in figure 4.3.1. Even if one use cheap gas price of 52 Gwei for transacting various operations there is not much difference in delivery cost. However, the overall transaction cost can be improved by altering the consensus mechanisms for instance: POA which provides flexibility to constraint the amount of gas used per transaction. Note the amount of time taken to confirm every transaction in the proposed POD system is significantly much less in comparison to old POD systems (figure 4.3.2).The minimum mean time to execute blocks in proposed POD system is close to 1700 seconds on the other hand POD Single Carrier and POD Multiple Carrier takes approximately 8000 seconds.

Gas consumption refers to the amount of gas consumed in execution and deployment of delivery smart contracts. Table 4.3.4 compares which part of our proposed

POD system consumes more amount of gas than old POD systems. Single carrier systems consume the least amount of gas in execution and deployment than multi-carrier systems because fewer deployed contracts required for such system. However, our proposed system deployment is relatively expensive than multi-carrier system by Khaled [42] because flexibility is given to join n number of carriers and maintain them in a pool on-chain. In terms of execution, our proposed POD system outperforms and consumes less amount of gas in delivering an item because of less computational extensive steps involved in delivery such as verification of agents via off-chain blind signature API's and division of delivery tasks among various contracts compared to the multi-carrier system which use heavy verification hash on-chain mechanisms to verify agents.

CONTRACT FUNC NAME	GAS ESTIMATED LIMIT	GAS USED	AVG MEAN TIME TO CONFIRM BLOCK(SECOND)	AVERAGE TRANSACTION FEE(ETH)	AVERAGE TRANSACTION FEE (USD)
Blind Signature Deployment	6721975	475435	381	0.0275752	37.11
Rating Contract Deployment	6721975	1092312	542	0.0032769	84.63
Rating Contract- Input Weights by Arbitrator	6721975	86363	381	0.0050091	6.68
Rating Contract- Consensus called by seller	6721975	31015	381	0.0017989	2.40
Rating Contract- Consensus called by buyer	6721975	31903	381	0.0018504	2.47
Rating Contract- Insert Transporter	6721975	5000000	2266	0.29	389.47
Dsipute Contract- Deployment	6721975	681242	381	0.039512	53.09
Main_Delivery- Deployment	6721975	1911033	542	0.1108399	147.94
Main_Delivery- Interest_to_Buy_Item	6721975	46009	381	0.0026685	3.56
Main_Delivery- BuyerDepositItemPrice by Buyer	6721975	76102	381	0.0044139	5.90
Main_Delivery- DeliveryAttributes by Buyer	6721975	49746	381	0.0028853	3.84
Main_Delivery- Assign_Transporter by seller	6721975	171046	381	0.0099207	13.22
Main_Delivery- CreatePackage	6721975	51786	381	0.0030036	4.00
Main_Delivery- DeliveryPackage	6721975	37451	381	0.0021722	2.90
Main_Delivery- PackageReceived	6721975	33213	381	0.0019264	2.57
Main_Delivery- Ask_For_Buyer_sig by seller	6721975	102904	381	0.0059684	7.99
Main_Delivery- Parties Verified off-chain	6721975	90000	381	0.00522	6.99
Main_Delivery- Verify parties verification results	6721975	54605	381	0.0031671	4.24
Main_Delivery- Settle_Payment	6721975	86835	381	0.0050364	6.75
Main_Delivery- Reputation Check	6721975	25113	381	0.0014566	1.96
Total	134439500	10134113	491.35	0.5277015	$787.71

Table 4.3.1: New POD System Cost

CONTRACT FUNC NAME	GAS ESTIMATED LIMIT	GAS USED	AVG MEAN TIME TO CONFIRM BLOCK(SECOND)	AVERAGE TRANSACTION FEE(ETH)	AVERAGE TRANSACTION FEE (USD)
POD Deployment	5000000	6669004	11294	0.3868022	520.54852337902
POD Signed_Term_And_Conditions: SELLER	5000000	78732	7148	0.0045665	6.145479089856
POD_Signed_Term_And_Conditions:CARRIER	5000000	53912	7148	0.0031269	4.2081021714816
POD_Signed_Term_And_Conditions:BUYER	5000000	184680	7148	0.0107114	14.41512859369
POD_Create_Package_And_Key	5000000	82360	7148	0.0047769	6.4286300370816
POD_deliverPackage()	5000000	79916	7148	0.0046351	6.2377992180864
POD_requestPackageKey() BY BUYER	5000000	86554	7148	0.0050201	6.7559223867264
POD_verifyTransporter()	5000000	136582	7148	0.0079218	10.660956148915
POD_verifyBuyer()	5000000	170734	7148	0.0099026	13.326666207206
Total	45000000	7542474	7608.6666666667	0.4374635	$588.73

Table 4.3.2: Single Carrier POD System Cost

CONTRACT FUNC NAME	GAS ESTIMATED LIMIT	GAS USED	AVG MEAN TIME TO CONFIRM BLOCK(SECOND)	AVERAGE TRANSACTION FEE(ETH)	AVERAGE TRANSACTION FEE (USD)
POD_Contract Deployment	5000000	2819060	11294	0.1635055	220.0415266236
POD Signed_Term_And_Conditions: SELLER	5000000	78732	7148	0.0045665	6.1454790899
POD_Signed_Term_And_Conditions:CARRIER	5000000	53912	7148	0.0031269	4.2081021715
POD_Signed_Term_And_Conditions:BUYER	5000000	184680	7148	0.0107114	14.4151285937
POD_Create_Package_And_Key	5000000	80708	7148	0.0046811	6.2997048434
Courier_Contract_Deployment	5000000	2988034	11294	0.173306	233.2307892580
Courier_Contract_Signed_Term_And_Conditions:CARRIER 2	5000000	78554	7148	0.0045561	6.1314830354
Courier_Contract_Carrier_2_Confirm_Package_Received	5000000	81854	7148	0.0047475	6.3890642678
Courier_verifyTransporter() hash entered by carrier 1	5000000	136712	7148	0.0079293	10.6710494574
Courier_verifyTransporter() hash entered by carrier 2(verification done internally)	5000000	112550	7148	0.0065279	8.7850811235
BT_contract_Deployment	5000000	2637202	11294	0.1529577	205.8465667322
BT_contract.ConfirmPackageReceived()	5000000	111898	7148	0.0064901	8.7342108488
BT_contract.KeysEnteredByTransporter()	5000000	136712	7148	0.0079293	10.6710494574
BT_contract.verifyKeysByBuyer()	5000000	112638	7148	0.006533	8.7919445733
POD.settle_payement()	5000000	63320	7148	0.0036726	4.9424912965
Courier.settle_payement()	5000000	63276	7148	0.00367	4.9389922829
BT_contract.settle_payement()	5000000	72172	7148	0.004186	5.6334119063
Total	85000000	9812014	7879.6470588235	0.5690969	$ 765.88

Table 4.3.3: Multiple Carrier POD System Cost

PROCESS CONSUMING GAS	NEW POD SYSTEM	GAS CONSUMPTION	SINGLE CARRIER	GAS CONSUMPTION	MULTIPLE CARRIERS	GAS USED
DEPLOYMENT	BLIND SIGNATURE CONTRACT	475435	POD CONTRACT	6669004	POD CONTRACT	2819060
	DISPUTE CONTRACT	681242			COURIER CONTRACT	2988034
	RATING CONTRACT	1092312			BT CONTRACT	2637202
	MAIN DELIVERY CONTRACT	1911033				
	INSERT NEW CARRIER IN A POOL	5000000				
TOTAL DEPLOYMENT GAS CONSUMPTION		9160022		6669004		8444296
EXTENSIVE DELIVERY EXECUTION STEPS			SIGNING TERMS & CONDITIONS BY BUYER	184680	SIGNING TERMS & CONDITIONS BY BUYER	184680
	ASSIGNMENT OF CARRIER	171046	VERIFICATION OF AGENTS	170734	REST OF DELIVERY EXECUTION STEPS	1183038
	REST OF DELIVERY EXECUTION STEPS	803045	REST OF DELIVERY EXECUTION STEPS	518056		
TOTAL DELIVERY EXECUTION GAS CONSUMPTION		974091		873470		1367718

Table 4.3.4: Gas Cost Comparison

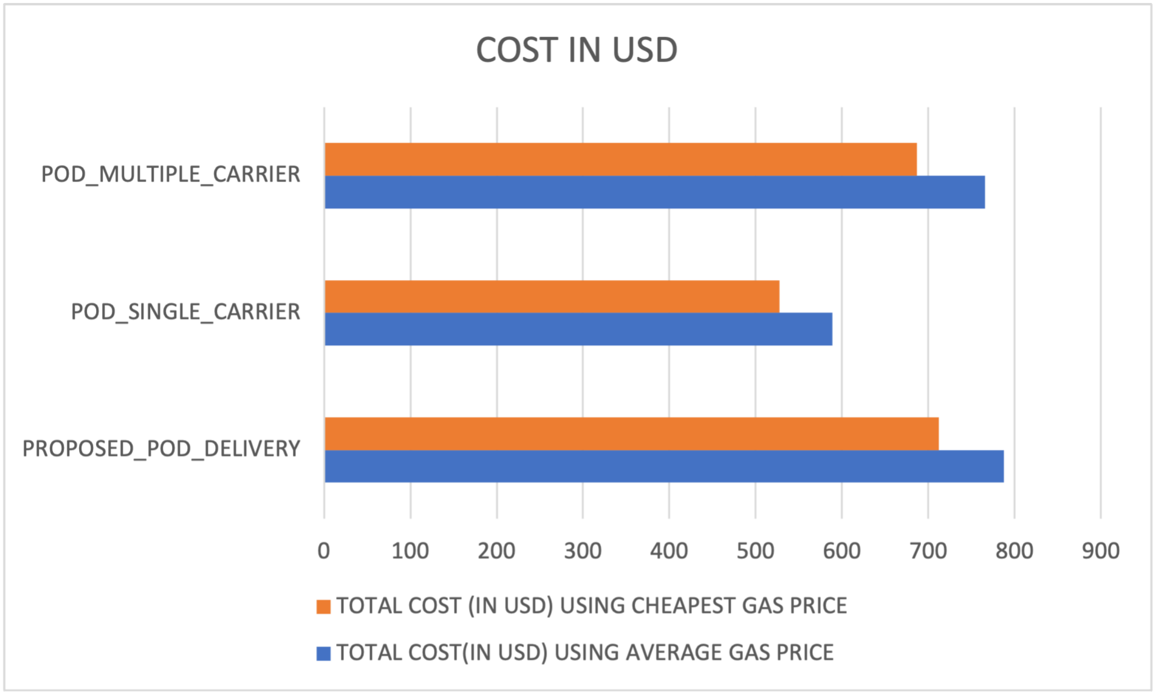


Fig. 4.3.1: Cost comparison between various POD systems

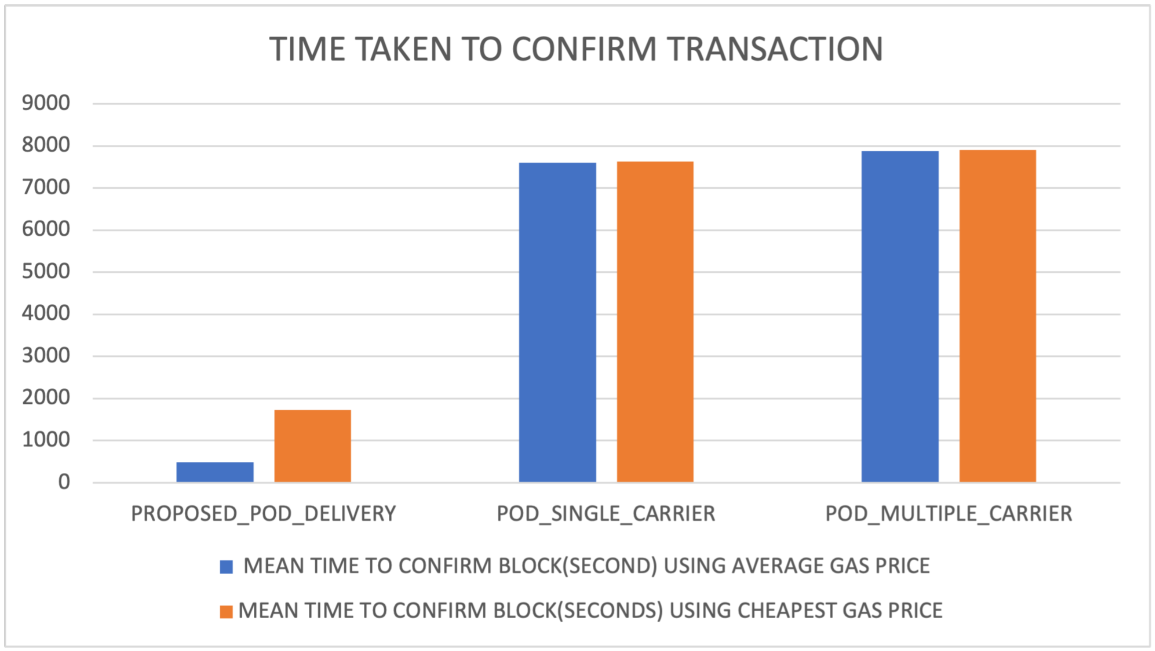


Fig. 4.3.2: Time comparison of between various POD systems

4.4 Tools Used for Implementation

- **Remix IDE**: Solidity-based IDE(Integrated Development Environment) that is used to write, compile and debug solidity code.For more features of remix refer [26].

- **Ganache-cli**:a fast and customizable blockchain emulator. It allows you to make calls to the blockchain without the overheads of running an actual Ethereum node [18].

- **Truffle**: A world class development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM), aiming to make life as a developer easier [53].

- **RSA Blind Signatures packages**:These already implemented version of RSA based blind signatures packages(for signature generation and verification) can be downloaded from here [29].

- **web3.js-Ethereum JavaScript API** :is a collection of libraries that allow you to interact with a local or remote ethereum node using HTTP, IPC or WebSocket [2].

CHAPTER 5

Conclusion and Future Work

5.1 Conclusion

The work presented in this thesis provided a crowd-sourced based proof of a delivery system that facilitates trading and tracking sold items in a decentralized way. The solution provides proof of physical objects' delivery in a decentralized manner taking advantage of crowd-sourcing, blind signatures and immutability that blockchain offers. Starting with Chapter 1, a brief overview of last-mile challenges and introduction of problems related to proof of delivery in last mile operations.

In Chapter 2, a literature review concerning blockchain use for obtaining proof of delivery(POD), blockchain use for crowd-sourcing and last-mile delivery in general. The primary focus is on literature that contributes to using blockchain to deliver physical assets and crowd-sourcing via reputation-based systems to build trust and provide equal opportunity to participate in the system.

In Chapter 3, a proposed POD system via blockchain and crowd-sourcing was described thoroughly and then implemented. A carrier selection policy introduced indicating how carriers selected under various possible conditions. To verify product reached to its correct customer and obtain proof of delivery- signature verification scheme based on blind signatures used and implementation design of the whole system elucidated. The proposed solution is generic enough and can be applied to almost all shipped items and assets. It has a trust mechanism that builds trust in the system and provides equal opportunities for new agents. It also eliminates the need for a trusted third party and utilizes smart contracts as escrow to settle payments even under disputes. In the end, a brief discussed how proposed system handles commonly occurred disputes. Moreover, I analyzed my solution's security and concluded that the solution is resilient to known security attacks and satisfies cyber-security features and objectives.

In chapter 4, I tested out key functionalities and demonstrated the correct behaviour and outcomes considering execution of multiple delivery simultaneously. The cost analysis reveals that the overall system resembles the cost of existing POD systems. The cost of each transaction in a smart-contract is proportional to the current value of the gas price used. However, the overall system cost will be reduced by altering the choice of consensus mechanism, which can be considered as part of future work.

Blockchain provides immutability in terms of non-tampering of information blocks stored over the chain. Immutability brings trust and integrity to the data used by various business stakeholders everyday.However, to achieve certain level of immutability in current proposed POD schema consumes a substantial amount of computational resources such as Ether gas, making it expensive to use. However, this cost can be significantly reduced with non-crypto-currency consensus protocols.

5.2 Future Work

In this thesis, I presented a step in the right direction but there is still much work to be done, in both obtaining secure proof of delivery and in making them practical for large scale delivery applications. As a part of future work, one can explore how to assign reputations to the delivery items, set multiple packages to the single carrier to make it cost-effective, or maintain the balance between the pool of items vs a collection of carriers? Is it useful to add the vehicle's description as the delivery attribute, the effect of different consensus algorithms on the delivery system's overall cost, and what type of information is necessary to put on-chain.

REFERENCES

[1] AlTawy, R., ElSheikh, M., Youssef, A. M., and Gong, G. (2017). Lelantos: A blockchain-based anonymous physical delivery system. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 15–1509.

[2] API, E. J. (2018). web3 - Ethereum JavaScript API. `https://web3js.readthedocs.io/en/v1.3.4/`. [Online accessed 07-March-2021].

[3] Awwad, M., Reddy, S., Kazhana Airpulli, V., Zambre, M. S., Marathe, A., and Jain, P. (2018). Blockchain technology for efficient management of supply chain.

[4] Barclays (2014). The Last Mile: Exploring the online purchasing and delivery journey. `https://docplayer.net/1342335-The-last-mile-exploring-the-online-purchasing-and-delivery-journey.html`. [Online accessed 23-Feb-2021].

[5] BinanceAcademy (2020). Proof of Burn (PoB). `https://academy.binance.com/en/articles/proof-of-burn-explained`. [Online accessed 23-Feb-2021].

[6] Boyer, K., Prud'homme, A., and Chung, W. (2009). The last mile challenge: Evaluating the effects of customer density and delivery window patterns. *Journal of Business Logistics*, 30:185 – 201.

[7] Brian, O. (2018). How crowdsourcing is transforming the face of last mile delivery.

[8] Castillo, V., Bell, J., Rose, W., and Rodrigues, A. (2018). Crowdsourcing last mile delivery: Strategic implications and future research directions. *Journal of Business Logistics*, 39.

[9] Chen, Y., Jing, Y., and Wei, J. (2018). Consumer's intention to use self-service parcel delivery service in online retailing: An empirical study. *Internet Research*, 28:00–00.

[10] Correa, J. C. (2018). Raw data of a web mining approach to collaborative consumption of food delivery services. `https://data.mendeley.com/datasets/m9z9hw4nsc/1`. [Online accessed 23-Feb-2021].

[11] Deloitte (2018). The last-mile challenge in Canada. `https://www2.deloitte.com/content/dam/Deloitte/ca/Documents/consumer-industrial-products/ca-final-mile-challengesIn-canada-report-2-aoda-en.pdf`. [Online accessed 23-Feb-2021].

[12] Demir, M., Turetken, O., and Ferwom, A. (2019). Blockchain and iot for delivery assurance on supply chain(bidas). pages 5213–5222.

[13] Ethereum (2018). Ethereum Developer Resources. `https://ethereum.org/en/developers/`. [Online accessed 23-Feb-2021].

[14] Ethereum (2020). Proof of Stake (PoS). `https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/`. [Online accessed 23-Feb-2021].

[15] EtherGasStation (2018). `https://ethgasstation.info/`. [Online accessed 23-Feb-2021].

[16] Foods, U. (2019). NEW STUDY SHOWS WHAT CONSUMERS CRAVE IN A FOOD DELIVERY SERVICE. `https://www.usfoods.com/our-services/business-trends/2019-food-delivery-statistics.html`. [Online accessed 23-Feb-2021].

[17] FoxNews (2019). Study: Estimated 36% of Americans have fallen victim to 'porch pirates'. `https://www.fox6now.com/news/study-estimated-36-of-americans-have-fallen-victim-to-porch-pirates`. [Online accessed 23-Feb-2021].

[18] Ganache (2018). Ganache CLI. https://docs.nethereum.com/en/latest/ethereum-and-clients/ganache-cli/. [Online accessed 07-March-2021].

[19] Gdowska, K., Viana, A., and Pedroso, J. P. (2018). Stochastic last-mile delivery with crowdshipping. *Transportation Research Procedia*, 30:90–100.

[20] geeksforgeeks (2019a). practical Byzantine Fault Tolerance(pBFT) . https://www.geeksforgeeks.org/practical-byzantine-fault-tolerancepbft/. [Online accessed 23-Feb-2021].

[21] geeksforgeeks (2019b). Producer Consumer Problem using Semaphores. https://www.geeksforgeeks.org/producer-consumer-problem-using-semaphores-set-1/. [Online accessed 23-Feb-2021].

[22] geeksforgeeks (2019c). Proof of Work (PoW) Consensus. https://www.geeksforgeeks.org/proof-of-work-pow-consensus/. [Online accessed 23-Feb-2021].

[23] Haag, M. and Hu, W. (2019). 1.5 Million Packages a Day: The Internet Brings Chaos to N.Y. Streets. https://www.nytimes.com/2019/10/27/nyregion/nyc-amazon-delivery.html. [Online accessed 23-Feb-2021].

[24] Hasan, H. and Salah, K. (2018). Blockchain-based proof of delivery of physical assets with single and multiple transporters. *IEEE Access*, PP:1–1.

[25] Hoffman, K., Zage, D., and Nita-Rotaru, C. (2009). A survey of attack and defense techniques for reputation systems. *ACM Comput. Surv.*, 42.

[26] IDE, R. (2018). Remix IDE. https://remix-ide.readthedocs.io/en/latest/. [Online accessed 07-March-2021].

[27] Jøsang, A. and Golbeck, J. (2009). Challenges for robust trust and reputation systems. *In: Proceedings of the 5th Int. Workshop on Security and Trust Management (STM2009).*

[28] Kavouras, I. (2018). How to Calculate Roulette Probabilities. `https://www.888casino.com/blog/calculating-probability-roulette`. [Online accessed 23-Feb-2021].

[29] KevineJohn (2018). RSA based Blind Signatures Implementation. `https://github.com/kevinejohn/blind-signatures`. [Online accessed 07-March-2021].

[30] Kirill, G. (2019). Overview of 9 blockchain consensus algorithms. `https://digiforest.io/en/blog/blockchain-consensus-algorithms`. [Online accessed 23-Feb-2021].

[31] Kretzschmar, J. and Eckardt, F. (2019). A blockchain approach towards cargo sharing in last mile logistics. *NexComm 2021*, pages 11–15.

[32] Liang, S., Li, M., and Li, W. (2019). Research on traceability algorithm of logistics service transaction based on blockchain. pages 186–189.

[33] Liu, Y., He, D., Obaidat, M., Kumar, N., Khan, K., and Choo, K.-K. R. (2020). Blockchain-based identity management systems: A review. *Journal of Network and Computer Applications*, 166:102731.

[34] Mak, H. Y. (2018). Peer-to-peer crowdshipping as an omnichannel retail strategy.

[35] Malik, S., Dedeoglu, V., Kanhere, S., and Jurdak, R. (2019). Trustchain: Trust management in blockchain and iot supported supply chains.

[36] Marti, S. and Garcia-Molina, H. (2006). Taxonomy of trust: Categorizing p2p reputation systems. *Computer Networks*, 50:472–484.

[37] NewYorkTimes (2018). 90,000 Packages Disappear Daily in N.Y.C. Is Help on the Way? `https://www.nytimes.com/2019/12/02/nyregion/online-shopping-package-theft.html?mod=djem10point`. [Online accessed 23-Feb-2021].

[38] Nguyen, D., De Leeuw, S., Dullaert, W., and Foubert, B. (2019). What is the right delivery option for you? consumer preferences for delivery attributes in online retailing. *Journal of Business Logistics*.

[39] Ni, M., He, Q., Liu, X., and Hampapur, A. (2019). Same-day delivery with crowdshipping and store fulfillment in daily operations. *Transportation Research Procedia*, 38:894–913.

[40] Quadient (2019). How to Ensure Secure Package Delivery in Your Residential Community. `https://www.parcelpending.com/blog/secure-package-delivery/`. [Online accessed 23-Feb-2021].

[41] Quora (2020). How many houses does the average postman deliver to in a day? `https://www.quora.com/How-many-houses-does-the-average-postman-deliver-to-in-a-day`. [Online accessed 23-Feb-2021].

[42] Salah, K. and Hasan, H. (2018). Blockchain-based solution for proof of delivery of physical assets.

[43] Segura, M. A. and Correa, J. C. (2019). Data of collaborative consumption in online food delivery services. *Data in Brief*, 25:104007.

[44] Serafini, S., Nigro, M., and Gatta, V. (2018). Sustainable crowdshipping using public transport: a case study evaluation in rome. *Transportation Research Procedia*, 30:101–110.

[45] ShipChain (2020). How ShipChain Works? `https://shipchain.io/how-it-works`. [Online accessed 23-Feb-2021].

[46] Shivaansh, K. (2019). Proof of Capacity(PoC). `https://medium.com/@shivaanshkapoor02/what-is-poc-proof-of-capacity-c85febb5d18e`. [Online accessed].

[47] Signatures, B. (2018). Blind Signatures. `https://www.sciencedirect.com/topics/computer-science/blind-signature`. [Online accessed 07-March-2021].

[48] Song, J. M., Sung, J., and Park, T. (2019). Applications of blockchain to improve supply chain traceability. *Procedia Computer Science*, 162:119–122.

[49] Song, L., Cherrett, T., Mcleod, F., and Wei, G. (2009). Addressing the last mile problem. *Transportation Research Record*, 2097:9–18.

[50] Taniguchi, E. and Tamagawa, D. (2005). Evaluating city logistics measures considering the behavior of several stakeholders. *Journal of the Eastern Asia Society for Transportation Studies*, 6.

[51] TransUnion (2020). Protecting the Last Mile of Retail E-Commerce: The Prevention of Shipping Fraud. `https://www.transunion.com/blog/prevention-of-shipping-fraud`. [Online accessed 23-Feb-2021].

[52] Treasureishere (2020). Last Mile Delivery Market to 2027 — Global Analysis and Forecasts by Technology. `https://medium.com/@treasureishere/last-mile-delivery-market-to-2027-global-analysis-and-forecasts-by-technology-ae2c04b6c9b4`. [Online; accessed 14-Feb-2021].

[53] Truffle (2018). Truffle test-framework. `https://www.trufflesuite.com/docs/truffle/overview`. [Online accessed 07-March-2021].

[54] Tuler de Oliveira, M., Reis, L., Medeiros, D., Carrano, R., Olabarriaga, S., and Menezes, D. (2020). Blockchain reputation-based consensus: A scalable and resilient mechanism for distributed mistrusting applications. *Computer Networks*, 179:107367.

[55] Uber (2021). Uber delivery-ratings-explained. `https://www.uber.com/us/en/deliver/basics/tips-for-success/delivery-ratings-explained/`. [Online accessed 23-Feb-2021].

[56] Wang, X., Zhan, L., Ruan, J., and Zhang, J. (2014). How to choose "last mile" delivery modes for e-fulfillment. *Mathematical Problems in Engineering*, 2014.

[57] Wang, Y., Zhang, D., Liu, Q., Shen, F., and Lee, L. H. (2016). Towards enhancing the last-mile delivery: An effective crowd-tasking model with scalable solutions. *Transportation Research Part E: Logistics and Transportation Review*, 93:279–293.

[58] Whitby, A. and Jøsang, A. (2004). Filtering out unfair ratings in bayesian reputation systems. *The Icfain Journal of Management Research*, 4.

[59] Zhao, Z. and Liu, Y. (2019). A blockchain based identity management system considering reputation. In *2019 2nd International Conference on Information Systems and Computer Aided Education (ICISCAE)*, pages 32–36.

Appendix A: Smart Contracts

```
//MAIN DELIVERY CONTRACT
pragma solidity ^0.4.26;
import './Rating_Contract.sol';
import './Blind_Signatures.sol';
import './Dispute.sol';

contract Main_Delivery {
    Rating_Contract rt;
    Blind_Signatures bs;
    Dispute dt;
    address public seller;
    address public  buyer;
    address public transporter;
    address  public arbitrator; // Trusted incase of dispute
    address public attestaionAuthority; // Party that attested the
        smart contract

    uint public itemPrice;
    uint public rep_score;
    string public Verified_parties;
    uint type_of_dispute;
    string itemID;


    string public TermsIPFS_Hash; // Terms and conditions agreement
        IPFS Hash

    // Enum wont allow the contract to be in any other state
    enum contractState {IntialState, waitingForItemPriceByBuyer,
       MoneyWithdrawn, PackageCreated
                         ,waitingForAssignmentofTransporter,
                            TransporterNotAssigned,
                         TransporterAssigned,
                            PackageAndTransporterAssigned,
                            ItemOnTheWay,ArrivedToDestination,
                         PaymentSettledSuccess, Aborted, Refund,
                            Dispute,CancellationRefund }

    contractState public state;

    // @notice Will receive any eth sent to the contract
    function () external payable {

    }


    mapping(address => bool) public cancellable;

    uint deliveryDuration;
    uint startEntryTransporterKeysBlocktime;
    uint buyerVerificationTimeWindow;
    uint startdeliveryBlocktime;

    constructor(address  _seller,
    address  _buyer,
    address  _arbitrator,
    address  _attestationAuthority,
    address _rt_addr,
    address _bs_addr,
    address _dt_Addr,
    string memory _itemID)
    public {
        seller = _seller;
        buyer = _buyer;
        arbitrator = _arbitrator;
        attestaionAuthority = _attestationAuthority;


        itemPrice = 1 ether;
        itemID = _itemID;
        deliveryDuration = 2 hours; // 2 hours

        TermsIPFS_Hash = "
            QmWWQSuPMS6aXCbZKpEjPHPUZN2NjB3YrhJTHsV4X3vb2td";

        state = contractState.IntialState;
        rt=Rating_Contract(_rt_addr);
        bs= Blind_Signatures(_bs_addr);
        dt= Dispute(_dt_Addr);

    }

    modifier costs() {
       require(msg.value == itemPrice);
       _;
    }
    modifier OnlySeller() {
        require(msg.sender == seller);
        _;
    }

    modifier OnlyBuyer() {
        require(msg.sender == buyer);
        _;
    }

    modifier OnlyTransporter() {
        require(msg.sender == transporter);
        _;
    }
     modifier OnlyArbitrator() {
        require(msg.sender == arbitrator);
        _;
    }

    modifier OnlySeller_Buyer_Arbitrator() {
        require(msg.sender == seller || msg.sender == buyer || msg.sender == arbitrator);
        _;
    }

    event TermsAndConditionsSignedBy(string,address);
    event RatingVerified(string info, address entityAddress);
    event RatingNotVerified(string info, address entityAddress);
    event PackageCreatedBySeller(string info, address entityAddress);
    event DeliveryDeclined(string info, address entityAddress);
    event Reputation_Check(string info, uint256);
    event ItemPriceWithdrawnSuccessfully(string,address);
    event ItemNotPaid(string, address);
    event InterestedInBuyItem(string,address);
    event PackageIsOnTheWay(string,address);
    event ArrivedToDestination(string,address);
    event BothPartiesVerified(string,address);
    event DeliveryTimeExceeded(string info,address);
    event CancellationRequest(address,string, string);
    event DeliveryAttributes(string,bool,string);
    event NoDeliveryAttributes(string,bool,string);
    event DisputeResolvedOffchain(string info,address);


    function InterestToBuyItem() public OnlyBuyer {
        require(state == contractState.IntialState);
        emit InterestedInBuyItem('Buyer wants to Buy Item', msg.sender);
        state=contractState.waitingForItemPriceByBuyer;


    }

    function PayItemPrice() public payable  costs OnlyBuyer {
        if(msg.sender == buyer) {
            require(state == contractState.
               waitingForItemPriceByBuyer);
            emit TermsAndConditionsSignedBy("Terms and Conditiond
               verified : ", msg.sender);
            emit ItemPriceWithdrawnSuccessfully("Item Price is
               withdrawn successfully from: ", msg.sender);
            state = contractState.MoneyWithdrawn;
             cancellable[seller]=true;
             cancellable[buyer] = true;

        }
        else{
            emit ItemNotPaid("Item Price Not Paid: ", msg.sender);
            state = contractState.Aborted;
        }
    }

    function delivery_attributes(bool custm_Delivery, string
       location ) public OnlyBuyer{

        if(custm_Delivery){
            emit DeliveryAttributes('Custom Delivery Requirements
               Entered',custm_Delivery,location);
        }
        else{
            emit NoDeliveryAttributes('No Custom Delivery
               Requirements Entered',custm_Delivery,location);
        }
    }

    function createPackage() public OnlySeller {
        require(state == contractState.MoneyWithdrawn);

        cancellable[msg.sender] = false;
        cancellable[transporter]=false;

        state = contractState.PackageCreated;
        emit PackageCreatedBySeller("Package created and given to
            transporter by the sender ", msg.sender);
    }

    function assignCarrier() public OnlySeller  returns(address){
        require(state == contractState.PackageCreated);
        transporter=rt.assign_transporter();
        cancellable[msg.sender] = false;
        cancellable[transporter]=false;
        emit PackageCreatedBySeller("Package given to transporter by
             the seller", transporter);
        state = contractState.PackageAndTransporterAssigned;

        return transporter;
    }

    function PackageOutForDelivery() public OnlyTransporter {
        require(state == contractState.PackageAndTransporterAssigned
            );
        startdeliveryBlocktime = block.timestamp;//save the delivery
             time
        cancellable[buyer] = false;
        emit PackageIsOnTheWay("The package is out for delivery  ",
            msg.sender);
        state = contractState.ItemOnTheWay;
    }

     function ReachedDestination() public OnlyTransporter {
        require(state == contractState.ItemOnTheWay);
        emit ArrivedToDestination("Transporter Arrived To
            Destination " , msg.sender);
        state = contractState.ArrivedToDestination;

    }



    function AskForBuyerMessage(string memory message) public
        OnlySeller returns(string memory){
        require(state== contractState.ArrivedToDestination);
        bs.verify(message);
        Verified_parties=bs.parties_verified(message);
        return Verified_parties;
    }
    function VerifiedParties() public OnlySeller {

        if(keccak256(abi.encodePacked(bs.x()))==keccak256(abi.
            encodePacked('2Parties Verified'))){

            emit BothPartiesVerified('Parties Verified via Blind
                Signatures',msg.sender);

            }

            else {
                type_of_dispute=2;
                state=contractState.Dispute;
            }
    }

    function settle_payment() public OnlyArbitrator returns(uint){
        seller.transfer(((90*itemPrice)/100));
        transporter.transfer(((10*itemPrice)/100));
        state=contractState.PaymentSettledSuccess;
        rep_score=reputation_check();
        rep_score=rep_score+1;
        return rep_score;

    }
    function exceedDeliveryTime() public OnlyBuyer{
         //refund incase delivery take more delivery time
         require(block.timestamp > startdeliveryBlocktime+
            deliveryDuration &&
         (state==contractState.ItemOnTheWay));
         emit DeliveryTimeExceeded('Item not delivered on time,
            Refund Request by',buyer);
         state=contractState.Dispute;
         type_of_dispute=3;
         disputeHandling();

     }

     function disputeHandling() public OnlySeller_Buyer_Arbitrator{
         require(state==contractState.Dispute);

         if(type_of_dispute==1){
             dt.carrierSelection();

         }
         else if (type_of_dispute==2)
         {
             dt.NoSignatureMatchFound();
         }
         else if (type_of_dispute==3){

             dt.CarrierTimeExceeded();
        }
        else{
            emit DisputeResolvedOffchain('Dsipute can be resolved
               offchain by', arbitrator);
        }
     }
    function cancelDelivery(string memory reason) public
       OnlySeller_Buyer_Arbitrator{
        require(cancellable[msg.sender]==true);
        state=contractState.CancellationRefund;
        buyer.transfer(itemPrice);
        emit CancellationRequest(msg.sender,'has requested a
           cancellation due to',reason);
        state = contractState.Aborted;
    }


     function reputation_check() public returns(uint) {
         rep_score=rt.reputation_check();
        emit Reputation_Check('Reputation_Score',rep_score);
        return rep_score;
    }
     function updateReputation() public {
        require(state==contractState.PaymentSettledSuccess);
        rt.setTarget(rep_score,transporter);
    }
}
```

```
//RATING CONTRACT
pragma solidity ^0.4.26;



contract Rating_Contract {

    address public seller;
    address  public buyer;
    address  public transporter;
    address  public arbitrator; // Trusted incase of dispute
    address public attestaionAuthority; // Party that attested the
       smart contract
    uint public itemPrice;
    uint public overall_reputation=0;
    uint public trustscore;
    uint public feature_weight_1;
    uint public feature_weight_2;
    uint public rep_score=0;
    address public md_addr;
    string itemID;

    enum contractState { waitingForWeightsbyArbitrator,
                         weights_set_by_arbitrator,
                            weights_accepted_by_Seller,
                            weights_accepted_by_Buyer,
                         agreed_feature_weights }

    contractState public state;

     modifier OnlySeller() {
        require(msg.sender == seller);
        _;
    }
     modifier OnlyBuyer() {
        require(msg.sender == buyer);
        _;
    }
     modifier OnlyArbitrator() {
        require(msg.sender == buyer);
        _;
    }

    modifier OnlySeller_Buyer_Arbitrator() {
        require(msg.sender == seller || msg.sender == buyer || msg.
            sender == arbitrator);
        _;
    }




    event InputWeightsByArbitrator(string info, address
       entityAddress);
    event TermsAndConditionsSignedBy(string info, address
       entityAddress);
    event WeightsAssignedSuccessfully(string info, uint,uint,address
        entityAddress);
    event Overall_Reputation(string info,uint, address entityAddress
       );
    event VerificationFailure(string info, uint,address
       entityAddress);
    event VerificationSuccess(string info,uint, address
       entityAddress);
    event PrintTransporterReputation(string info,uint );
    event Dispute(string info,address entityAddress);
    event PrintCountAddress(string info,uint,address entityAddress);
    event Reputation_Check(string info,uint, address);

    constructor(
        address _seller,
        address _buyer,
    address  _arbitrator,
    address  _attestationAuthority
    )
    public {

        seller=_seller;
        buyer=_buyer;
        arbitrator = _arbitrator;
        attestaionAuthority = _attestationAuthority;
       state = contractState.waitingForWeightsbyArbitrator;

    }


    struct Agent_Struct {
    string name;
    string ID;
    uint  reputation;

    uint buyer_score_for_carrier;
     uint seller_score_for_carrier;
     bool carrier_already_allocated;
    }


    mapping(address => Agent_Struct) public sellerStruct;
    mapping(address => Agent_Struct) public buyerStruct;
    mapping(address => Agent_Struct) public transporterStruct;

    address[] public sellersAddress;
    address[] public buyerAddress;
    address[] public transporterAddress;


    // input ratings

    function getSellerReputation() external view returns(address[]
       memory){
        return sellersAddress;
    }
     function getBuyerReputation() external view returns(address[]
        memory){
        return buyerAddress;
    }


    function InsertTransporter(string memory name,string memory ID,
       uint _reputation,
                                uint _buyer_score_for_carrier,uint
```

```
                                    _seller_score_for_carrier)
    public{
       transporterStruct[msg.sender].name = name;
        //set user name
      transporterStruct[msg.sender].ID = ID;
      transporterStruct[msg.sender].reputation = _reputation;

      transporterStruct[msg.sender].buyer_score_for_carrier=
         _buyer_score_for_carrier;
      transporterStruct[msg.sender].seller_score_for_carrier=
         _seller_score_for_carrier;
        transporterStruct[msg.sender].carrier_already_allocated=
           false;
        //address'i degistirdim
       transporterAddress.push(msg.sender);
    }
    function getTransporterReputation() public view returns(address
       [] memory){
        return transporterAddress;
    }

    function getTransporterReputationCount() public view returns(
       uint ){
        return transporterAddress.length;
    }
    function getBuyerReputationCount() public view returns(uint ){
        return buyerAddress.length;
    }
     function getSellerReputationCount() public view returns(uint ){
        return sellersAddress.length;
    }


    // weights_assigned

    function input_weights_by_Arbitrator(uint w1,uint w2) public
    {       if(msg.sender==arbitrator){
            require(state ==contractState.
               waitingForWeightsbyArbitrator);
            feature_weight_1=w1;
            feature_weight_2=w2;
            emit InputWeightsByArbitrator("Weights Inputed by
               arbitrator  : ", msg.sender);

            state = contractState.weights_set_by_arbitrator;
            }
    }

    //weights agreed by each Party

    function consesus_on_weights(bool weight_flag) public {
        if(msg.sender==seller)
        {
            require(state==contractState.weights_set_by_arbitrator);
            if(weight_flag)
            {
            emit TermsAndConditionsSignedBy('weights accepted by
               seller',msg.sender);
            state=contractState.weights_accepted_by_Seller;
            }
            else
            emit Dispute('Dispute occurs: weights are not accepted
               by each party',msg.sender);
        }
        else if(msg.sender==buyer)
        {
            require(state==contractState.weights_accepted_by_Seller)
               ;
            if(weight_flag)
            {
            emit TermsAndConditionsSignedBy('weights accepted by
               buyer',msg.sender);
            state=contractState.weights_accepted_by_Buyer;

            }
         else
            emit Dispute('Dispute occurs: weights are not accepted
               by each party',msg.sender);

        }
    }
    function setTarget(uint _rep_score,address _transporter) public
       returns(uint){
        transporter=_transporter;
        transporterStruct[transporter].reputation= _rep_score;
        return transporterStruct[transporter].reputation;
    }



    function carrier_selector() public view returns (uint256) {

        return uint256(uint256(keccak256(abi.encodePacked(block.
           timestamp, block.difficulty)))%5);
    }





    function assign_transporter()public returns (address){

        // uint x=getTransporterReputationCount();



            uint i=carrier_selector();

```

```

                transporterStruct[transporterAddress[i]].reputation=
                    transporterStruct[transporterAddress[i]].reputation+
                feature_weight_1*transporterStruct[transporterAddress[i
                    ]].buyer_score_for_carrier+
                 feature_weight_2*transporterStruct[transporterAddress[
                     i]].seller_score_for_carrier;



                    transporter=transporterAddress[i];
                    transporterStruct[transporterAddress[i]].
                        carrier_already_allocated=true;


              emit VerificationSuccess('criteria met',
                  transporterStruct[transporterAddress[i]].reputation,
                  transporter);
              // emit VerificationFailure('criteria not met, assign
                  different transporter',transporterStruct[
                  transporterAddress[i]].reputation,msg.sender);


        return address(transporter);

    }



    function reputation_check() public returns(uint) {
        emit Reputation_Check('Reputation_Score',transporterStruct[
            transporter].reputation,msg.sender);
        return transporterStruct[transporter].reputation;
    }


```

```
}
```

```
//BLIND_SIGNATURE CONTRACT
pragma solidity ^0.4.26;
contract Blind_Signatures{
    address public buyer;
    address public seller;
    address public arbitrator;
    address public attestationAuthority;
    string public x;
    constructor(
        address _buyer,
        address _seller,
        address _arbitrator,
        address _attestationAuthority
    )public{
        buyer=_buyer;
        seller=_seller;
        arbitrator=_arbitrator;
        attestationAuthority=_attestationAuthority;

    }
    event Verification(string info, address entityAddress, string
       message);
    function verify(string memory message) public returns(string
       memory){
        emit Verification('Verification of blind-signatures off-
           chain ',msg.sender,message);

    }
     function parties_verified (string memory temp_message) public
        returns (string memory){
         x=temp_message;
         return x;
     }




}
```

```
//DISPUTE CONTRACT
pragma solidity ^0.4.26;
import './Rating_Contract.sol';

contract Dispute{
    Rating_Contract rt;
    address public buyer;
    address public seller;
    address public arbitrator;
    address public attestationAuthority;
    uint256 public rep_score;

    address public  rt_addr;
    string public x;

    constructor(
        address _buyer,
        address _seller,
        address _arbitrator,
        address _attestationAuthority,
        address _rt_addr

    )public{
        buyer=_buyer;
        seller=_seller;
        arbitrator=_arbitrator;
        attestationAuthority=_attestationAuthority;
        rt=Rating_Contract(_rt_addr);

    }
    event AvaialableCarriersPool(string info, address []);
    event SignatureNotVerified(string info,address);
    event ReputationDecreased(string info,address);
    event ProposedWeights(string info,address,uint256,uint256);
    event TimeWindowExceeded(string info,address);

    function carrierSelection() public {
        emit AvaialableCarriersPool('Available Carriers ', rt.
            getTransporterReputation());
        rt.assign_transporter();
    }

    function NoSignatureMatchFound() public{
        emit SignatureNotVerified('No Key Match Found',msg.sender);
        rep_score=rt.reputation_check();
        rep_score=rep_score-1;
        //right now linear decreement but one can add non-linearity
            using some feature table
        emit ReputationDecreased('Reputation Decreased',msg.sender);



    }
    function ConsensusWeightsDispute(uint _w1,uint _w2) public {
        emit  ProposedWeights('arbitrator proposed new weights for
            consensus',msg.sender,_w1,_w2);

    }
    function CarrierTimeExceeded() public{
        emit TimeWindowExceeded('Delivery Time Window Exceeded',msg.
            sender);
        rep_score=rt.reputation_check();
        rep_score=rep_score-1;

        emit ReputationDecreased('Reputation Decreased',msg.sender);
    }
}
```

Appendix B: Scripts

```
//CONTRACT LISTENING SCRIPT
const Web3=require('web3');

let web3 = new Web3('ws://localhost:8545');
var  contractAddress= "0x5C29808662b0F24eAE42aacF542FB9565260d7bd";


var abi=[
  {
    "inputs": [
      {
        "internalType": "address",
        "name": "_buyer",
        "type": "address"
      },
      {
        "internalType": "address",
        "name": "_seller",
        "type": "address"
      },
      {
        "internalType": "address",
        "name": "_arbitrator",
        "type": "address"
      },
      {
        "internalType": "address",
        "name": "_attestationAuthority",
        "type": "address"
      }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "constructor"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": false,
        "internalType": "string",
        "name": "info",
        "type": "string"
      },
      {
        "indexed": false,
        "internalType": "address",
        "name": "entityaddress",
        "type": "address"
      },
      {
        "indexed": false,
        "internalType": "string",
        "name": "message",
        "type": "string"
      }
    ],
    "name": "Verification",
    "type": "event"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "arbitrator",
    "outputs": [
      {
        "internalType": "address",
        "name": "",
        "type": "address"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "attestaionAuthority",
    "outputs": [
      {
        "internalType": "address",
        "name": "",
        "type": "address"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "buyer",
    "outputs": [
      {
        "internalType": "address",
        "name": "",
        "type": "address"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "internalType": "string",
        "name": "temp_message",
        "type": "string"
      }
    ],
    "name": "parties_verified",
    "outputs": [
      {
        "internalType": "string",
        "name": "",
        "type": "string"
      }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "seller",
    "outputs": [
      {
        "internalType": "address",
        "name": "",
        "type": "address"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "internalType": "string",
        "name": "message",
        "type": "string"
      }
    ],
    "name": "verify",
    "outputs": [
      {
        "internalType": "string",
        "name": "",
        "type": "string"
      }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "x",
    "outputs": [
      {
        "internalType": "string",
        "name": "",
        "type": "string"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  }
]
var BlindedContractDeployed = new web3.eth.Contract(abi,
    contractAddress);


BlindedContractDeployed.events.Verification({
    filter: {myIndexedParam: [20,23], myOtherIndexedParam: '0
       x123456789...'}, // Using an array means OR: e.g. 20 or 23
    fromBlock: 'latest'
        }, function(error, event){

            const blind_signature=require('./blind_signature');

            console.log(blind_signature.verify(event.returnValues['
               message']));

            console.log(event);
            BlindedContractDeployed.methods.parties_verified(
               blind_signature.verify(event.returnValues['message'])
               ).send({ from: '0
               x6f18e806d860F65028c1304194f755425f020EF6'});
        });
```

```

//Blind Signature Verification Script


function verify(_string ){

  const BlindSignature = require("blind-signatures");
```

```
    const Seller = {
      key: BlindSignature.keyGeneration({ b: 2048 }), // b: key-
          length
      blinded: null,
      unblinded: null,
      message: null,
    };

    const Buyer = {
      message: _string,
      N: null,
      E: null,
      r: null,
      signed: null,
      unblinded: null,
    };

    // Buyer wants Seller to sign a message without revealing it's
        contents.
    // Buyer can later verify he did sign the message

    console.log(' Message Signed on-chain:', Buyer.message);

    // Buyer gets N and E variables from Seller's key
    Buyer.N = Seller.key.keyPair.n.toString();
    Buyer.E = Seller.key.keyPair.e.toString();

    const { blinded, r } = BlindSignature.blind({
      message: Buyer.message,
      N: Buyer.N,
      E: Buyer.E,
    }); // Buyer blinds message
    Buyer.r = r;

    // Buyer sends blinded to Seller
    Seller.blinded = blinded;

    const signed = BlindSignature.sign({
      blinded: Seller.blinded,
      key: Seller.key,
    }); // Seller signs blinded message

    // Seller sends signed message to Buyer
    Buyer.signed = signed;

    const unblinded = BlindSignature.unblind({
      signed: Buyer.signed,
      N: Buyer.N,
      r: Buyer.r,
    }); // Buyer unblinds
    Buyer.unblinded = unblinded;

    // Buyer verifies
    const result = BlindSignature.verify({
      unblinded: Buyer.unblinded,
      N: Buyer.N,
      E: Buyer.E,
      message: Buyer.message,
    });
    if (result) {

      console.log('Buyer: Seller Signatures verified!');
    } else {

      console.log('Buyer: Invalid Seller signature provided');
    }

    // Buyer sends Seller unblinded signature and original message
    Seller.unblinded = Buyer.unblinded;
    Seller.message = Buyer.message;

    // Seller verifies
```

```
      const result2 = BlindSignature.verify2({
        unblinded: Seller.unblinded,
        key: Seller.key,
        message: Seller.message,
      });
      if (result2) {

        console.log('Seller : Buyer Verified!');
      } else {

        console.log('Seller: Buyer Not Verified');
      }
      return result+result2+'Parties Verified';
}
module.exports = { verify};
```

VITA AUCTORIS

NAME:	Vipul Malhotra
PLACE OF BIRTH:	Chandigarh, India
YEAR OF BIRTH:	1996
EDUCATION:	Chitkara University, Bachelor's of Computer Science, Punjab, 2018
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2021