A HYBRID PEER-TO-PEER FRAMEWORK FOR SUPPLY CHAIN VISIBILITY

A Thesis

Submitted to the Faculty

of

Purdue University

by

Zhijie Li

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electric and Computer Engineering

May 2017

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

This thesis work is dedicated to my wife, Shasha Wang, who has been a constant source of support and encouragement during the challenges of graduate school and life. I am truly thankful for having you in my life. This work is also dedicated to my parents, Yingyu Zhou and Kejun Li, who have always loved me unconditionally and who taught me to work hard for the things that I aspire to achieve.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Dr. Zina Ben Miled and my thesis committee members Dr. Brian King and Dr. Dongsoo Kim of the Purdue School of Engineering and Technology at Indiana University Purdue University Indianapolis for their advice and guidance. Special thanks are extended to Mr. Jeffrey Tazelaar and Mr. John Wassick for their valuable feedback. I would also like to thank my teachers and colleagues that helped me throughout my learning journey.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

Zhijie Li. MSECE, Purdue University, May 2017. A Hybrid Peer-to-Peer Framework for Supply Chain Visibility. Major Professor: Zina Ben Miled.

Current supply chain information systems are transaction-based and suffer from lack of real-time transparency. Furthermore, they are often centralized and therefore cannot adequately scale to include a large number of small and medium size companies. This thesis presents a hybrid peer-to-peer supply chain physical distribution framework (HP3D) that addresses these increasingly critical gaps in a global market. HP3D leverages the advantages of hybrid networks through flexible peers and a light-weight index server in order to share supply chain physical distribution information in pseudo real-time among stakeholders. The architecture of HP3D consists of a hierarchy of dynamic sub-networks that evolve based on market demands and digitize the transfer of goods between suppliers and customers. These sub-networks are created on demand, emulate the end-to-end movement of the shipment and terminate when the delivery of goods is completed. A variation of blockchain technology is also proposed in order to increase the security level of the proposed framework.

1. INTRODUCTION

1.1 Background

Global supply chain is a complex and dynamic group of interactions and trade-offs between suppliers, manufacturing, warehousing, carriers, and customers to deliver the right product, at the right time, and in the right condition [1]. The data to support trade-off decisions to maximize profit, not just to minimize costs, are spread across the global supply chain with ownership of the systems and underlying data varying based on the type of decision being made. More than 20 years ago the electronic data interchange (EDI) standard enabled the automated electronic document exchange between supply chain trading partners. Later, Supply Chain Operating Networks (SCONs) emerged to enable global transactions across members of the trading network based on the movement of goods. On the one hand, SCONs have brought added value to supply chain by enabling electronic data exchange. On the other hand, this added value became directly proportional to the level of participation by a given industry sector in a given SCON. Therefore, companies venturing into new industries or markets were forced to support multiple SCONs often leading to inefficiencies and high costs [2].

This thesis addresses the above limitations through a cost-effective, main stream and open peer-to-peer solution.

1.2 Supply Chains

Supply chain management consists of multiple interacting processes with multiple stakeholders. A typical supply chain includes eight main processes as identified in [3]. These processes are defined as follows:

1. Customer relation management (CRM): This process covers the relationship between the supplier and the customer including the stratification of the customers in different groups based on their demands, purchase levels and habits. CRMs document the communication between the supplier and its customers and maintain a history of this communication. This communication often includes general marketing campaigns as well as targeted campaigns, in addition to a record of feedback gathered from the customers with respect to a given product or product line. CRMs also assign and handle account managers for high-value customers and are able to facilitate plans for customer sustainment and satisfaction.

2. Customer service management: This process informs customers or potential customers about the different products including pricing, availability, shipping dates and enables the customers to track the status of their orders.

3. Demand management: This process allows the company to plan procurement, production and distribution based on forecasted market demands. While the first two processes manage the relationship with the customer, demand management is concerned with the ability of the supplier to fulfill customer demands though appropriate planning of its own procurement of raw material, production or manufacturing of the final product.

4. Order Fulfillment: This process is concerned with the distribution and delivery of the final product to the customer. It starts with the handling of the placement of the order by to customer and ends with the delivery of the goods to customer. In addition to the placement of the order and payment handling, a major subprocess of order fulfillment involves the logistics network that is responsible for the transport of the goods from the supplier to the customer. The specific focus of this thesis is on this sub-process which is defined in supply chain as the physical distribution of goods.

5. Manufacturing management: This process is also internal to the company and consists of a mapping between forecasted market demands and the ability of the manufacturer to adapt to these demands through manufacturing plant configuration subject to resource constraints. Several strategies may be used to drive this mapping including, for example, shifting resources from one production line to another.

6. Supplier relation management: This process covers the interactions of a company with its suppliers. Maintaining alternate lists of suppliers, product pricing and order lead times allows manufacturers to balance the risks of manufacturing delays against excess or stagnant inventories.

7. Product Development: This process is concerned with a sustained strategy for the development of new products and enhancement of current products as well as the reduction of time-to-market for these products. Time-to-market delays usually translate to increased product costs and loss of market shares.

8. Returns management: The goal of this process is to minimize the number of product returns from customers. This entails documenting policies and regulations associated with each product and categorizing the reasons for returns as well as implementing procedures for handling product returns.

Information systems have been developed during the last two decades to support the above processes. These information systems evolved from isolated standalone applications to integrated systems. For example, customer resource management systems (CRMs) are now widely available and deployed in many companies. These systems are able to handle both customer relation management as well as customer service management processes. CRMs are also integrated with Enterprise Resource Planning (ERP) information systems. ERPs encapsulate several core and support processes within an organization including demand management, order fulfillment, manufacturing management, supplier relation management, product development, and returns management. These processes cross the boundaries of various departments in an organization and provide a solution for an automated information flow that is mapped onto the functionalities associated with each department. For instance, the accounts payable department is involved in all the aspects of the chain including supplier relation management, product development, and returns management. Similarly, the accounts receivable department is involved with the customer resource management and customer service management.

The integration between ERPs and CRMs allows the information flow underlying the processes in an organization to extend to both its customers and its suppliers. Additionally, digital market places are now available for the exchange of this information across multiple companies thereby facilitating electronic trading. Indeed, Supply Chains Operating Networks (SCONs) have emerged as models for these exchanges and have facilitated the exchange of documents and information between trading partners including purchase orders, payment orders, bill of lading, delivery notes, etc.

While the above information systems have led to significant advances in the automation of the supply chain processes, they still have some limitations. The automated workflow embodied in the ERPs, CRMs and SCONs are able to support the timely delivery of information in both direction from the supplier to the customer as well as from the customer to the supplier. However, the increasing complexity of the supply chain, which is primarily due to the involvement of multiple parties, has started to highlight the inefficiencies of these state-of-the-art information systems. Inefficiencies can be observed in the ability of small and medium companies to penetrate new markets because of the cost associated with their participation in multiple SCONs.

Other inefficiencies are related to the real-time delivery of field information during the execution of the physical distribution segment of the order fulfillment process. The physical distribution segment is the sub-process associated with the transport of goods from the supplier to the customer. This leg of the supply chain involves other parties including carriers, brokers and freight forwarders. A given order is considered fulfilled by the supplier when it reaches its destination (i.e., the customer). Both the customer and the supplier are fully committed to the timely delivery of the goods because the customer needs the goods for its own supply chain and the supplier's payment is contingent on the delivery. Moreover, some of these goods may have high value or are associated with high risks and, therefore, visibility during the physical distribution phase is necessary in order to allow both customers and suppliers to take timely remedial action as and when needed. For example, the goods may be hazardous material whose transportation is under strict policies requiring additional insurance fees. The goods can also be medical equipment (e.g., implants) needed for a surgery or can be packaging material that needs to be delivered in time to ensure the appropriate packaging of a new batch of manufactured drugs by a pharmaceutical company.

The above scenarios show the importance for both supplier and customer to have exact knowledge of the location of their goods during shipment. However, once it leaves the supplier premises, the shipment is often handled by a third party, the carrier. Field information during the transportation segment is often relayed by the carrier through a back office communication. That is, the truck carrying the goods would need to communicate his location back to his home office. This information is then shared with the customer and the supplier. Unfortunately, this flow of information is often not performed in a timely and efficient manner making delays in information sharing with stakeholders a common practice. The focus of this thesis is to provide a solution that improves visibility for the supplier and the customer during the physical distribution phase of the supply chain. Moreover, the solution is cost-effective and scalable in order to accommodate the participation of an increasing number of small and medium companies.

1.3 Proposed System

The supply chain is an end-to-end process for the delivery of goods from a manufacturer to a customer. As mentioned above, it includes different processes that call for the interaction of different trading partners (e.g., demand planning, order fulfillment, etc.). This thesis focuses specifically on one process of the supply chain and presents an efficient, scalable and fault tolerant framework that provides real-time visibility in the physical distribution sub-process of the supply chain. The features of the proposed framework include:

- An event-based approach that leverages advances in sensor technology and the growing trend of Internet of Things (IoT).
- A hybrid peer-to-peer (P2P) architecture that can be dynamically customized to accommodate a scalable number of both small and medium companies.
- An architecture that is fault-tolerant because it minimizes potential for single point of failure and efficiently manages network traffic.
- A blockchain-enabled security approach that increases the validity and trust level of the proposed system.

The proposed hybrid P2P physical distribution framework (HP3D) is a seamless, plug-and-play, standardized digital integration system across the different stakeholders of the supply chain network. It delivers the pseudo real-time status of each shipment throughout the physical distribution process of the supply chain to the trading partners.

Each peer application is modular and consists of three tiers which is aimed at simplifying the development of the application and enhancing its maintainability. Furthermore, in order to enhance the security level of the system, an enhanced blockchain model is proposed. This model combines the concepts of a public ledger and private sub-ledgers in order to increase the privacy and trust levels in the system.

Chapter 2 of the thesis reviews several software architectures and current supply chain information management systems. Chapter 3 is a review of cryptographic tools and their use in the blockchain model. Chapter 4 discusses the design of the proposed HP3D framework including the underlying software architecture and event handling mechanisms. A blockchain-based model for HP3D is also introduced in Chapter 4. Chapter 5 describes the implementation of the proposed framework and demonstrates the use of HP3D for relevant test case scenarios. Chapter 6 outlines directions for future work and summarizes the contributions of this thesis.

2. SOFTWARE ARCHITECTURE

The industry is at a tipping point, where efficient networks are being established to enable step changes in efficiency and interoperability of supply chains across several industries. Current supply chain management systems primarily rely on centralized supply chain operating networks (SCONs) [4] and the electronic data interchange (EDI) [5] standard. Examples leading edge systems include E2Open [6] and SAP [7]. However, these systems lack cost-effective solutions for real-time transparency in the distribution phase of the supply chain and in particular with respect to the transport segment of this phase.

This chapter includes a review of previous related work by other researchers and highlights the various software models that have been used in supply chain management systems.

2.1 Client-Server Architecture

The client-server architecture is a network architecture where each node in the network consists of a computing device that is either a client or a server. The role of the server is to respond to requests from the clients and therefore it may need extended resources in terms of processing power, memory, network bandwidth and disk storage. The client in the client-server network issues the requests and presents the responses back to the user. The client-server architecture is the most commonly used service-oriented software architecture. Nearly all web services are based on this model. This architecture has the advantages associated with a centralized management and control. For instance, managing software updates to the server node is handled in one central location. Furthermore, access to the server by the clients can also be controlled through a single point in the architecture. However, it suffers from limited scalability which is constrained by the server resources. Furthermore, the client-server architecture also suffers from an asymmetric distribution of resources and a direction-biased communication [8]. Indeed, the server is responsible for processing all client requests and therefore must have high capacity and high availability. This high capacity is delivered through large storage, powerful CPUs and superior network bandwidth. High availability is delivered through the replication of these resources using for example an additional fail-over or standby server. The client, on the other hand, is usually thin with far less resources. That is most of the processing is performed on the server and only limited processing is performed by the client. The client can, for instance, be a simple web application that captures user queries and displays the results. The interactions between the clients and the server in the client-server architecture are also asymmetric as they are always initiated by the client and serviced by the server which is constantly listening for inbound client requests [8].

In summary, the main disadvantages of the client-server architecture include fault-tolerance, limited scalability, and increased maintenance cost. For instance, the server can be targeted by cyber-attacks or be subject to power outages and hardware failures making it unavailable to service requests from the clients. The server has to also accommodate an increasing number of clients which may necessitate regular compute and network capacity upgrades. Lastly, the server in a client-server model has a high maintenance cost because of its 24/7 mode of operation.

2.2 Peer-to-Peer

The peer-to-peer architecture is a network architecture where all nodes in the network have equivalent roles and privileges. Peers or nodes in the network cooperate in order to service each others requests through distributed resource sharing. The peer-to-peer architecture is an emerging distributed architecture that is commonly used for file sharing (e.g., video and audio content). This architecture addresses some of the limitations of the client-server model by decentralizing the processing of the services. Several well-known software applications use this architecture. These include bitcoin [9], bittorrent [10], Napster [11] and Skype [12]. In general, the peer-to-peer architecture can be classified into three main categories:

- Pure peer-to-peer: This unstructured network is formed by nodes that randomly connect to each others without a predefined hierarchy. This architecture does not include a centralized data or control flow manager and all exchanges are handled by the peers using message forwarding. Gnutella is an example of an unstructured peer-to-peer application [13].
- Hierarchical peer-to-peer: This model was adopted by Skype. Unlike the pure peer-to-peer architecture, the model includes a hierarchy of regular nodes and supernodes. Supernodes are selected among the peers with higher resources. They are assigned a group of peers and are responsible for establishing and managing the communication within their assigned group of peers. Communication between two peers in different groups is established through the supernodes of the respective groups.
- Hybrid peer-to-peer: This model combines the features of the pure peer-to-peer and the client-server architectures [14]. It includes a lightweight server that allows peers to lookup the IP address of other peers in the network by using the ID of the target peer. The supply chain framework proposed in this thesis uses this architecture.

2.2.1 Pure peer-to-peer

The unstructured characteristic of the pure peer-to-peer network makes it highly robust especially when a large number of peers dynamically join or leave the network. However, this characteristic also presents disadvantages. For instance the network is vulnerable to broadcast storm: an excessive network traffic caused by a large number of redundant messages especially in a highly connected network [15]. This can occur, when a search query cannot be quickly serviced by nearby peers and as a result the query is forwarded to a large number of nodes.

As previously mentioned, Gnutella [16] is an example of an unstructured P2P network. It is dedicated to online file sharing. The message exchange mechanism used in Gnutella is called flooding technique [16] where each peer forwards the request to all neighboring peers until the requested file is found. This technique is efficient for the distribution of common files that are held by the majority of peers. However, when the file is rare and only available within few peers, searching for it can lead to a broadcast storm.

2.2.2 Hierarchical peer-to-peer

In a hierarchical peer-to-peer network model, a set of nodes are designated as supernodes based on their high capacity of network or hardware resources. The peers are thus organized into a hierarchy consisting of two classes: supernodes and ordinary nodes [12]. The supernode stores the information of all nearby peers. An example implementation of this model is Skype. The selection of supernodes in Skype is based on geolocation and network resources.

Each supernode in the hierarchical peer-to-peer model is associated with a group of ordinary nodes. Once the supernodes are selected, they maintain a structured overlay network of the ordinary nodes assigned to their group. An ordinary node sends requests to its assigned supernode. Once the request is accepted, the issuing node and the servicing node establish a direct communication channel [12].

The hierarchical peer-to-peer network structure is less robust than the unstructured peer-to-peer network as each supernode must maintain a list of information related to the ordinary nodes in its group. Moreover, the communication channel between a supernode and an ordinary node can become a bottleneck.

2.2.3 Hybrid peer-to-peer

The hybrid peer-to-peer network model can be viewed as the combination of the peer-to-peer and the client-server models. This architecture consist of an index server and the nodes. The index server is a lightweight server and allows the nodes to request the IP address of the other nodes in the network. Once the IP of the target peer is retrieved, connection is directly established between any source-target pair of nodes. The server in the hybrid peer-to-peer network is labeled lightweight because of the simple functionalities underlying its software application. Indeed, this application is limited to a) receiving queries from the nodes in the network consisting of the ID of a target node, b) retrieving the corresponding IP from the index server's local database and c) returning the response back to the node that issued the query. Example applications that use this model include Spotify [17], bittorrent and Napster. Spotify is a peer-to-peer music sharing software application. The application is supported by two main data centers. A node uniformly chooses which data center to connect to. Each data center has an independent peer-to-peer overlay network. There are two different ways of finding a particular node in the network. The first uses a tracker deployed in the Spotify server and the second consists of issuing a query over the overlay network. For the first mechanism, the sever holds a table that associates a music file with a user. If another user is searching for a given music file, he/she needs to lookup the user who holds this file through the above mentioned table. It will then establish a connection with that user. This mechanism guarantees that there is no broadcast storm in the network. However, the server may become a bottleneck during high network traffic. Therefore, this method is suitable for small number of inquiries to which the server can quickly respond.

2.3 Supply Chain Information Systems

E2Open [6] and SAP [7] are examples of cutting edge supply chain information management systems. These software applications are cloud-based. They also provide well-developed transaction-based functionalities that support the exchange of shipment information and documents between trading partners. Given that current business environments have a large amount of traffic and numerous stakeholders, a centralized, cloud-based solution may become inefficient with respect to both scalability and affordability. The latter limitation is a major barrier for small and medium-sized companies. The HP3D system proposed in this thesis is based on a hybrid peer-to-peer architecture and attempts to address both of these limitations.

Current supply chain information systems also use the electronic data interchange (EDI) [5] standard as a common format for the messages being exchanged by the partners in the network. HP3D uses this standard in order to maintain interoperability with existing systems.

3. CRYPTOGRAPHY AND BLOCKCHAIN

Internet and information technology usage is increasingly penetrating every aspect of our daily lives. As a consequence, we are becoming progressively more and more exposed to computer and network security breaches. Computer security, also known as cyber security or IT security, is a field of research that focuses on the protection of computer or network systems from attacks that may materialize in the theft or damage to the hardware, the software or the data [18]. Because of these potential threats and associated damages, researchers have been focusing on computer and data security. As a result, numerous cryptography mechanisms [19] such as digital signatures and encryption algorithms were proposed. More recently, data security has been gaining importance and became a significant component of computer security. Data security can be viewed from different aspects including integrity and confidentiality and there are several approaches that are used to enhance data security. This chapter reviews the cryptographic tools that are relevant to the proposed system.

3.1 Cryptographic Tools

Cryptography is a collection of mathematical or programing methods that can protect data integrity and confidentiality. Different cryptographic tools have different features. For example, a cryptographic hash function can enhance data integrity and an encryption algorithm can maintain data confidentiality.

3.1.1 Integrity

Data integrity is concerned with two different aspects, data itself and its sender. Bishop [20] defines data integrity as the trustworthiness of data and/or sources. To maintain data integrity, the recipient has to be able to verify that the data was not modified during transmission. In addition, the recipient must be able to ascertain that the sender specified in the data package is the same as the actual sender. Mechanisms for maintaining data integrity fall under two main categories: prevention and detection. Prevention mechanisms focus on preventing an unauthorized user from modifying the data. Detection mechanisms do not attempt to prevent data modification during transmission, but instead report to the user when the data has been altered. For the purpose of this thesis, we are focusing on detection mechanisms since the network underlying the proposed system is public. The methods used in the proposed system to maintain data integrity include cryptographic hashing and digital signature both of which are discussed later in this chapter.

3.1.2 Confidentiality

Confidentiality can be defined as keeping the information contained in the data accessible only to authorized users. The data is public while it is being transmitted through the Internet. Mechanism are available to enable the sender to "scramble" all the bits in the data thus making it unreadable. The receiver can apply a reverse operation to reproduce the readable data from the "scrambled" data. In this context, "readable" implies the ability to interpret data in some meaningful way. The procedure of "scrambling" the data bits is called encryption. Re-producing readable data is called decryption. Usually encryption and decryption require a key which can be either symmetric or asymmetric.

3.1.3 Terminology

In order to describe different cryptographic mechanisms, the definitions of domain-specific terms are introduced below:

- Plain-text refers to the text that is "readable". In other words, the information that is carried in the text can be easily interpreted.

- Encryption is the process of ”scrambling” the bits in the plain-text in order to make the data ”unreadable” without knowledge of the associated secret. In modern cryptography system, the secret usually refers to a cryptographic key used in the encryption process.
- Cipher-text is produced by an encryption process. The information carried in the text cannot be easily interpreted by someone who does not have access to the secret/key.
- Decryption is the reverse operation of encryption. It is the process that reconstructs the plain-text (meaningful data) from a given cipher-text (”scrambled” data).

With the above definitions, we can mathematically describe the cryptographic operations of interest. Let M denote the plain-text, C denote the cipher-text and K denote the security key. Given an encryption function/algorithm $E_k()$, the cipher-text C can be generated from the plain-text M by using the equation below:

$$C = E_k(M) \tag{3.1}$$

Similarly, given the decryption function/algorithm $D_k()$ the plain-text is derived from the cipher-text using the following equation:

$$M = D_k(C) \tag{3.2}$$

3.1.4 Public Key Cryptography

A secret key is needed in order to perform both encryption and decryption. Initially, we assume that there is no key shared between two entities. A key issue is the distribution of the key to each involved entity without revealing it to external parties. The Deffie-Hellem Key Exchange protocol is the first and most well-known key exchange agreement [21] that addresses this issue. In the prototype of the proposed framework, the Deffie-Hellem protocol is used to exchange a key between a mobile device or another device that need to be authorized (i.e., a peer) and the index server as shown in Figure 3.1. Other more elaborate key exchange mechanisms are available such as RSA [20]. The use of the Deffie-Hellem mechanism as part of the implementation of HP3D is for illustration purposes. It can be easily replaced by other mechanisms that may be available through advanced public key infrastructures.

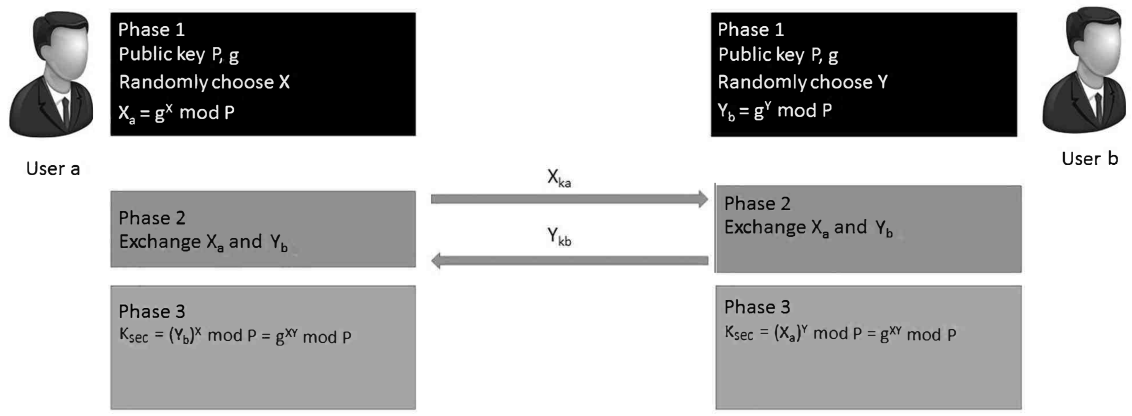

Fig. 3.1. Diffie-Hellman Key Exchange

The exchange starts with a public key (i.e., a publicly known key) which consists of two parts: a prime number P and a generator g. This key needs to have certain properties in order to ensure that the protocol is secure. First, the number of bits in P should meet the length requirements stated in [22]. Second, the generator g has to be within the range *1* to *P-1* and has to satisfy the following:

$$\forall \quad X \in [1, P-1], \quad \exists \quad i \in [0, P-1] \quad | \quad X = g^i \quad mod \quad P \tag{3.3}$$

Based on the above setup, users a and b can generate random numbers denoted by X and Y, respectively. The first phase of the protocol starts by calculating two numbers X_a and Y_b using the following equations:

$$X_a = g^X \quad mod \quad P \tag{3.4}$$

$$Y_b = g^Y \quad mod \quad P \tag{3.5}$$

In the second phase, the two users exchange X_a and Y_b. The third phase consists of users a and b calculating the shared secret key, K_{sec}, independently by performing:

$$K_{sec} = Y_b^X \quad mod \quad P = g^{XY} \quad mod \quad P \tag{3.6}$$

$$K_{sec} = X_a^Y \quad mod \quad P = g^{XY} \quad mod \quad P \tag{3.7}$$

This calculation will allow the users a and b to have access to the same secret key, K_{sec}. Both users have the same public key P and g. The secrete key, K_{sec}, can then be written as $(g^X \ mod \ P\ )^Y \ mod \ P$ or $(g^Y \ mod \ P\ )^X \ mod \ P$ and both expressions can be simplified to $g^{XY} \ mod \ P$ [21] as shown in Equation (3.7).

A possible external attack on this algorithm is to eavesdrop on the connection between a and b, in order to capture the values of X_a and Y_b. Since the adversary also has access to the public key P and g, a brute force approach can be used to determine the values of X and Y by calculating:

$$var_i = g^i \quad mod \quad P \quad \forall \quad i \in [0, P-1] \tag{3.8}$$

The adversary can try different values of i until he/she finds a var_i that is the same as X_a or Y_b. Once this is performed, i is established as the value of either X or Y. The adversary can then perform the same operation as a or b to find the secret key, K_{sec}. If the prime number P is large enough, the brute force method becomes impractical. Indeed, given X_a, Y_b, P and g, solving g^{XY} mod P is called the discrete logarithm problem [19] which is considered a hard problem.

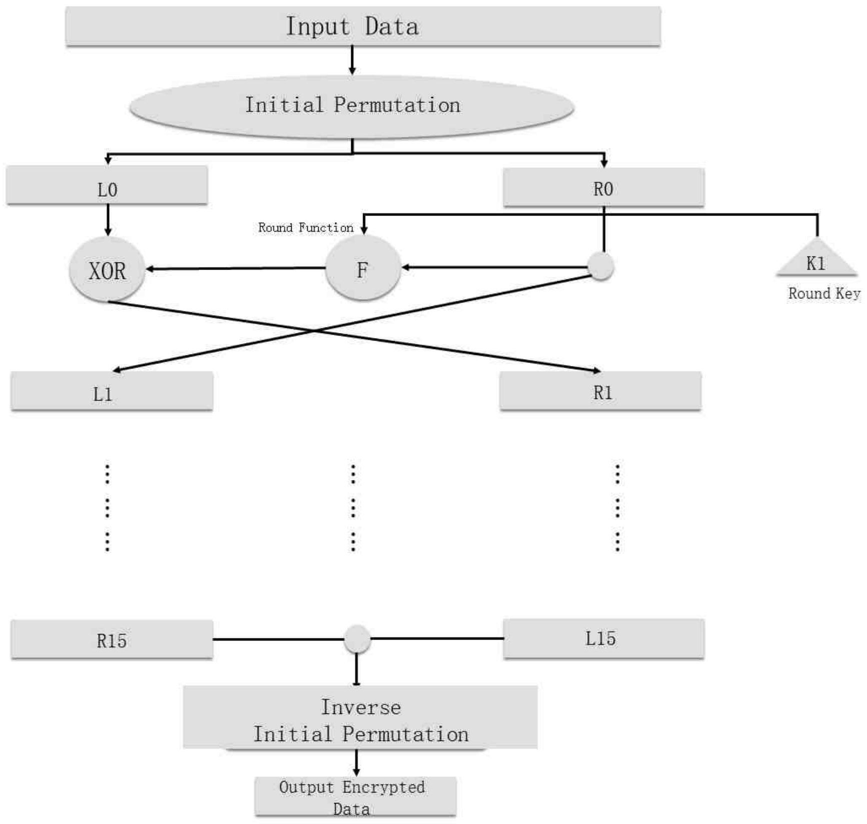

Fig. 3.2. Data Encryption Standard(DES)

The Diffie-Hellman key exchange protocol can only distribute a symmetric key to both sender and recipient of the information exchange. The other aspect of data confidentiality consists of the encryption/decryption of the information being exchanged using the key. For this purpose, and again for illustration purposes, we use the Data Encryption Standard (DES) which was adopted in 1977 by the National Institute of Standards and Technology(NIST) [21]. Other encryption mechanisms that offer higher levels of security are available (e.g., 3 DES [23]). DES is used in the implementation in order to illustrate the use of encryption and decryption in the HP3D prototype. DES is considered as a block cipher algorithm which means that the algorithm breaks the data in to small blocks and encrypts each block individually and then concatenates all of the blocks together. The recipient has to decrypt the data in the same manner. The block size can be 64-bits and the key length 56-bits. The encryption process for each block in this case is shown in Figure 3.2. There are a total of 16 rounds in each encryption. During each round, the algorithm breaks the data into two halves: left and right. Furthermore, for each round a round-key is generated using the original key. A substitution is performed on the left half of the data. This is done by applying the round function *F()* to the right half of the data. Subsequently, an exclusive-OR is performed on the output of the round function and the left half of the data. The round function has the same structure in every round. However, it uses a different round-key in order to "scramble" the data to the highest extent possible. The two halves of data are interchanged at the end of each round.

3.1.5 Digital Signature

Digital signatures and cryptographic hash functions are used in the proposed system in order to maintain data integrity. Digital signature is used to maintain the integrity of the source of the data. The authenticity of the sender is verified by examining the corresponding signature.

There are several types of digital signature scheme. RSA [20] and Elliptic Curve Digital Signature Algorithm (ECDSA) [24] are two examples of popular public key digital signature methods. RSA is used to verify the sender by using the sender's public key. In order to setup RSA, several parameters are needed. First, two prime numbers P and Q are selected at random. Let $n = PQ$, the Euler totient function of n is defined as:

$$\phi(n) = (P-1)(Q-1) \tag{3.9}$$

This function corresponds to the number of elements from *1* to *n-1* that have the greatest common divisor with n equal to *1*. Two additional random numbers α and β are selected and must satisfy the following equation:

$$\alpha\beta \equiv 1 \quad mod \quad \phi(n) \tag{3.10}$$

The public key can be defined as either K_{pub}= *(n, β)* or K_{pub} = *(n, α).* In this context, the parameters P, Q and α (or β) are secret and constitute the private key K_{priv}. A signature of a give message M can be generated by the following equation:

$$Sig_M = sig(M, K_{priv}) = M^{\alpha} \quad mod \quad n \tag{3.11}$$

The verification process can, in turn, be performed as follows:

$$ver(Sig_M, K_{pub}) = {Sig_M}^{\beta} \quad mod \quad n \tag{3.12}$$

The signature is only accepted when *ver(Sig_M,K_{pub})* = *M.* Therefore, the adversary cannot produce the same message signature unless he/she knows the private key, K_{priv}. Producing the private key using the public key can be split into two different problems: discrete logarithm problem and factoring problem [19]. Both are classified as hard problems.

3.2 Supply Chain Security and Blockchain

A number of cryptographic and security models have been proposed for securing the supply chain. Some have focused on protecting the content of the load (i.e., containers), such as in [25]. Other secure multi-party computations models focused on e-auctions and required additional communication rounds that were computationally intensive [26]. Many have proposed using RFID in their supply chain protocol. However, this latter approach can impact privacy [27] [28] [29] and often introduces additional difficulties, such as the need to re-encrypt data which subsequently makes information tracking problematic. The goal of the proposed system is to support the integrity of the physical distribution phase of supply chain, protect shipment custody information, and provide means for the tracking of this custody in a scalable, secure and reliable way, all the while protecting the privacy of the participants. Towards this purpose, we have developed a new cryptography model based on the block chain [30] technology.

At the core of the blockchain [30] technology is a distributed public ledger with two types of transactions: a single genesis transaction which creates value and a transfer transaction that transfers value from one party to another. In the blockchain taxonomy, this latter transaction is called a smart contract. Each transaction is digitally signed by the issuer and posted to the public ledger. A group of transactions are then collected into a block, the block is validated by a third party (a miner) and is locked. This mechanism represents the strength of the blockchain technology. Each block in the chain is immutable since it is linked to its predecessor and any change to any of the blocks invalidates all the blocks downstream in the chain. Furthermore, the more mature the block is (i.e., the longer it has been in the public ledger chain), the greater is its integrity.

Each peer participating in the network keeps a copy of the public ledger and every time a new block is created, it is broadcasted to all the peers that add it to their local copy of the ledger. In general, participation in the public ledger is anonymous as each party is identified by a digital ID. From a business perspective, issuers and beneficiaries are encouraged to participate in the ledger because of this anonymity, in addition to the lack of a central controlling party, reduced transaction fees and the real-time execution of the transactions. Miners are also incentivized because they receive a fee for every block they validate.

Despite the level of protection in the traditional blockchain, the approach is plagued with an increasing level of criticism. While the approach is attractive because of its underlying freedom of trade and anti-regulation, it makes risk/flexibility trade-offs hard to manage. For instance participants are not protected against mining of the public ledger for trends and transaction patterns.

Bitcoin [9] was the first decentralized digital currency and one of the most wildly known application of the blockchain technology. Bitcoin is based on a peer-to-peer architecture and allows online payments to be sent directly from one party to another without going through a financial institution. In Bitcoin, a public ledger and a proof-of-work concepts are used to ensure the integrity of the transactions and to incentive the miners. Transactions are chained and stored in the public ledger. Every peer has a copy of the public ledger, and every update to the ledger is broadcasted to all peers in the network. Moreover, every transaction is verified by one of the peers in the network (i.e., miner). When a transaction is posted, the miner will back track the chain of transactions in order to find the sender's latest balance. The transaction is deemed valid when the last balance is greater than the current transfer amount. Once this is confirmed, the miner chains the transaction to the previous transaction by calculating a hash value for the current transaction using the previous transaction's hash value. This verification process is called the proof of work which is also used to time-stamp the transactions. Multiple miners can attempt to verify a single transaction. The miner who finishes the verification the first will receive the associated reward in the form of a transaction fee [9]. A variation of the blockchain technology is used to support the transfer of information in the proposed HP3D framework. This aspect along with the design of HP3D are discussed in the next chapter.

4. SYSTEM DESIGN

The architecture of the proposed HP3D supply chain framework is shown in Fig 4.1. It is based on a collection of purpose-centric customized sub-networks that can be configured dynamically in real-time. This is a departure from the traditional transaction-based SCONs or ERP systems. HP3D allows stakeholders to share information related to a given shipment and provide them with pseudo real-time visibility in the physical distribution segment of the supply chain.

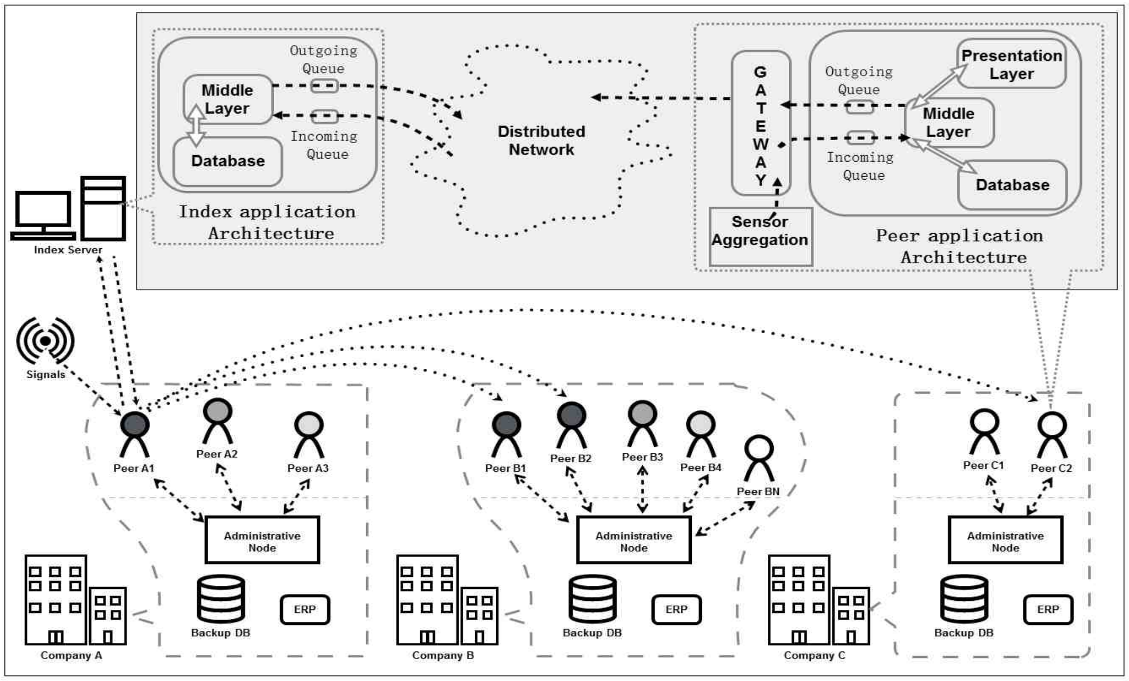

Fig. 4.1. System Architecture

The network model adopted in HP3D is based on the hybrid peer-to-peer architecture. Previous projects successfully used the hybrid P2P model for selected applications in other sectors. For instance, the model was used to share music among different peers [11]. In the proposed HP3D, the P2P network allows any registered user to enroll in the system. Once the user is registered, he/she has the ability to dynamically establish sub-networks with trading partners. The software architecture that enables this ability is described in this chapter.

4.1 Architecture

The overall architecture of the system contains several components namely index server, node, administrative node and external monitor. The roles of these components in the system are discussed next:

- Index server: the role of the index server is to dynamically collect and share the IP addresses of the peers. This server is lightweight and does not participate in the data exchanges between peers. The index server is responsible for providing the IP address of the target peer to the requesting peer. In order for the server to maintain this information up-to-date, it is designed to receive a heartbeat message from active peers at regular intervals. The rate of the heartbeat message can be adjusted depending on whether access is being performed from a mobile device or computer. The index server processes this heartbeat message and updates the corresponding peer's information. In order to make the index server lightweight, all types of requests have been designed to reduce the involvement of the index server in the communication among peers as well as reduce the size of the messages being exchanged between the index server and the peers.

- Peer: A peer is the most common node in the system. It can be a mobile device or a desktop computer. Shipment update messages are shared among these peers. While each of the peers can assume a different role for each individual shipment, all peers have a unified architecture. A given peer may assume more than one role with respect to different shipments. For instance, a carrier in one shipment can also be a customer in another shipment.

- Administrative node: In general, peers are expected to have limited resources. However, one of the peers must be designated as the interface with the in-house ERP and would therefore need extended processing and storage resources. This peer is labeled “administrative node”. Each partner will have a designated administrative node. This node interacts with the enterprise resource planning system (ERP) of the partner and is able to retrieve order information and warehouse sensor information. In addition, it participates in all the shipment-centric sub-networks associated with a given partner. As such, it is able to act as a persistent record for all messages related to shipments involving the corresponding partner. One of the main motivation for the administrative node is its function as a data source when active peers drop out of the sub-network. For instance, if a peer experiences a loss of connectivity, once it reconnects, it can retrieve the most recent shipment updates from the administrative node in its organization.
- External monitor: The external monitor is mainly responsible for the posting of the geolocation data to the semi-public ledger which is used to track the shipment location in the proposed HP3D.

4.2 Support Processes

During the design phase, few processes have been identified to enable the information flow in HP3D. These processes consist of three core processes and three support processes. This section discusses the support processes. Before shipment information can be shared among peers, the shipment-centric sub-network has to be established. That is, peers need to know information about other peers that are involved in the same shipment. This information is normally included in a purchase order which is issued by the customer's ERP system and shared with the other trading partners. The administrative node of the customer will retrieve the purchase order from the customer's ERP and use the information to establish the shipment-centric sub-network.

Once the network is established, peers can send shipment update information to each other. This process is labeled query IP Address and involves querying the index server about the IP address of other peers. The workflow of the query IP Address process is shown in Fig 4.2

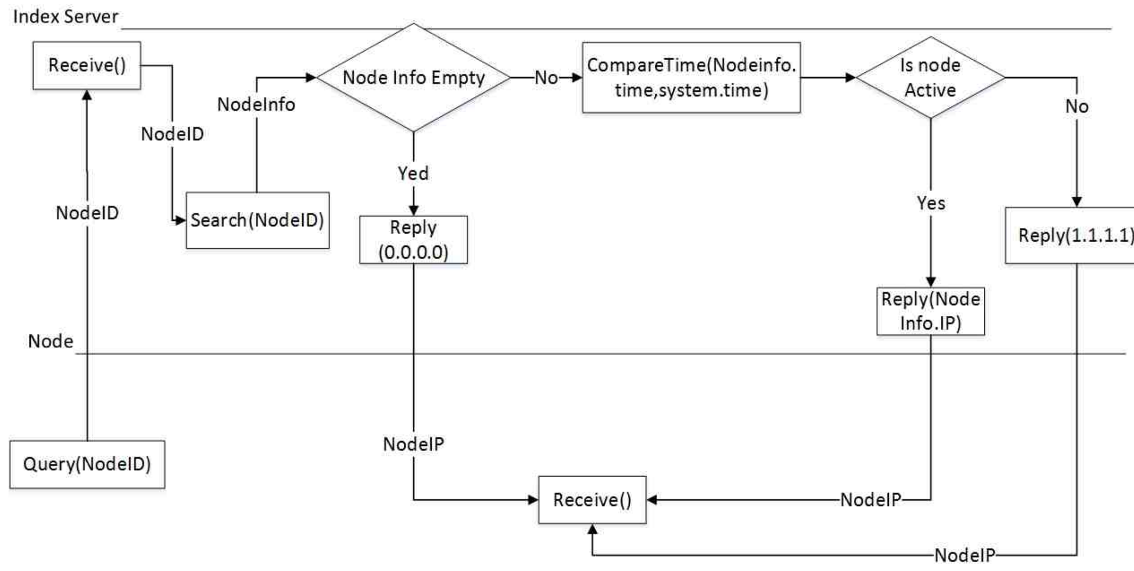

Fig. 4.2. Query IP Address Process

A peer initiates the process by sending the index server the ID of the target peer. When the index server receives the ID, its searches its local database for the corresponding peer IP. The index server will reply to the requesting peer with a 0.0.0.0 IP address, if the target peer ID does not exist in the database. Otherwise, the index server will compare the timestamp of the target peer with the current system time. It will reply with 1.1.1.1 IP address to the peer if the difference between these two times is greater than some preset threshold indicating that the target peer is not active. If the target peer is active, the index server replies with the desired target peer's IP address. This support process is used in all exchanges including any kind of shipment update messages.

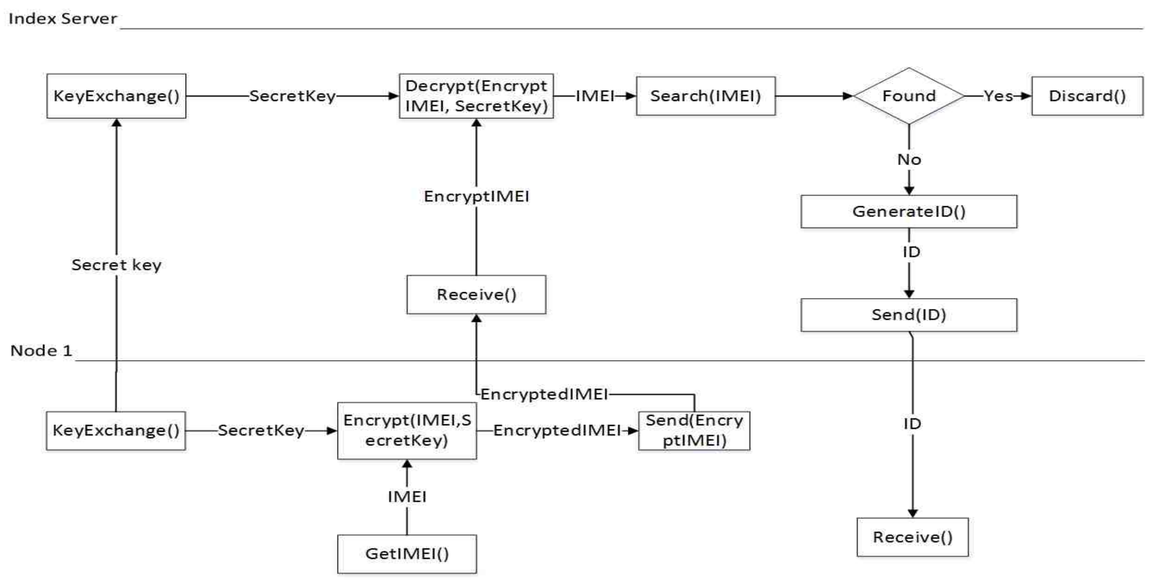

Fig. 4.3. Mobile Verification Process

In order to keep unauthorized users from joining the network, an authentication method is needed. The mobile verification process enables the identification of a valid mobile device when it joins the network. The focus in this case is on mobile devices, since desktops and other connected devices may be authenticated through their organizational infrastructure. The mobile verification process uses the IMEI number to identify a mobile device. This number is unique for every device. It is further assumed that the index server has prior knowledge of the IMEI numbers of valid mobile devices (e.g., stored in the index server's local database during the registration).

The mobile verification process consists of two different sub-processes. The first sub-process is a key exchange protocol and the second sub-process is the validation of the IMEI number of the mobile device. The mobile verification routine is shown in

Fig 4.3. Assuming that the mobile device and the index server have exchanged the necessary keys to secure the subsequent exchange of information, the mobile device will retrieve its IMEI number and encrypt it. The encrypted IMEI number is then sent to the index server. The index sever decrypts the IMEI number and use the number to search its local database. The mobile device will be identified as valid only when the IMEI number is registered in the index server's database. The index server generates an ID for the mobile device and returns it to the mobile device if the device is valid. If the device is not found in the database, an error is generated. However, no message is returned to the issuing device in order to prevent the index server from being overloaded with erroneous or malicious requests.

4.3 Core Processes

The proposed HP3D has three core processes namely heartbeat message, local sensor message and GPS update. The heartbeat message is sent by the active nodes in the network. The workflow of this process is shown in Fig 4.4.

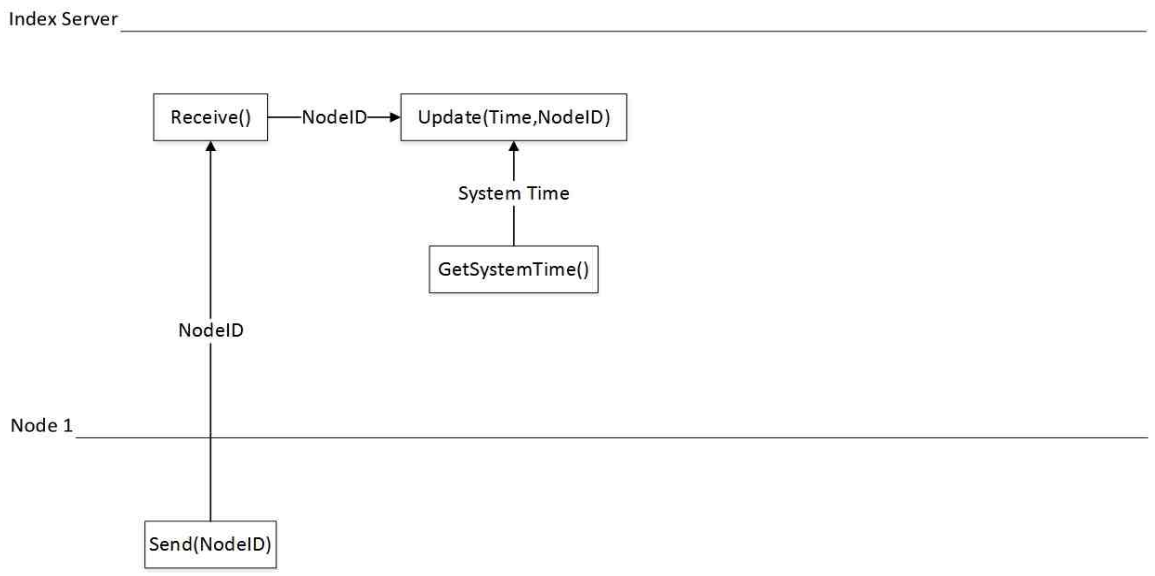

Fig. 4.4. Heartbeat Message Process

An active node sends the index server its ID. The index server will retrieve the current system time and update the node's timestamp in the database. The heartbeat message is sent by each node in the network periodically with an adjustable rate in order to fit both resource and time delay requirements.

The second core process is related to the sharing of internal sensor information with trading partners. Companies may have sensors internal to their local operation that they may be willing to share with their trading partners (e.g., sensors in the loading points). The administrative node in the company is responsible for the acquisition of these signals and for broadcasting the related messages to the other nodes in each shipment-specific sub-network. Fig 4.5 shows the workflow of the internal sensor update process.

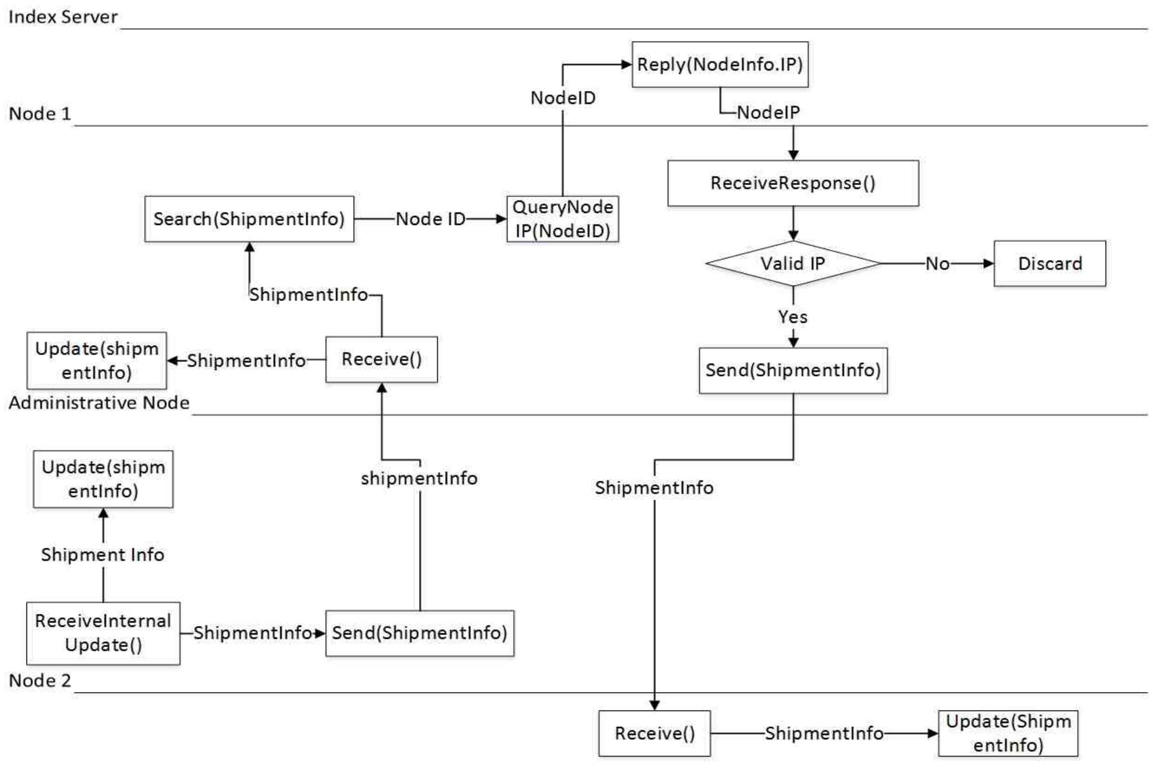

Fig. 4.5. Internal Sensor Update Process

The internal update message is generated by the administrative node in a company and sent to the corresponding nodes in the company. The nodes will update their local database and also broadcast the message to other nodes involved in the shipment. In order to do so, the nodes will query the index server for the most up-to-date IP address of other nodes. After querying the index server, the node will send the update message to those valid IP addresses returned by the index server. This process is important, for example, for the supplier to notify the carrier when a shipment is ready to be picked up, since the internal sensor can only be accessed by the administrative node of the supplier.

The third core process is the GPS update process which allows the trading partners to share the geolocation status of a shipment. This process relies on the blockchain technology. The workflow starts with a node querying the semi-public ledger which is maintained by the monitors. The external monitor will search its local database for the target truck ID and reply to the requesting node with the latest geolocation update of the corresponding truck. In order to broadcast the update message to the trading partners, the node will first query the index server about the target nodes' most recent IP addresses using their IDs. The node will then send the update message to all valid IP addresses that are returned by index server. The recipient nodes will in turn update their local database once they receive the update message. The recipient nodes include the administrative node which also receives the message and updates its own local database.

As mentioned earlier, the GPS update in the proposed design is done by the external monitors. However, due to implementation time constraints, the GPS update process is being emulated in a laboratory environment by using a mobile GPS emulator application developed for testing purposes. The mobile GPS emulator acts as an external monitor with few differences. For instance, instead of a node querying the external monitors, the GPS emulator will periodically send GPS updates to one of the nodes in the shipment-centric sub-network (e.g., carrier). The node will then broadcast the updated GPS information to the other nodes.

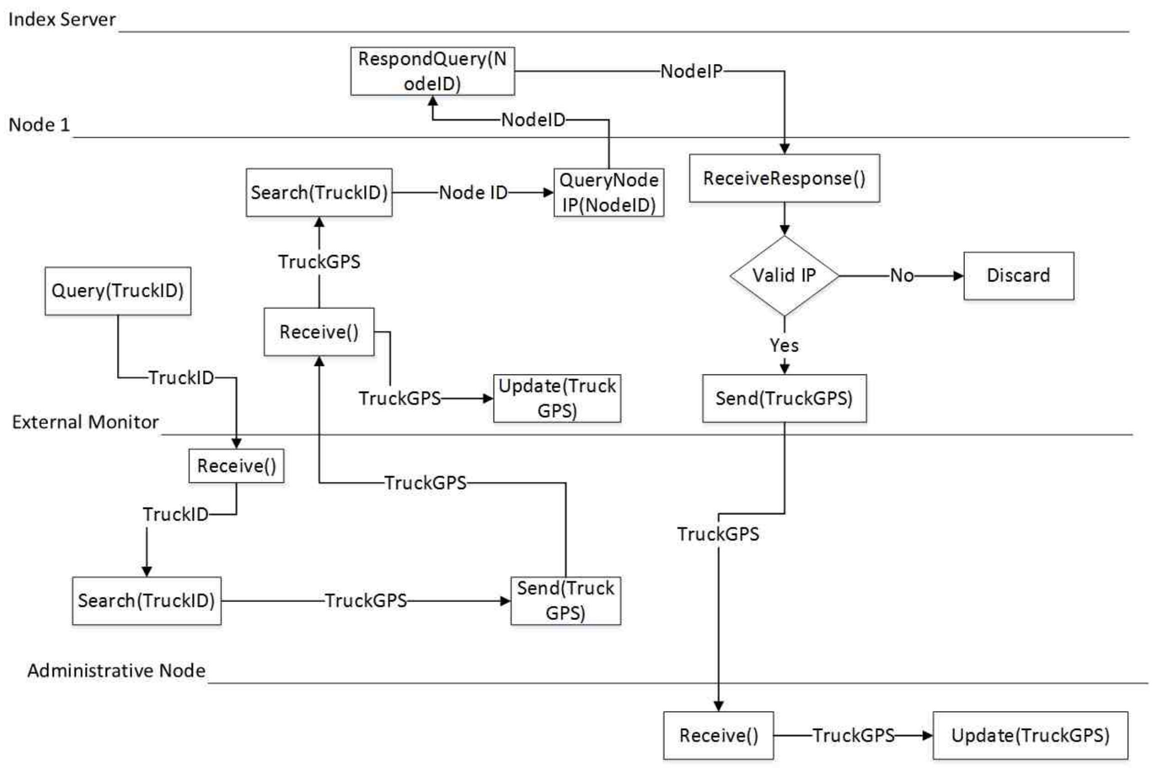

Fig. 4.6. GPS Update Process

4.4 Cryptographic Blockchain Model

Despite the availability of the verification routine which prevents unauthorized user from accessing the network, few security issues still remain in the proposed HP3D. For example, the truck driver can fake the geolocation information by using a GPS spoofing software. In order to overcome these issues, an enhanced model that uses external monitors and the blockchain technology is introduced.

Each shipment-centric sub-network will have a sub-ledger that is private to the trading partners. In addition, the framework relies on a semi-public distributed ledger that is maintained by external monitors. External monitors have been previously proposed by others (e.g., Blockfreight [31]). In order to improve the trust level of the blockchain-based framework, the semi-public ledger in the proposed model contains the geolocation information associated with the trucks during the transport phase of the shipment. It can only be updated by external monitors. Furthermore, each of the monitors has a pre-distributed public key and private key pair. Any record that is posted to the semi-public ledger is signed by the monitor using his/her private key. The records can be verified using the corresponding public key. Using this protocol, an external monitor can update the geolocation information of a given truck in the semi-public ledger by broadcasting the information to all external monitors in the network.

While the semi-public ledger is common to all shipments. Every shipment is associated with a unique private sub-ledger. This sub-ledger contains shipment information including supplier, customer, carrier as well as packing list and the ID of the truck associated with the shipment. The private ledger is only readable and writable by the trading partners for the specific shipment thereby protecting the privacy of the partners.

In the traditional blockchain model, blocks consist of transactions. However, in the proposed model, for both the semi-public ledger and sub-ledgers, blocks consist of different types of events. These ledgers include three types of events as shown in Table 4.1:

- Genesis event: This is the event that starts a new shipment. It corresponds to the delivery note and essentially establishes the supplier as the custodian of the shipment. The event also includes the shipment ID, a timestamp, supplier location and the beneficiary. The format of the data is based on the EDI 214 standard.

- Custody event: An event that documents the current custody of the shipment. This custody can remain the same or indicate a transfer from one party to another (e.g., from Dow, the supplier to Carrier Inc., the Carrier). In a custody event, the timestamp and geolocation refer to the status of the shipment when the event occurred.

- Monitoring event: An event based on physical proximity and issued from an external party which can be an element of the transportation infrastructure (e.g., a traffic light, toll booth, a future smart road mile marker or another vehicle). In this thesis, we consider only the case where monitoring events are issued by users in other vehicles. Future extensions can consider other types of external monitors.

Table 4.1.
Events types and associated smart custody contracts

Genesis Event Event: Dow ->Dow Timestamp: 02/01/2017 Geolocation: Long: X, Lat: Y BN: Pharma Inc a) Genesis event if this is the first event or Custody event with no transfer
Custody Event Event: Dow -> Carrier Inc. Timestamp: 02/02/2017 Geolocation: Long: X, Lat: Y BN: Pharma Inc. b) Custody event with transfer
Monitoring Event Event: Monitor ID Timestamp: 02/09/2017 Geolocation: Long: X, Lat: Y BN: N/A c) Monitoring event

The custody events form the shipment-centric sub-ledger which is private to the supplier, carrier and customer. The monitoring events are submitted by external monitors to the semi-public distributed ledger. Each event in the sub-ledger is digitally signed by the issuer and a set of events are combined into a block. The block is then validated by searching in the semi-public ledger for the corresponding truck ID. Each validated block is shared with the other peers in the shipment-centric subnetwork and linked to the previous block in the chain thereby creating a custody ledger chain.

Each party, including external monitors, will have a pair of public-private keys (e.g., $\mathrm{S_{PUB}}$ and $\mathrm{S_{PRIV}}$ supplier public and private key, respectively). The private sub-ledger contains shipment information that is encrypted using a public key agreed upon by supplier, carrier(s) and customer. These are the only parties that can post to the sub-ledger which consists entirely of custody events (Table 4.1). Furthermore, each posted custody event is digitally signed and appropriately encrypted by the issuer.

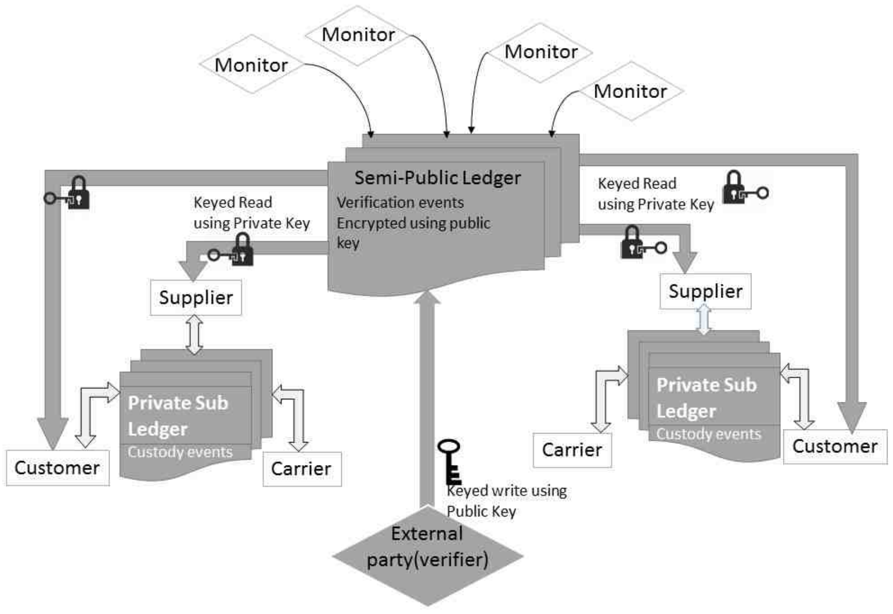

Fig. 4.7. Cryptographic Model

Each external monitor will also have a public/private key pair. A potential mechanism for generating events can be as follows: When a delivery truck is in the physical proximity of an external monitor (e.g., another vehicle), a radio communication channel is established between the truck and the monitor. The monitor will then send a challenge (a nonce) to the truck, who then signs it with his private key (C_{PRIV}) and sends the signature back to the monitor. The external monitor verifies the signature using the truck's public key (C_{PUB}). She will then sign the current geolocation of the truck using her private key (M_{PRIV}). The posting of this information (geolocation and signature) to the semi-public ledger can be performed in multiple ways. For example, the monitor can post the information directly to the semi-public ledger or the monitor can send this information back to the truck which will then post it to the semi-public ledger.

Under this cryptographic model (Fig 4.6), the monitors are not privy to the supplier or the customer private information. Moreover, the external monitors cannot read the semi-public ledger. Given that this ledger contains information about a large number of shipments that are handled by numerous carriers, this additional restriction prevents the monitors from mining the ledger for information that can be used to infer sensitive trading patterns.

One challenge of the proposed approach is the reconciliation between the fact that data in the sub-ledger will have high integrity (i.e., high level of trust). However, the data in the semi-public ledger will have entries from external monitors and the accuracy/truthfulness of these entries can vary. The goal is to develop high valued information from these combined ledgers in order to approximate the current state of the shipment. Let $\theta(t_1)$ represent the actual state-geolocation and custody - of the shipment at time $t=t_1$. An ideal solution will deliver $\theta(t_1)$ at time t_1 to the supplier and the customer. The problem is that this state is not directly available to them. What these trading partners have are the original shipping documents, as well as under the proposed framework information from the sub-ledger and semi-public ledger.

The customer and supplier can analyze these ledgers and produce a "view" - VIEW-supplier(shipment, sub-ledger, semi-public ledger; t_1) which closely approximates $\theta(t_1)$.

The proposed blockchain model is an initial contribution towards increasing the validity of the information reported to the supplier and the customer from the field. Additional research is needed to consolidate events from different monitors for a single truck or multiple trucks within a close geographic location. Furthermore, approaches that allow the efficient search of the semi-public ledger for a given truck-ID are needed. Given the number of events in the semi-public ledger, this search may take a long time if it is not supported by an adequate indexing mechanism.

5. SYSTEM IMPLEMENTATION AND TESTING

The core and support processes of HP3D have been implemented on multiple commonly used platforms in order to demonstrate the practicality of the proposed design. In addition, a GPS emulator was developed in order to allow the testing of the GPS update process. Moreover, the applications for the nodes in the network, including the administrative node, have also been implemented. This implementation was based on Javascript and HTML for the presentation tier, MGO and Golang [32] for the middle tier and MongoDB [33] for the data tier. This chapter discusses the details of this implementation.

5.1 Data Structure

Processes in HP3D rely on messages that are exchanged by various components of the system. There are two main types of messages, node request and shipment update. The first type of messages is being exchanged between nodes and the index server. The second type is exchanged among different nodes. The data structure used to support the first type of messages is shown below:

```
type indexServer struct{
    ReqType int
    timestamp time
    IpAddr string
    key []byte
    IMEI []byte
    ID bson.ObjectId ‘bson: ‘‘_id,omitempty’’ ’
}
```

This structure is implemented in Golang and consists of six fields.

- *Reqtype* is an integer value which defines the type of a request. The index server checks this field first when a request is received.
  - *Reqtype* 0 corresponds to the heartbeat message
  - *Reqtype* 1 corresponds to an IP address query
  - *Reqtype* 2 corresponds to the verification by using the IMEI number
- The *timestamp* field is a system time that is used for a *Reqtype* 1 request.
- The *IPAddr* field is a string that contains the IP address of a target node. If the target node is not found or it is inactive an IP of 0.0.0.0 or 1.1.1.1 is returned in the *IPAddr* field, respectively.
- The *key* and *IMEI* fields are byte arrays which are used in the mobile verification routine.
- The *ID* field contains a node ID which is generated by the index server. The ID is unique throughout the system. The *ID* is used to identify the IP address of a particular node.

The data structure used to support the second type of messages (i.e., shipment update) follows the EDI214 [34] standard. EDI214 is a global standard for supply chain message exchange. The standard is complex and include information about capital flow, contractual obligations and payment terms. It also involves several other parties including manufacturers, assemblers, distributors, freight forwarders, brokers, financial institutions, etc. For the purpose of this thesis we limit the trading partners to the simple case of supplier, customer and carrier. Furthermore, we only focus on the activity of truckload transportation. Truckload transportation refers to activities associated with the physical distribution of shipments using trucks. Based on this limited scope, a Golang data structure that aligns with the EDI 214 standard is developed as follows:

```
type EDI214 struct{
    EnvelopDet EnvelopDetail
    TCSSM TCSSMessage
    EnvelopSum EnvelopSummary
}
type TCSSMessage struct{
    THeading TCCSSMHeading
    TDetail TCCSSMDetail
    TSummary TCCSSMSummary
}
type TCCSSMDetail struct{
    L1000 []Loop1000
    L1100 []Loop1100
    L1200 []Loop1200
    L1300 []Loop1300
}
type Loop1100 struct{
    AT7 string
    MS1 string
    MS2 string
    M7 string
    MS104 string
    MS105 string
}
```

The TCSSMessage in EDI214 specifies the transportation carrier shipment status and consists of a header, a footer and the record of the shipment status details (TCC-SSMDetail). TCCSSMDetail has a specific structure and includes four loops labeled Loop1000, Loop1100, Loop1200 and Loop1300. Loop1000 includes information such as business instructions and lading handling requirements. Loop1100 specifies the transportation carrier shipment status. Loop1200 covers the party's basic information (e.g., identification, address). Loop1300 is reserved for the order information details.

In Loop1100 of EDI214, AT7 is a segment reserved for status details, MS1 is for shipment location and MS2 identifies the owner. The timestamp is part of the AT7 segment and adhere to the X6 EDI code which refers to a shipment that is enroute to the delivery location. For example, AT7='"X6*NS***20170320*1125" refers to a shipment with normal status (NS) on March 20_{th} 2017 at 11:25 am. The codes MS104 (longitude) and MS105 (latitude) are used to capture the geolocation of the shipment within the MS1 segment.

5.2 Software Components

The various software components of the proposed HP3D are shown in Fig 5.1 for each of the index server, administrative node and regular node. The external monitor components have not been implemented and remain for future work.

5.2.1 Index Server

The index server includes two tiers: a data tier which consists of a database for storage of peer-related information and an application tier that can read and update the database as well as manage the direction of information flow between the server and the peers. MongoDB, a noSQL database, is used for the database. The application tier uses MGO, a Golang-based driver, to interact with the database. The index server is constantly active in order to receive and process requests from all the nodes. The server invokes the function *serverListen()* shown below. In order to listen for these incoming request on port 9999, the function *serverListen()* accepts a connection when it occurs and makes a call to *receiveMsg()*.

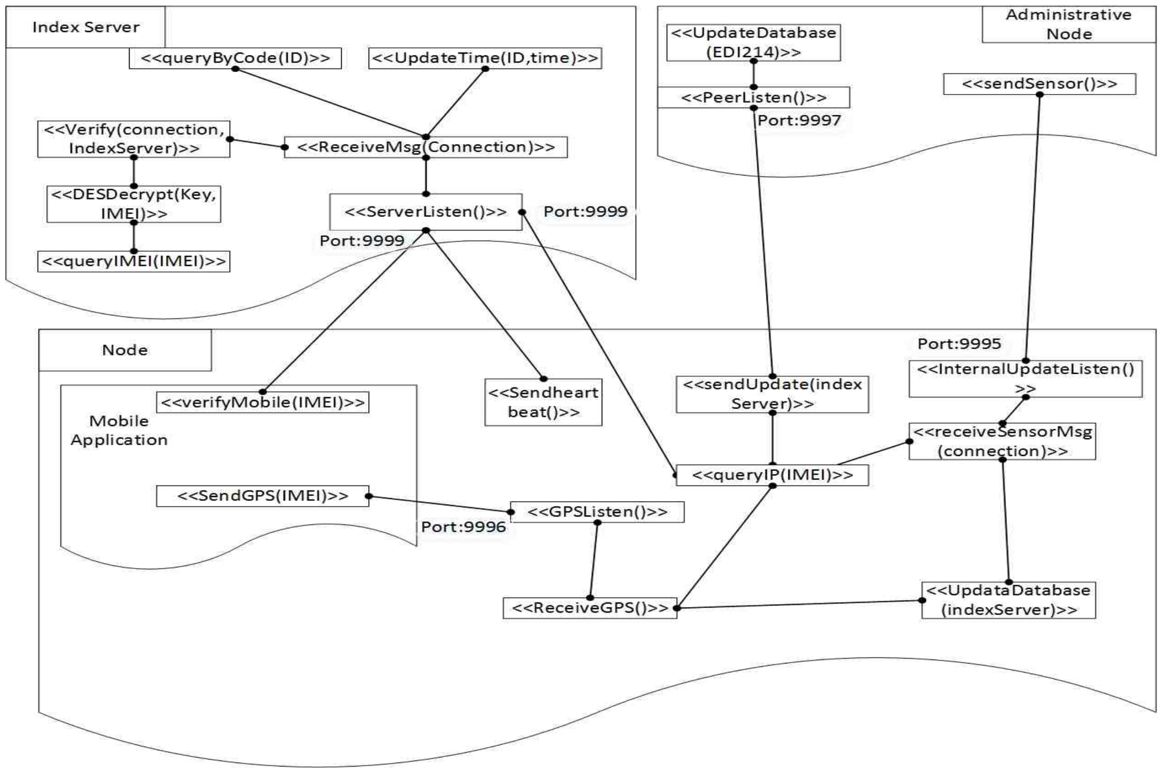

Fig. 5.1. Software Components

The *receiveMsg()* function accepts a connection and extracts the data in the connection using the format of the *indexServer* struct described in Section 5.1. The index server will then check the *Reqtype* field in order to determine the request type and invoke the corresponding function or operation.

- *Reqtype* 0: A heartbeat message which will trigger a local database update using the ID field in the message and the current system time as the timestamp.
- *Reqtype* 1: It triggers a local database search as previously discussed in Chapter 4. The index server will reply to the requesting node with a *indexServer* struct which contains an IP address in the *IPAddr* field. The field will contain 0.0.0.0 if the target node does not exist, 1.1.1.1 if the node is not active and a normal IP address if the node exists and is currently active.

```
Function serverListen()
    ln,err = Listen(tcp, :9999)
    if $err$ != $nil$ then
        print err
    end

    while $True$ do
        c,err = ln.Accept()
        if $err$ != $nil$ then
            print err
            continue
        end
        receiveMsg(c)
    end
    return
```

Algorithm 5.1. Server Listen Funtion

- *Reqtype* 2: This is the mobile verification routine. The *verify()* function shown below is invoked in order to process this request.

The *verify()* function takes two arguments, the *indexServer* struct and a connection. The reason for the connection is that during the verification process the index server and the node being verified need to exchange messages several times. Therefore, the connection needs to be maintained until the verification routine is completed. For testing purpose, the Diffie-Hellman key exchange protocol is used between the index server and the node. As indicated in Section 3.1.4, the Diffie-Hellman protocol is based a publicly known key, in this case P and g. In order to ensure the appropriate security level and meet the requirement of the algorithm, these two parameters are defined as bigInt. In the key exchange phase, the index server generates a random number (*random*) in the range of 1 to *p-1* and calculates $myKey = g^{random} \ mod \ P$. Then based on the key sent by the node, the index server can calculate the secret Key (*secretKey*). This key is used for the encryption/decryption of the IMEI number. In order for the node to get the same secret key (*secretKey*), the index server sends *myKey* back to the node. The node will reply with an encrypted IMEI number rep-

```
Function receiveMsg(connection c)
1   indexServer message = c.getMessage()
2   indexServer result
3   if message.Reqtype ==0 then
4       message.timestamp = getSystemTime()
5       updateTime(message.ID,message.timestamp)
    end
6   if message.Reqtype ==1 then
7       result = queryByCode(message.ID)
8       if result != nil then
9           result.IPAddr = 0.0.0.0
        else
10          if result.timestamp - current time ¿ threshold then
11              result.IPAddr = 1.1.1.1
            end
        end
12      reply with result
    end
13  if message.Reqtype ==2 then
        verify(message, c)
    end
14  return
```

Algorithm 5.2. Server Receive Message Funtion

resented as a byte array in the *IMEI* field of the message. The index server will then use the *secretKey* to decrypt the IMEI number by invoking *DESDecrypt()*. Once, the IMEI number is decrypted, the index server will search its local database. A node ID is returned to the node if the IMEI number is pre-registered in the index server's database.

5.2.2 Peer

Peer applications have a three-tier architecture consisting of a presentation tier, a middle tier and a data tier. The presentation tier consists of a local HTTP server hosting a web application. The HTTP server is responsible for processing the user requests and invoking the appropriate functions based on the request. The presen-

```
Function verify(indexServer message, connection c)
    bigInt random = random(1,P-1)
    bigInt peerKey = bitInt(message.key)
    bigInt myKey = $g^{random}$ mod P
    bigInt secretKey = $peerKey^{random}$ mod P
    message.key = myKey.toByteArray()
    c.send(message)
    message = c.getMessage()
    string IMEI = DESDecrypt(message.IMEI,secretKey)
    bool exist = queryIMEI(IMEI)
    if exist then
        message.ID = generateID()
        c.send(message)
    end
    return
```

Algorithm 5.3. Verification Function Server Side

tation tier is implemented using HTML, CSS, JavaScript and the Go programming language. The middle tier handles three main types processes:

- Query IP address: This is a support process which was described in Section 5.1. The process is executed when a node wants the IP address of its partner nodes. This information is needed, for instance to send the partner nodes updated shipment information. The function *QueryIP()* takes the ID of the target node in string format. Inside the function a new *indexServer* struct called *request* is defined, and the *Reqtype* field is initialized to 1 indicating it is a query IP request. The ID field is initialized to the ID of the target node. The node will then dial the IP address of the index server on port 9999. This will establish a connection with the index server and the node can send the *request* to the index server. The index server will response with an *indexServer* struct which includes an IP address. Finally, the function returns the *IPAdd* to the invoking function. This process is used by two core processes: the GPS update process and the internal sensor update process.

- Internal sensor update process: This is a core process which was discussed in Section 5.2. The node receives an internal sensor update message which is generated by the administrative node. It will then update its local database and broadcast the update message to all related nodes.

- GPS update process: This process is similar to the internal update message. However, instead of receiving an internal sensor update message, the node receives a GPS update from the simulator. The local database of the node is updated and all partner nodes are notified.

```
Function QueryIP(string ID)
1     indexServer request
2     request.Reqtype = 1
3     requst.ID = toObjectID(ID)
4     c = net.Dial(tcp, Index:9999) // Index is the IP address of index server
5     c.send(request)
6     request = c.getMessage()
7     string IPAdd = request.IPAddr
8     return IPAdd
```

Algorithm 5.4. Query IP Address

In order to receive the internal sensor update message from the administrative node, a given node has to listen on port 9995 as shown in Algorithm 5.5. The *receiveSensorMsg* will then be invoked once a new connection is received. The node will extract the incoming message from the connection using the struct *EDI214* as defined in Section 5.1. The node will first update its local database and then send the update to other nodes. A while loop is used to broadcast the message to all nodes that are involved in the same shipment in a sequential manner. The *EDI214* struct contains the information of all nodes in the *L1200* which is an array since there are usually multiple nodes participating in a shipment. Since the administrative node participates in all the shipments that involve the associated company, the information related to the administrative node is also included as an element in *L1200*. The *extractID()* function extracts the node information by accessing each element of the

```
Function InternalUpdateListen()
    ln,err = Listen(tcp, :9995)
    if err != nil then
        print err
    end
    while True do
        c,err = ln.Accept()
        if err != nil then
            print err
            continue
        end
        receiveSensorMsg(c)
    end
    return
Function receiveSensorMsg(connection c)
    EDI214 message = c.getMessage()
    updateDatabase(message)
    while i=0 to length(message.TSSM.L1200) do
        string ID = extractID(message.TSSM.TDetail.L1200[i])
        string IPAddr= QueryIP(ID)
        if IPAddr != 0.0.0.0 & IPAddr != 1.1.1.1 then
            c = net.Dial(tcp, IPAddr:9999)
            c.send(message)
        end
    end
    return
```

Algorithm 5.5. Internal Sensor Update

L1200 array and for each node, the *QueryIP()* function is called in order to obtain the corresponding IP address from the index server. If the returned IP address is valid, the updated message is sent to the target node.

The GPS update function is shown in Algorithm 5.6. It uses port 9996 to receive a GPS update from the GPS emulator. The message sent by the GPS emulator is in string format. It contains the longitude, latitude, TruckID and a timestamp. In order to extract the information separately, few functions are called such as *extractTruckID()*. After the information is extracted, a local database search function *getAllShipment()* is called. Since there may be multiple shipments in one truck, this

```
Function GPSListen()
    ln,err = Listen(tcp, :9996)
    if err != nil then
        print err
    end
    while True do
        c,err = ln.Accept()
        if err != nil then
            print err
            continue
        end
        receiveGPSMsg(c)
    end
    return
Function receiveGPSMsg(connection c)
    string GPSmessage = c.getMessage()
    string TruckID = extractTruckID(GPSmessage)
    string timestamp =extractTime(GPSmessage)
    string long = extractLongitude(GPSmessage)
    string lat = estractLatitude(GPSmessage)
    EDI214 allShipment [] = getAllShipment(TrcukID)
    while i=0 to length(allShipment) do
        allShipment[i].TCSSM.TDetail.L1100[0].AT7 =X6*NS*+timestamp
        allShipment[i].TCSSM.TDetail.L1100[0].MS2 = TruckID
        allShipment[i].TCSSM.TDetail.L1100[0].MS104 = long
        allShipment[i].TCSSM.TDetail.L1100[0].MS105 =lat
        updateDatabase(allShipment[i])
        while j=0 to length(allShipment[i].TSSM.L1200) do
            string ID = allShipment[i].TSSM.L1200[j]
            string IPAddr= QueryIP(ID)
            if IPAddr != 0.0.0.0 & IPAddr != 1.1.1.1 then
                c = net.Dial(tcp, IPAddr:9999)
                c.send(message)
            end
        end

    end
    return
```

Algorithm 5.6. GPS update

function returns an array of all shipments that are being carried by the truck with the specified *TruckID*. For each shipment in the array, GPS update (Algorithm 5.6) is executed. In the *EDI214* struct, *Loop1100* contains the details of the shipment status where code AT7 specifies the timestamp, longitude and latitude. The function will also send an update message to all other nodes in the shipment-centric sub-network by first invoking the *QueryIP()* function to get the IP address of the target nodes and sending them the message through a TCP connection.

A peer cannot only receive messages from both GPS simulator and administrative node, but also receive messages from other peers through port 9999. Algorithm 5.7 shows the receive message function from other peers. This function is used by all peers in the network including the administrative node. Once a new update is received, the *receiveUpdate()* function is called in order to update the corresponding field in the local database.

```
Function PeerListen()
    ln,err = Listen(tcp, :9999)
    if err != nil then
        print err
    end
    while True do
        c,err = ln.Accept()
        if err != nil then
            print err
            continue
        end
        receiveUpdate(c)
    end
    return
Function receiveUpdate(connection c)
    EDI214 message = c.getMessage()
    updateDatabase(message)
    return
```

Algorithm 5.7. Peer Listen and Update Functions

The mobile verification routine is shown in algorithm 5.8. As discussed in the *verify()* function associated with the index server, the verification process is initiated by the peer. The peer generates a random number *random*. Based on the number *random*, a *MobileKey* can be generated by performing g^{random} *mod P* where *P* and *g* are pre-defined public parameters. The server will process the request and reply with a server key, *serverKey*. A secret key is calculated based on *random* and *serverKey*. The peer will then retrieve its IMEI number in a string format and encrypt it using *secretKey* and the DES encryption algorithm. The encrypted IMEI number *verReq* will be sent to the server. An ID is returned by the server if the mobile device is an authorized device.

```
Function verifyMobile()
    indexServer verReq
    verReq.Reqtype = 2
    bigInt random = random(1,P-1)
    bigInt MobileKey = $g^{random}$ mod P
    verReq.key = MobileKey.toByteArray()
    c = net.Dial(tcp, Index:9999) // Index is the IP address of index server
    c.send(verReq)
    verReq = c.getMessage()
    bigInt serverKey = bigInt(verReq.key)
    bigInt secretKey = $serverKey^{random}$ mod P
    string IMEI = getIMEI()
    verReq.IMEI = DESEncrypt(IMEI,secretKey)
    c.send(verReq)
    verReq =c.getMessage()
    ID = verReq.ID
    return
```

Algorithm 5.8. Verification Function Peer Side

5.3 Testing Scenario

The test environment consists of four computers, three of them represent the three parties in a shipment-centric sub-network, namely carrier, supplier and customer. The fourth node represents the index server. An Android based GPS emulator is also used to test the GPS update process. Moreover, the mobile verification routine is tested by using an Android based mobile phone and the above fourth machine as the index server. We assigned static IP addresses and several parameters were predefined including the parameters P and g which are used for the mobile verification scenario.

Scenario 1: Heart beat message. Fig 5.2 shows the data being exchanged during the heartbeat message between the index server and the three nodes. The heart beat message rate was set to 30 seconds. The message being transferred is in the *indexServer* format. For illustration purpose, only the ID field is shown in the graph since this is the only field being used.

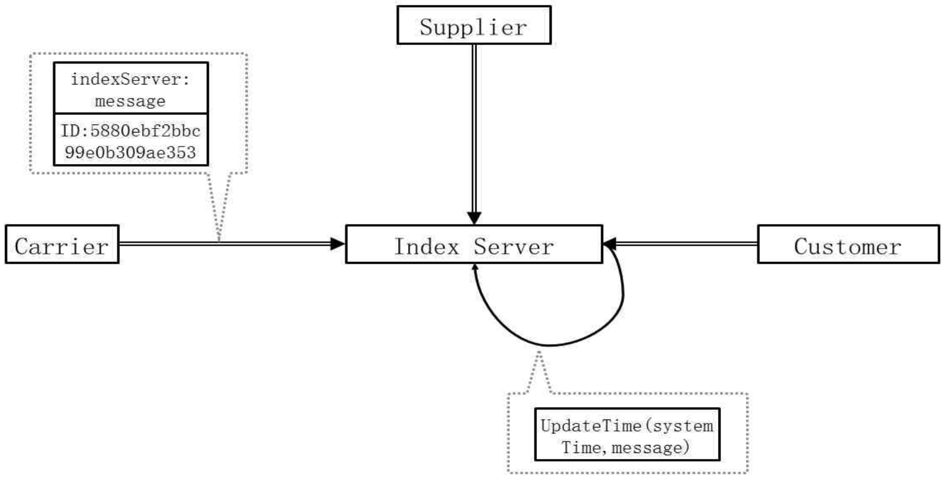

Fig. 5.2. Heartbeat Message Testing Scenario

Scenario 2: GPS update. Fig 5.3 shows the GPS emulator sending a GPS update to the carrier. This message is then forwarded to both customer and supplier. This scenario also shows how the query IP address process is used by the carrier to obtain the IP addresses of the customer and the supplier. The GPS emulator sends a GPS update event to the carrier in string format that is separated by commas. The carrier can extract the update information from the string including, the TruckID, longitude (Long), latitude (Lat) and a timestamp. The longitude and latitude is generated by the GPS sensor in the mobile device in decimal representation. The carrier retrieves all shipments' information. Based on this information, the carrier can query the index server about other trading partners' IP addresses, then send them the updated message in the fields *EDI214* format. Fig 5.3 shows the content of the updated message in the *EDI214* format. As indicated before, *Loop1100* contains the updated information in *AT7*, *MS2*, *MS104* and *MS105*. The customer and the supplier will update their respective databases when they receive the message using the *update()* function.

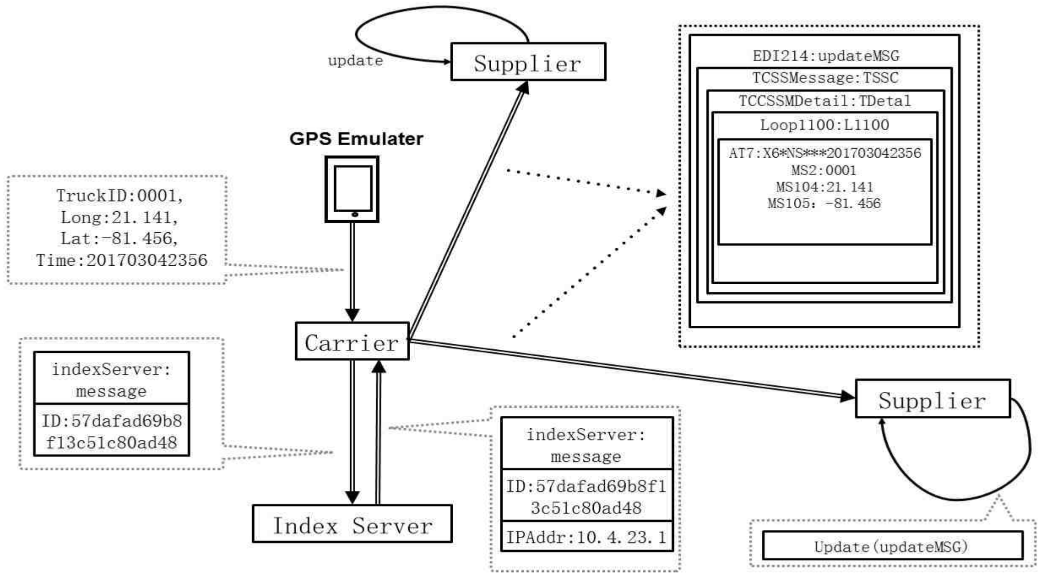

Fig. 5.3. GPS Update Testing Scenario

Fig 5.4 shows the mobile verification routine scenario. The process begins with the mobile device sending an *indexServer* struct labeled *message* to the index server. The *Key* field in *message* contains a number that is generated by the mobile device based on the public parameter P and g. The index server receives the *message* and generates a number Y using the same public parameter and calculates the secret key based on the number received and the random number Y. The index server sends another number back to the mobile device in order to enable the creation of the secret key. The mobile device encrypts its IMEI number and stores in a byte array *message.IMEI* and sends to the index server. The index server decrypts it and searches for the IMEI number in its database. Then sends back an ID to the mobile device if the IMEI number is valid.

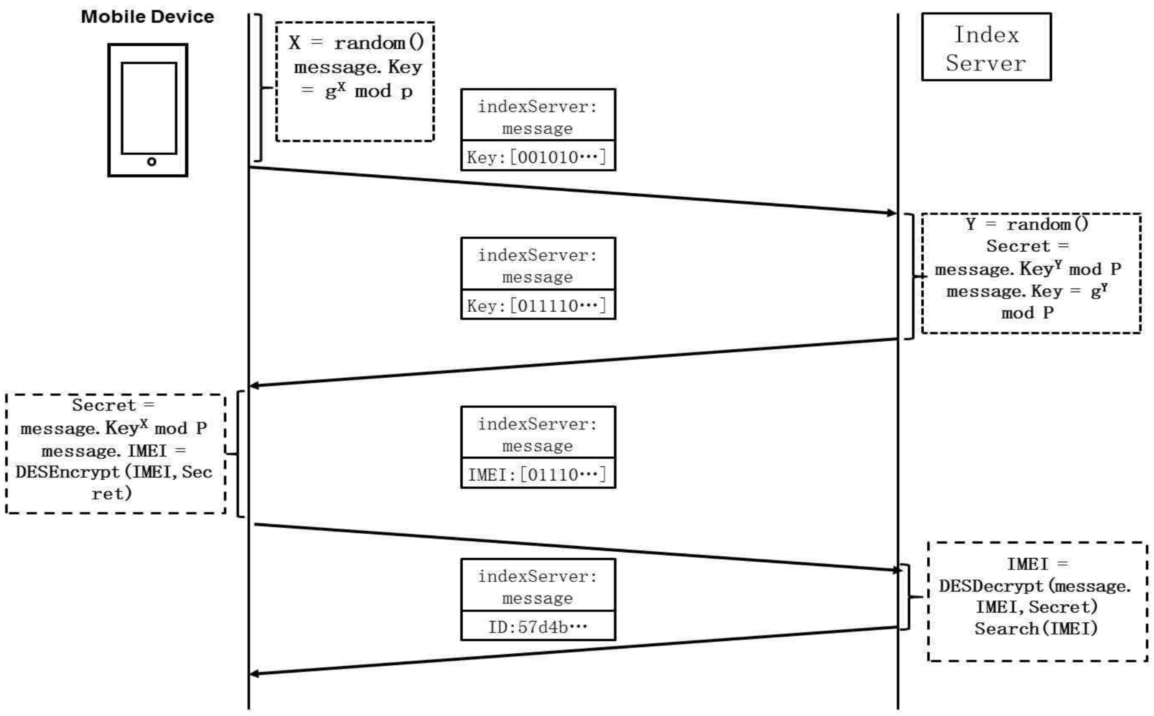

Fig. 5.4. Mobile Device Testing Scenario

6. CONCLUSION

The proposed framework is the foundation for a dynamic, resilient and scalable supply chain digital system that can provide trading partners with total visibility in the distribution phase. We believe that total visibility in the distribution phase is the most difficult to achieve and the one with the highest impact on the efficiency of the end-to-end supply chain. However, we also anticipate additional extensions and improvements that can enhance the maturity of the proposed system and extend its functionalities. For instance, the proposed blockchain model and in particular the semi-public ledger need to be fully implemented and tested. A potential implementation will store the semi-public ledge in the database of the monitors, index server and peers. MongoDB is a good choice for the blockchain data structure since the messages being transmitted between the parties use the JSON format which is compatible with MongoDB.

All partners that are involved in the network must have public/private key pairs. These keys are denoted $\mathrm{Cus_{pub}}$, $\mathrm{Cus_{priv}}$ for the customer. Similarly, the key pairs for the supplier, carrier and external monitor are denoted $\mathrm{Su_{pub}}$, $\mathrm{Su_{priv}}$, $\mathrm{Ca_{pub}}$, $\mathrm{Ca_{priv}}$, $\mathrm{Ex_{pub}}$ and $\mathrm{Ex_{priv}}$, respectively. The external monitor uses $\mathrm{Ex_{priv}}$ to sign each monitoring event it generates before posting it to the semi-public ledger. The other parties can then verify the monitoring events using $\mathrm{Ex_{pub}}$. The key pair for external monitors can be generated when the monitors register with the index server. Furthermore, for better security and message integrity, these key pairs should be updated periodically. Potentially, key pairs may be generated on a daily-basis. The encryption of the messages can be implemented using popular public key encryption/signature mechanism such as RSA [20] or ECDSA [24].

A second enhancement is with respect to the distribution of keys among carrier supplier and customer. In order to exchange custody events, the trading partners have to use key pairs. These key pairs can be generated in two ways. Under the first option, the key pairs are generated when the trading partners join the network based on credentials that are provided by each partner. The key pairs can then be updated periodically in order to maintain the security level of the keys. Every time a shipment is initiated, all parties can share their public keys with the other partners involved in the shipment. These keys are then used to encrypt the message that is shared among the partners. The method is easy to implement since the key pair only needs to be generated once and can be reused for different shipments. Nevertheless, the custody update information will have to be re-encrypt few times when it is sent to different parties, since these parties have different key pairs. For instance if a message is encrypted by Su_{pub}, the message can only be decrypted by Su_{priv}. Therefore, in order to share information among different parties while maintaining data confidentiality, the message needs to be re-encrypt using Su_{pub} ,Cu_{pub} and Ca_{pub} and sent to supplier, customer and carrier, respectively.

The second option takes advantage of the physical contract signing process. A new key pair can be generated every time a new contract is signed. An optical key can be created during this process. Under this option, the customer, carrier and supplier will have the same key pairs for a given shipment. The key will expire when the shipment is completed. The key pair may be stored in the sub-ledger for future validation purposes. Computationally, this option eliminates the need for re-encryption of the custody events as was required for the first option. However, it suffers from the overhead associated with the generation of the key pair. Selecting the appropriate option depends on the computation speed of the underlying encryption method.

A third area for future work is with respect to the mobile application. A verification routine was developed to verify that the mobile device is valid. In addition, the application needs to be able to establish a connection and capture the GPS coordinates of the trucks for the external monitors. Other enhancements to the mobile application include a bar code/QR code scanner in order to automate the scanning of codes associated with the shipment.

In conclusion, this thesis investigates the limitations of current supply chain management systems and proposes a new event-based hybrid peer-to-peer framework that support pseudo real-time transparency in the physical distribution phase of the supply chain. The proposed framework is a departure from the traditional centralized transaction-based supply chain management systems.

A prototype of the proposed framework was implemented. The index server application was built using Golang. It is responsible for the storage and distribution of peer information. The peer application is web-based and was implemented using HTML, Golang, Javascript and MongoDB. In order to improve the security level of the proposed framework, a blockchain model was also introduced. This model consists of a semi-public ledger which is maintained by external monitors and includes information related to the geolocation of trucks. The proposed model also relies on private sub-ledgers, where each sub-ledger is associated with a given shipment and is only accessible to the trading partners involved in the shipment. The main contribution of this thesis is the development of a distributed architecture to enable the sharing of reliable field information among trading partners during the physical distribution phase of the supply chain. The innovative features of this architecture include the use of a peer-to-peer communication model with mechanisms for privacy and reliability that are needed to support commercial applications.

REFERENCES

REFERENCES

[1] J. T. Mentzer, W. DeWitt, J. S. Keebler, S. Min, N. W. Nix, C. D. Smith, and Z. G. Zacharia, “Defining supply chain management,” *Journal of Business logistics*, vol. 22, no. 2, pp. 1–25, 2001.

[2] B. Heaney, “The supply chain visibility: A critical strategy to optimize cost and service,” *Aberdeen Group*, vol. 20, 2013.

[3] K. L. Croxton, S. J. Garcia-Dastugue, D. M. Lambert, and D. S. Rogers, “The supply chain management processes,” *The International Journal of Logistics Management*, vol. 12, no. 2, pp. 13–36, 2001.

[4] J. Holmström, “Business process innovation in the supply chain–a case study of implementing vendor managed inventory,” *European journal of purchasing & Supply Management*, vol. 4, no. 2-3, pp. 127–131, 1998.

[5] A. Bidgood, “Vads interworking: a cloud on the edi horizon,” *The EDI Handbook, Blenheim Online, London*, 1988.

[6] R. Karpinski, “E2open at one,” *InternetWeek, August*, vol. 1, 2001.

[7] F. Andera and D. Derringer, “(” systems, applications, products in data processing”) sap: Implications for computer information systems,” *Journal of Computer Information Systems*, vol. 39, no. 1, pp. 72–75, 1998.

[8] S. Poslad, *Ubiquitous computing: smart devices, environments and interactions.* John Wiley & Sons, 2011.

[9] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008 (accessed February 10, 2017).

[10] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu, “Influences on cooperation in bittorrent communities,” in *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems.* ACM, 2005, pp. 111–115.

[11] G. Fox, “Peer-to-peer networks,” *Computing in Science & Engineering*, vol. 3, no. 3, pp. 75–77, 2001.

[12] S. Guha and N. Daswani, “An experimental study of the skype peer-to-peer voip system,” Cornell University, Tech. Rep., 2005.

[13] I. Filali, F. Bongiovanni, F. Huet, and F. Baude, *A survey of structured P2P systems for RDF data storage and retrieval.* Springer, 2011.

[14] R. Schollmeier, “A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications,” in *Proceedings of the First International Conference on Peer-to-Peer Computing.* IEEE Computer Society, 2001, p. 101.

[15] S. Jiang, L. Guo, and X. Zhang, “Lightflood: an efficient flooding scheme for file search in unstructured peer-to-peer systems,” in *2003 International Conference on Parallel Processing, 2003. Proceedings.* IEEE, 2003, pp. 627–635.

[16] M. Ripeanu, “Peer-to-peer architecture case study: Gnutella network,” in *First International Conference on Peer-to-Peer Computing, 2001. Proceedings.* IEEE, 2001, pp. 99–100.

[17] G. Kreitz and F. Niemela, “Spotify–large scale, low latency, p2p music-on-demand streaming,” in *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P).* IEEE, 2010, pp. 1–10.

[18] M. Gasser, *Building a secure computer system.* Van Nostrand Reinhold Company New York, 1988.

[19] W. Trappe, L. Washington, M. Anshel, and K. D. Boklan, “Introduction to cryptography with coding theory,” *The Mathematical Intelligencer*, vol. 29, no. 3, pp. 66–69, 2007.

[20] M. A. Bishop, *The art and science of computer security.* Addison-Wesley Longman Publishing Co., Inc., 2002.

[21] D. R. Stinson, *Cryptography: theory and practice.* CRC press, 2005.

[22] *BlueKrypt Cryptography Key Length Recommendation*, 2017 (accessed February 10, 2017). [Online]. Available: https://www.keylength.com/

[23] S. William, *Computer Security: Principles And Practice.* Pearson Education India, 2008.

[24] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography.* Springer Science & Business Media, 2006.

[25] R. Sarathy, “Security and the global supply chain,” *Transportation journal*, pp. 28–51, 2006.

[26] M. J. Atallah, H. G. Elmongui, V. Deshpande, and L. B. Schwarz, “Secure supply-chain protocols,” in *IEEE International Conference on E-Commerce, 2003. CEC 2003.* IEEE, 2003, pp. 293–302.

[27] F. Kerschbaum and A. Sorniotti, “Rfid-based supply chain partner authentication and key agreement,” in *Proceedings of the second ACM conference on Wireless network security.* ACM, 2009, pp. 41–50.

[28] B. King and X. Zhang, “Securing the pharmaceutical supply chain using rfid,” in *MUE’07. International Conference on Multimedia and Ubiquitous Engineering, 2007.* IEEE, 2007, pp. 23–28.

[29] G. Kapoor, W. Zhou, and S. Piramuthu, “Rfid and information security in supply chains,” in *The 4th International Conference on Mobile Ad-hoc and Sensor Networks, 2008. MSN 2008.* IEEE, 2008, pp. 59–62.

[30] M. Swan, *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[31] *Blockfreight*, 2016 (accessed February 10, 2017), https://www.cryptocoinsnews.com/blockfreight-taps-blockchain-technology-disrupt-global-shipping.

[32] R. Pike, "The go programming language," *Talk given at Google's Tech Talks*, 2009 (accessed February 10, 2017).

[33] K. Chodorow, *MongoDB: the definitive guide.* " O'Reilly Media, Inc.", 2013.

[34] D. A. Johnson, B. J. Allen, and M. R. Crum, "The state of edi usage in the motor carrier industry," *Journal of Business Logistics*, vol. 13, no. 2, p. 43, 1992.