Computers & Industrial Engineering 136 (2019) 149–159

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Smart contract-based approach for efficient shipment management

Haya Hasan[a], Esra AlHadhrami[a], Alia AlDhaheri[a], Khaled Salah[a], Raja Jayaraman[b,*]

[a] Department of Electrical & Computer Engineering, Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates
[b] Department of Industrial & Systems Engineering, Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates

ARTICLE INFO

Keywords:
Supply chain management
Smart contracts
Ethereum
Blockchain
IoT
Internet of things

ABSTRACT

Efficient tracking of shipments is critical in managing global trade and logistics activities. The volume of global container movement combined with information opaqueness and process complexity necessitates implementing a robust technology solution with real time tracking capabilities. Blockchain is an emerging technology that offers the necessary platform to track and manage shipment movements in the supply chain using a peer-to-peer, secured, distributed ledger, and without intermediaries or trusted third parties. In this paper, we propose a blockchain-based solution for efficient supply chain management involving items shipped via smart containers. Our proposed solution utilizes the features of smart contracts in Ethereum blockchain to govern and manage interactions between the sender and receiver. Shipped items are included in smart containers equipped with Internet of Things (IoT) sensors that can be used to track and monitor predefined shipping conditions related to temperature, geographical location, humidity, pressure, light exposure, sudden fall, broken seal, etc. Ethereum smart contracts are used to manage shipment conditions, automate payments, legitimize receiver and also issue a refund in case of violations to predefined conditions. In the paper, we present and discuss key aspects related to architectural design, entity relations, interactions among participants, information flow, implementation and testing of the overall system functionality with a potential business case applied to vaccine supply chain. The Smart contracts were implemented in Solidity language and tested using Remix IDE environment, the code has been made publicly available for academic, research and practice community.

1. Introduction

Supply chain management of container transportation is important since over 90% of the global trade is based on containerized shipment movement. Levinson (2006) describes a comprehensive history of how containerization has transformed the transportation industry. Offshoring due to global production and distribution operations currently involves multiple parties submitting manual, paper based approvals and obtain necessary clearance before transporting goods. The delays and inefficiencies in the current shipment management process can range from several days to weeks. This presents an important opportunity and need to adopt technology and automation solution that enables timely supply chain decisions.

Since its inception in year 2008, Blockchain technology has continued to gain significant attention by industry, academia, and government. The disruptive nature of the technology with immense application possibilities in transaction oriented services such as financial, insurance, logistics, healthcare and makes it quite popular. Bitcoin is one of the first and widely known applications of blockchain technology. Blockchain provides a distributed peer-to-peer network with shared ledger among all vested parties and participants. No centralized authority or intermediary party is required to provide management or governance, or to verify or process the transactions. Transactions are validated by miner nodes that aggregate all transactions into blocks and chain them into the shared ledger. Miner nodes follow a consensus algorithm such as proof-of work to validate and agree on the integrity and accuracy of the transactions and blocks. This creates a secure, synchronized and shared timestamped records that cannot be altered. We direct the attention of readers to Toyoda, Mathiopoulos, Sasase, and Ohtsuki (2017) for technical details on how blockchain technology has been used to authenticate products in the supply chain. Zhao, Fan, and Yan (2016) for applications to financial products and Yue, Wang, Jin, Li, and Jiang (2016) for healthcare application focused on privacy risk control.

Buterin (2014) discussed the ability to program the Ethereum blockchain and another competing product Hyperledger developed by IBM Blockchain. Unlike bitcoin blockchain, both Ethereum and Hyperledger enable users to build programmable logic and decentralized

* Corresponding author.
E-mail addresses: haya.hasan@ku.ac.ae (H. Hasan), esra.alhadhrami@etic.ac.ae (E. AlHadhrami), alia.dhaheri@ku.ac.ae (A. AlDhaheri), khaled.salah@ku.ac.ae (K. Salah), raja.jayaraman@ku.ac.ae (R. Jayaraman).

https://doi.org/10.1016/j.cie.2019.07.022

Available online 10 July 2019

applications (DApps) using smart contracts. Smart contracts have the ability to provide solutions to govern and manage business transactions and agreements in multi-party settings. A Smart contract gets converted to bytecode which is readable by the 'Ethereum Virtual Machines' (EVMs). The EVM takes an arbitrary bytecode and produces a guaranteed result that is agreed on by all miner nodes using a consensus algorithm. Christidis and Devetsikiotis (2016) present potential application of blockchain and smart contracts for IoT. A smart contract translates the contractual clauses into a code of events and functions. Therefore, once a function is called or an event is triggered the contract self-executes and it can verify the enforcement of predefined terms of the contract while eliminating the need for trusted intermediaries between the transacting parties.

Efficiency of supply chain transactions can be improved considerably from the unique features and architecture of blockchain. The ultimate aim of supply chain management is to have visibility, provenance, and tracking capabilities. Typically, the shipment process of containers involves many participants and with each participant having their own records and log. These logs are updated based on information given by the other participants along the chain. For any adequate visibility and traceability, the tracking process must have a unified, trusted, tamper proof shared log that is globally accessible by all stakeholders. Huh, Cho, and Kim (2017) emphasize that any change or violation that occurs to the condition of shipment should be recorded and communicated to all involved parties, especially, when shipment contains, kits or packages containing temperature-sensitive items such as donor organs, vaccines, blood samples, milk, etc. Supply chain management of perishable, temperature sensitive items are commonly referred as cold chain management. Fig. 1 illustrates an example of cold chain management in healthcare with IoT devices used in smart containers to monitor and trigger notifications with changes in temperature. Other key criteria that can be monitored using IoT enabled devices during shipment transportation include geographical location, humidity, light exposure, sudden fall, broken seal are related criteria. Monitoring geographical location also enables geo-fencing to detect when a shipment has deviated from its anticipated route, been delayed; or if a kit is removed from a storage location. This is particularly useful for high-value products or manage controlled substances.

Motivated by the critical need for a pragmatic supply chain solution, and the widespread adoption of blockchain and IoT technologies in various fields, we propose a blockchain-based supply chain management solution for IoT-enabled smart containers. IoT-enabled containers can facilitate efficient supply chain management by sending accurate sensory data to cloud storage and at the same time trigger notifications to be recorded on the blockchain ledger when violations occur. Smart contracts were used to void shipment when violations occur, automate the payment and issue a refund. The implementation was carried out in a decentralized manner without any intermediaries.

The key contributions of this paper can be summarized as follows:

- Present a framework and solution using Ethereum blockchain via smart contracts for shipment supply chain management. The proposed solution eliminates the need of a trusted third party or intermediary.
- Demonstrate how IoT and blockchain technologies can be jointly utilized to ensure reliable shipment tracking, with no violations.
- Develop and implement a solution for single echelon supply chain transaction between a buyer and seller. We describe the overall system, architectural design, entity relations, and interactions among participating parties.
- Highlight key aspects and details that are generic enough to be applied to mutli-echelon and multiparty settings.

The rest of this paper is organized as follows. Section 2 presents the background literature. Section 3 presents our proposed solution for shipment supply chain management. Section 4 describes a potential business application of the proposed solution to vaccine supply chain. Section 5 describes the implementation aspects and testing of the smart contract and Section 6 presents conclusions and future work.

2. Related literature

We discuss relevant literature on supply chain and cold chain management solutions for shipment tracking using RFID and recent applications including blockchain technology. We emphasize related work on techniques used to improve the reliability, security and data management in supply chains.

2.1. Supply chain management using RFID

RFID technology has been effectively used for product safety, visibility and traceability across several industries. Being a mature technology RFID solution enables systemic improvements across several supply chain & logistics processes. Musa, Gunasekaran, and Yusuf (2014) present an extensive review on supply chain product visibility detailing various standards, systems and architectures. Lee and Park (2008) proposed a dynamic tracing task model for traceability function using RFID technology in supply chains using existing enterprise applications. Angeles (2005) assert "RFID can enable "process freedoms" and real-time visibility into supply chains" present several case studies and implementation guidelines. Lin (2009) presents an integrated framework for the development of RFID technology. Toyoda et al. (2017) deployed RFID technology to track initial product movement in supply chain. RFID-enabled supply chain makes use of the EPC (Electronic Product Code) to uniquely identify an item. The EPC is stored in the RFID tag and each party in the supply chain adds more information to the tag that acts as a piece of evidence that it passed through the right intended party. This method is used to track the products and find counterfeits by checking the tags and identifying any inconsistencies.

Ko, Kwak, Choi, and Song (2015) designed a system compatible with any kind of IoT device to monitor and track the temperature ranges needed during the transportation process. The system proposed

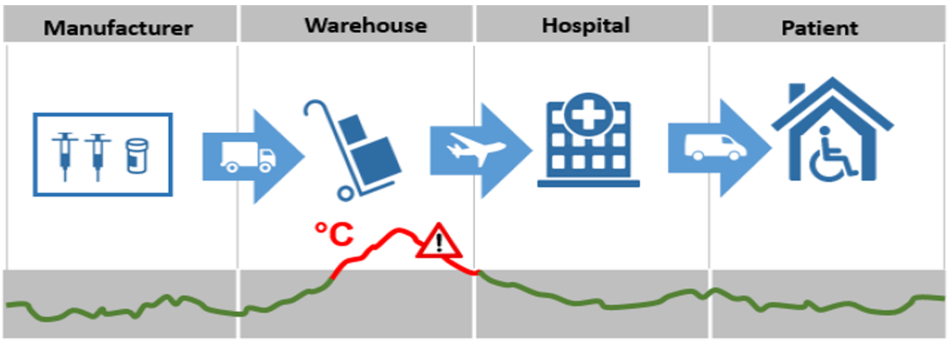


Fig. 1. An example of Cold supply chain tracking in healthcare.

is connected to the cloud platform Hadoop to allow the processing of large distributed information. Although, the system looks promising, the authors do not provide a functional prototype or even a simulation to verify the design criteria. Furthermore, Li and Chen (2011) proposed a similar system using RFID technology for pharmaceutical applications. They discuss the design architecture and a comparison between the use of RFID and infrared technology (IR). Their comparison yields to the conclusion that RFID is more sensitive to temperature changes and unlike IR it does not have limited coverage of all the 3D space. Similar to the work proposed in Ko et al. (2015), they do not provide any verification results of the proposed system. Furthermore, Chou et al. (2013) present an Intelligent Insulating Shipping Container (IISC) that can be used in cold chain logistics management. The container communicates its temperature with the deliverer through Bluetooth Smart and their architecture uses a low power circuit design, hence their proposed solution consumes significantly less power. However, security and privacy of communicated data were not part of their system design. Unlike Chou et al. (2013), Urien and Piramuthu (2013) incorporate the use of the internet smart card along with RFID tags to secure the communication between the tags and the rest of the system. Their application focusses on avoiding degradation and contamination of food products while transferring them from the farm to the fork. They create an authentication system and use a registration server. Although creating an authentication system improves security, it creates an overhead on the overall system's performance.

Although RFID technology has been successfully adopted for supply chain management across many industries, its purpose to track product movement transcending multiple echelons is limited. Thus end to end product tracking requires adoption of technology that enables multiparty authentication and information sharing in real time.

2.2. *Supply chain and cold chain management using blockchain*

Blockchain is an emerging technology that can be leveraged to solve a variety of supply chain management issues (Christidis & Devetsikiotis, 2016; Kim & Laskowski, 2018). Due to blockchain's intrinsic powerful features such as highly secure, trusted, immutable, shared ledger, and globally accessible. These features are important to supply chain decision makers and provides end-to-end tracking capability during the transportation process, offers transparency, and operations in a decentralized manner without the involvement of intermediaries (Zhao et al., 2016). Jayaraman, Salah, and King (2019) highlight improvement opportunities in healthcare supply chains using IoT and blockchain. Banerjee (2018) present use cases and a high level overview of how enterprise resources planning systems and blockchain technology can provide transparency in supply chain operations.

Toyoda et al. (2017) proposed a novel Blockchain-based product ownership management system (POMS) to detect counterfeits in the after-sale supply chain. Their findings suggest that the cost is less than US$1 if the number of ownership transfers in the system does not exceed six. A proof of concept of the system was built using Ethereum, where the proposed POMS utilizes the 'proof of possession of balance' in bitcoins into the 'proof of possession of products'. POMS relies on an RFID system for product transfers between manufacturer and until first retail sale. They do not consider monitoring and tracking the system during the entire transportation process.

Cold chain management deals with efficiently managing, storing, handling and transporting temperature sensitive products including perishables from the point of manufacture till final point of use. Cold supply chain management is heavily used in pharmaceutical, food and related industries. Literature on cold supply chain management using blockchain is scant but has gained significant attention among research and practice community. Mackey and Nayyar (2017) discuss several existing and emerging technologies to manage counterfeits in pharmaceutical supply chain. They assert the literature lacks extensive research on this subject, except for some governmental and industrial initiatives on the use of blockchain to provide solutions for the pharmaceuticals industry such as Blockverify (Mackey & Nayyar, 2017; Toyoda et al., 2017), iSolve (Mackey & Nayyar, 2017) and Parexel (Zobel, 2016). For instance, Tian (2016) attempt to address food safety issue in China by proposing a system that integrates RFID with blockchain to trace the food transported using trusted information. They showcase that centralized systems are monopolistic and therefore, suffer from trust issues. However, blockchain based systems are of a high cost with a low rate of transactions which is restricted to ~7 transactions per second. The conceptual system proposed by Tian (2016) can be extended if a simulation or a use case scenario were demonstrated. Furthermore, Tian (2017) present a solution to centralized solution of supply chain management using the blockchain technology and addressing scalability issue. The authors incorporate HACCP (Hazard Analysis and Critical Control Points) along with blockchain and IoT devices. Their work compares distributed databases, blockchain and BigChainDB which effectively is a scalable blockchain. The comparison is based on: latency, throughput, capacity and immutability factors. They concluded that BigChainDB combines the advantages of both blockchain and distributed databases. Thus, their proposed solution employs BigChainDB with a complete scenario from the supplier to the retailers and consumers. However, their approach presents security vulnerabilities due to the use of third party for certifications and audits.

Bocek, Rodrigues, Strasser, & Stiller, 2017 present a use case for pharmaceutical supply chain to effectively monitor the distribution of medical products in compliance with European Union (EU) regulations. Their work uses IoT sensors along with the blockchain technology and smart contracts to ensure the EU regulations are in compliance during the transport of the medical products was implemented in year 2016 lead to the formation of the company 'Modum.io AG. Temperature is monitored and recorded during the delivery of the product from the manufacturer to the retailers. Their approach depends on Ethereum nodes to execute and validate smart contracts. Every shipment has a new smart contract that is written with Solidity and a database server to store the temperature data. The client would use an Android based application to associate the tracking number of the package communicated through Bluetooth LE with the device MAC address stored in the server. The proposed system should be improved and tested for more number of distributors and shipments. The authors can improve by adding an offline mode feature which would result in a more refined architecture of the system. It would have been a more complete system if changes in temperature were associated with alerts or other types of triggers that would show interference is needed in case any additional criteria for a certain shipment have changed along the way. Therefore, controls are needed not only to monitor the temperature but also to handle exceptions ensuring all shipments reach successfully per the prespecified conditions. Additionally, the authors failed to address the issue of counterfeits or after sales supply chain management. Bocek et al. (2017) only consider product tracking at pre-specified the required temperature during the entire transportation process.

Blockchain offers potential solution for end to end shipment management across multiparty settings. Our solution and implementation highlights important features of this emerging technology using smart contracts.

3. Proposed solution for shipment management

3.1. *System overview*

Our proposed solution utilizes sensor information from IoT-enabled shipments, combined with Ethereum smart contracts. The smart contracts play an essential role in triggering notifications and enabling the participating entities to continuously monitor, track and receive alerts if any violations occur. Therefore, ensuring the integrity of the items within the shipment. Fig. 2 illustrates an overview of the proposed

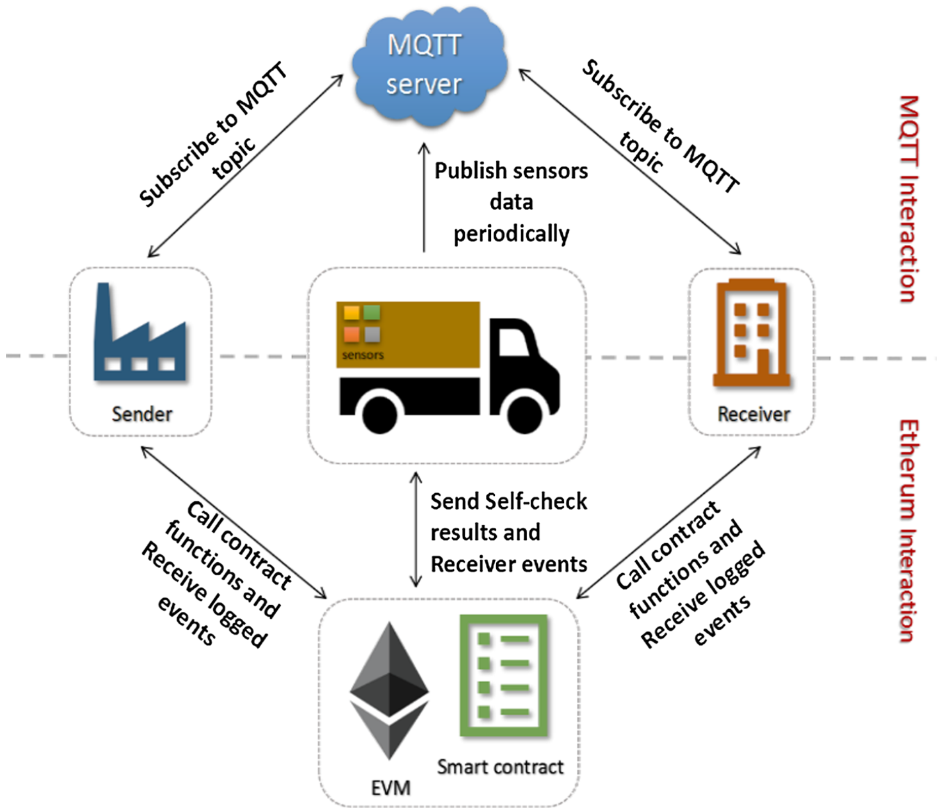

MQTT server
Subscribe to MQTT topic
Publish sensors data periodically
Subscribe to MQTT topic
MQTT Interaction
sensors
Sender
Receiver
Call contract functions and Receive logged events
Send Self-check results and Receiver events
Call contract functions and Receive logged events
Ethereum Interaction
EVM
Smart contract

Fig. 2. General system overview.

system. As shown in the figure, the main components are: sender and receiver of the shipment, IoT-enabled container, Ethereum blockchain that has the EVM executing the smart contract and the Message Queue Telemetry Transport (MQTT) server hosted in the cloud to aggregate, store, and publish all sensor data generated from the IoT sensors installed within the shipment.

3.2. System design

We discuss key design aspects of our proposed solution. We consider four main participants: the sender, the receiver, IoT device installed within the shipment container, each of the participant has an EA (Ethereum Addresses) and the smart contract. In Ethereum, unique EA and key pairs are generated instantly with no centralized entity for managing key distributions (Khan & Salah, 2018). The smart contract is created by the sender with conditions of the sale, adherence to criteria such as temperature, route, payment terms, etc.

3.2.1. Sender and receiver

The typical supply chain model for shipping container shipment can be considered as single echelon with a single sender and single receiver, or could be multi-echelon with many senders and receivers. We focus in this paper on single echelon shipments. However, the presented solution, design, system principles are flexible and can be extended to include multi-echelon shipments whereby the receiver becomes subsequent sender, in which case the IoT device within the container has to be reset at each echelon.

A smart contract that joins the supply chain entities together is created and initiated by the sender after the IoT-enabled container performs the self-check to ensure that the shipment is in agreement with pre-specified shipping conditions. The smart contract governs automatically the rules whereby the receiver has to deposit the payment prior to shipping, and if the shipment reaches the final destination with no violations, then the payment would be issued to the sender. On the other hand, if violations to shipping criteria occur, the shipment is aborted, and a refund is issued to the receiver. In addition, with the use of smart contracts, we can ensure that the shipment reached the legitimate receiver. This is accomplished by having the sender provide a `keccak256` hash of a passphrase that gets stored in the smart contract prior to shipment dispatch. The carrier will not release the container to the receiver unless the receiver provides the correct passphrase which gets sent to the smart contract and compares it to the stored `keccak256 hash`. If the hashes match, then the carrier will deliver the shipment to the receiver. The receiver has 48 h to provide the correct passphrase. Failure to do so will result in 50% refund to the receiver.

3.2.2. IoT-enabled container

The IoT-enabled smart container is equipped with IoT sensors to continuously monitor and track the shipment from the sender to the receiver. The IoT sensors includes temperature sensor to monitor the temperature, Global Positioning System (GPS) receiver to trace the location of the shipment, pressure sensor to detect pressure differences that can indicate opening or closing the container, and an accelerator to detect jerks in case of sudden fall or drop. All the sensors are connected to a Raspberry Pi 3 Model B- the onboard IoT processing hardware placed inside the container. High-end Arduino boards such as Arduino Mega with networking capabilities can be used. 4G or 5G connectivity will be needed for periodically transmitting the data to the MQTT server, and for communicating with the Ethereum blockchain network.

Automatic push alerts and notifications are sent to the sender and receiver when certain conditions are not met or violations occur during transportation. Violations are determined by a rule-based engine at the Raspberry Pi. Certain rules should be specified for each sensor reading, failing to meet these rules is considered as a violation. The rules and their thresholds can be changed according to the nature of the shipment with agreement between the sender and receiver. Sensor readings are monitored by the IoT-enabled container and whenever there is are variations, the shipment container initiates call to certain functions

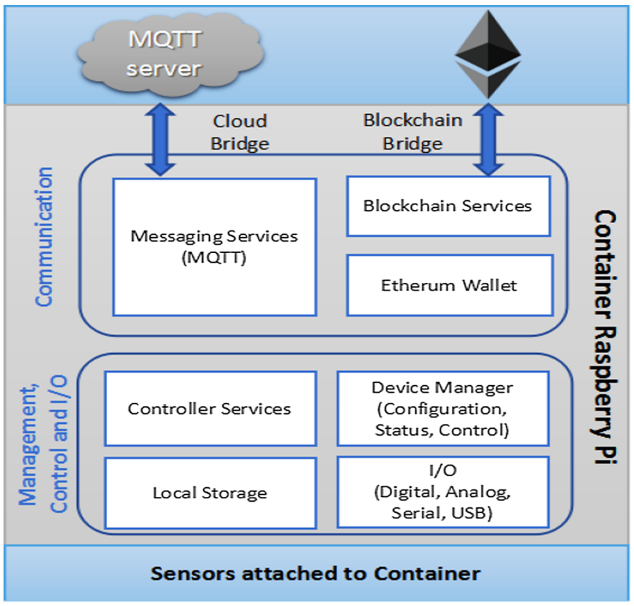

MQTT server
Cloud Bridge
Blockchain Bridge
Communication
Messaging Services (MQTT)
Blockchain Services
Etherum Wallet
Container Raspberry Pi
Management, Control and I/O
Controller Services
Device Manager (Configuration, Status, Control)
Local Storage
I/O (Digital, Analog, Serial, USB)
Sensors attached to Container

Fig. 3. Interfaces and interactions of IoT-enabled container with MQTT server and Ethereum blockchain network.

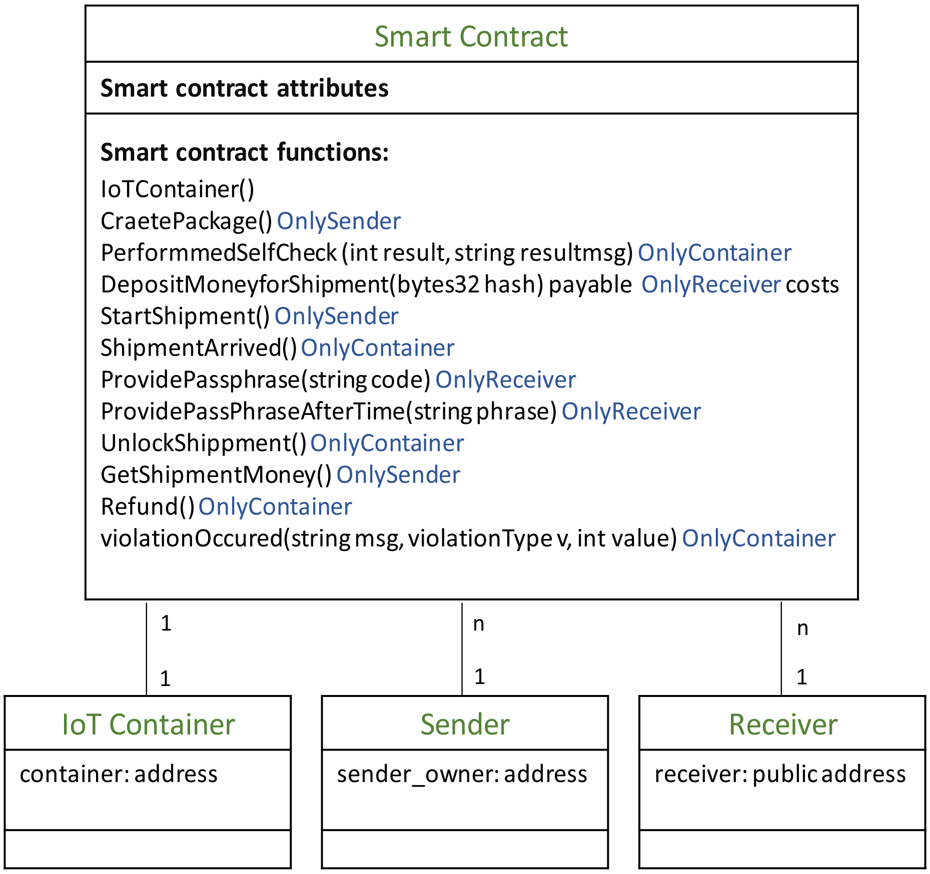
within the smart contract and then events for these violations are broadcasted to all stakeholders including the sender and receiver.

Apart from the smart contracts, periodic readings of sensor data will be sent to a cloud-based MQTT server. The Raspberry Pi of the container is the publisher to the container's state that contain the readings of the temperature, location, motion, and state (open/close). The MQTT structure gives the option to the parties such as the sender and receiver to subscribe to all or some of the published data or topics. As described in Bahga and Madisetti (2016), Fig. 3 shows key processing, interfaces, and communication components among IoT-enabled container, sensors, Raspberry Pi, cloud-hosted MQTT server, and the smart contract through the Ethereum blockchain network.

3.2.3. Smart contract

For each IoT-enabled shipment container, a separate smart contract is created by the sender. The smart contract contains the following key elements:

- **Variables**. The variables used in the smart contract contain the values and states that reflect certain conditions and parameters. Variables can be made public or private. Some of the important variables in the proposed smart contract is the package state and the type of violation. Also, the addresses of the participating entities are saved as variables.
- **Methods**. Methods are basically code functions. The methods in the smart contract are directly related to the functionality of the contract. In our case, the methods will trigger alert notification for violation and package related events as well as authenticate the legitimacy of receiver using the `keccak256 hash`. Methods can be publicly accessible and modifiable by all, or can be restricted to certain participants.
- **Modifiers.** Modifiers are logic tests that are used as conditions or requirements before executing certain methods/functions. For example, a modifier can be used to require the shipment cost to be of a certain value before accepting the deposited money by the receiver.
- **Events.** Events can be used to log important information to be recorded on the ledger, or to broadcast this information to all nodes and participants. For example, events are used in our system to record and propagate violations if received from the IoT device within



```
IoTContainer()
CraetePackage() OnlySender
PerformmedSelfCheck (int result, string resultmsg) OnlyContainer
DepositMoneyforShipment(bytes32 hash) payable OnlyReceiver costs
StartShipment() OnlySender
ShipmentArrived() OnlyContainer
ProvidePassphrase(string code) OnlyReceiver
ProvidePassPhraseAfterTime(string phrase) OnlyReceiver
UnlockShippment() OnlyContainer
GetShipmentMoney() OnlySender
Refund() OnlyContainer
violationOccured(string msg, violationType v, int value) OnlyContainer
```

Fig. 4. Entity relationship diagram (The full code of the smart contract is available at https://github.com/smartcontract694/project.git).

the shipment container. Therefore, if the temperature or geo-location values are beyond predefined ranges, or the IoT-enabled container has opened unexpectedly, or a sudden fall had occurred, the Raspberry Pi will capture such violations and call the suitable method within the smart contract, and subsequently the smart contract will use events to record and broadcast such violations.

Fig. 4 shows the entity-relationship diagram that illustrates the smart contract attributes, and functions and the relationship among participants including the sender, receiver and IoT-enabled container, and the smart contact. As shown in Fig. 4, each shipment container can have one smart contract while each sender and receiver can have multiple smart contracts associated with different shipments.

4. Potential business application to vaccine supply chain

Potentially, there are many real-world use cases that can leverage our proposed solution such as shipment traceability in agriculture and food supply chain, as well as package tracking and delivery in variety of industries including automotive, retail, and healthcare. In this section, we discuss how our solution can be applied to vaccine supply chain. Other use cases can follow the same pattern, but the actors and their interactions can be adapted to the context.

Vaccine supply chain shares significant waste and fraud in addition to challenges related to global procurement, storage and distribution, payment processing and track and trace. Vaccines are unique healthcare products that are required to be handled, stored and administered to prevent substantial deviations of storage temperature or other conditions that may render the product ineffective for use. It is worth noting that over 50–60% of vaccines lose their potential due to supply chain failures and 80% of vaccine costs are supply chain related (StaTwig, 2019). The use case in this application has the following four actors: manufacturer (sender), hospitals (receiver), IoT device enabled shipment container and the smart contract. The delivery and handling of the shipment is done by shipment carrier.

In the first step, the manufacturer (sender) creates a smart contract and invokes an shipment order via `CreatePackage()`for supplying specific quantity of vaccines to the sender with agreed conditions of the shipment such as the cost and quantity, temperature, validation conditions during transit and at receiving. The global nature of the vaccine shipments can have validation and inspection at various transit points. So the opening of the container at various locations can be precisely recorded, and thereby preventing any unauthorized access to the shipment. The shipment is sent via an IoT device enabled container that invokes `PerformmedSelfCheck()` to ensure the shipment conditions are in accordance with the agreed conditions. If in violation, the shipment is aborted. When a successfulnotification of self-check is received, the hospital (receiver) deposits the agreed transaction amount invoking the function `DepositMoneyforShipment()`.The manufacturer (sender) invokes the `startshipment()` function notifying the shipment carrier to transport the container to the hospital (receiver). Throughout the transit of the container, the IoT device relays periodic alert notification to both the manufacturer (sender) and hospital (receiver) about the geographic location, violation to the temperature or other conditions such as unexpected container opening or jolts via `violationoccured()` function. If any agreed parameters are violated the shipment is aborted and sent back to the sender. Note that the manufacturer (sender) adds the `keccak256` hash of a passphrase at the time of `CreatePackage()` to the smart contract and used for validating the authenticity when the hospital(receiver) provides the correct `keccak256`. Upon receiving notification regarding the arrival of the shipment the hospital (receiver) has to provide the correct passphrase within 48 h, else 50% of the shipment amount would be refunded to the receiver and the shipment is aborted. The receiver authentication is validated by matching the passphrase and is critical to ensure receiver authentication and traceability in the vaccine supply chain. The implication of our proposed approach can potentially lead to reduction in wastes, effective traceability, efficient funds transfer and receiver authentication.

5. Implementation and testing

The proposed smart contract was implemented and tested using Remix IDE http://remix.ethereum.org/. In this section, we provide the main implementation details and focus primarily on testing the correct interaction and functionality among system participants with the Ethereum smart contract. The Remix IDE offers rich features that make it possible to test and debug smart contracts prior to deploying them. Remix IDE offers multiple Ethereum wallets, allowing us to simulate real-life scenarios. Additionally, a built-in debugger comes with Remix and allows the investigation of various transactions to ensure the correct behavior of the contract in various conditions.

5.1. Implementation details

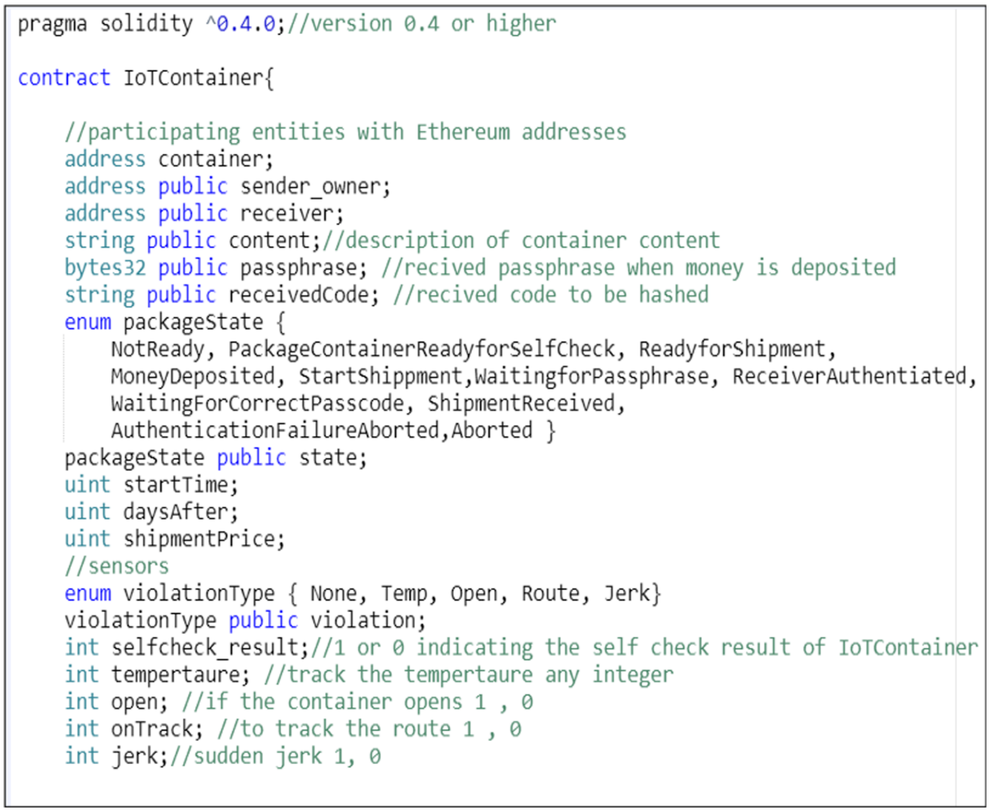
The code was written in Solidity using the web browser based Remix IDE. There are three entities participating in the contract, sender, receiver and IoT-enabled container. Each of the entity has an Ethereum address and can participate by calling functions within the smart contract at certain times. An entity is not allowed to make any function call, this is done through the use of modifiers. The modifiers restrict the functions to be called by only specific stakeholder in the supply chain. For example, `CreatePackage()` can only be called by the sender, `PerformmedSelfCheck()` can only called by the container carrier and the `DepositMoneyforShipment()` can only be called by the receiver.

Fig. 5 shows the attributes used in the contract code. In order to only allow the execution of the functions in proper order based on the shipments logical flow, the state of the package is maintained throughout the code and within every function call. This is done using the variable state, which is an *enum* named *packageState*. The first line of code in every function is made to check the current state of the package and based on the execution, any state changes and gets updated. The new state is propagated as an event to all the entities involved so that the next caller can act accordingly.

Violations related to the shipment container if detected after the package has been transported, the container will call the function `violationOccurred(string, violationType, int)` which would take as parameters a string message for the triggered event, an *enum* for the violation type and an integer for the violation value. The function changes the state of the package to Aborted and issues a refund to the receiver. Fig. 6 shows the code that is called by the container when a violation occurs.

As shown, the smart contract was coded as it eliminates the need for a trusted third party between the sender and the receiver. The contract gets created by the sender. Once the smart contract is created, the item is included in the shipment container and gets sealed, and subsequently the Raspberry Pi board within the shipment container performs a self-check, calls the function `PerformedSelfCheck()` within the smart contract to proceed with the shipment. The shipment will be aborted if the self-check fails.

This ensures that the container provided from a source is trusted by both the sender and the receiver, works as expected with no flaws before the package is shipped. If the self-check's result is successful, then an event is triggered with the successful result and the receiver will now have to deposit the money to the contract and a `keccak256 hash` is provided. The hash is provided to check the authenticity of the receiver when the shipment arrives at the destination. Upon arrival, the receiver provides the correct passphrase to the carrier who will send it to the smart contract. The smart contract hashes the provided passphrase and compares it to the originally submitted hash, if they match then it is the right passphrase and the shipment container unlocks. The receiver has

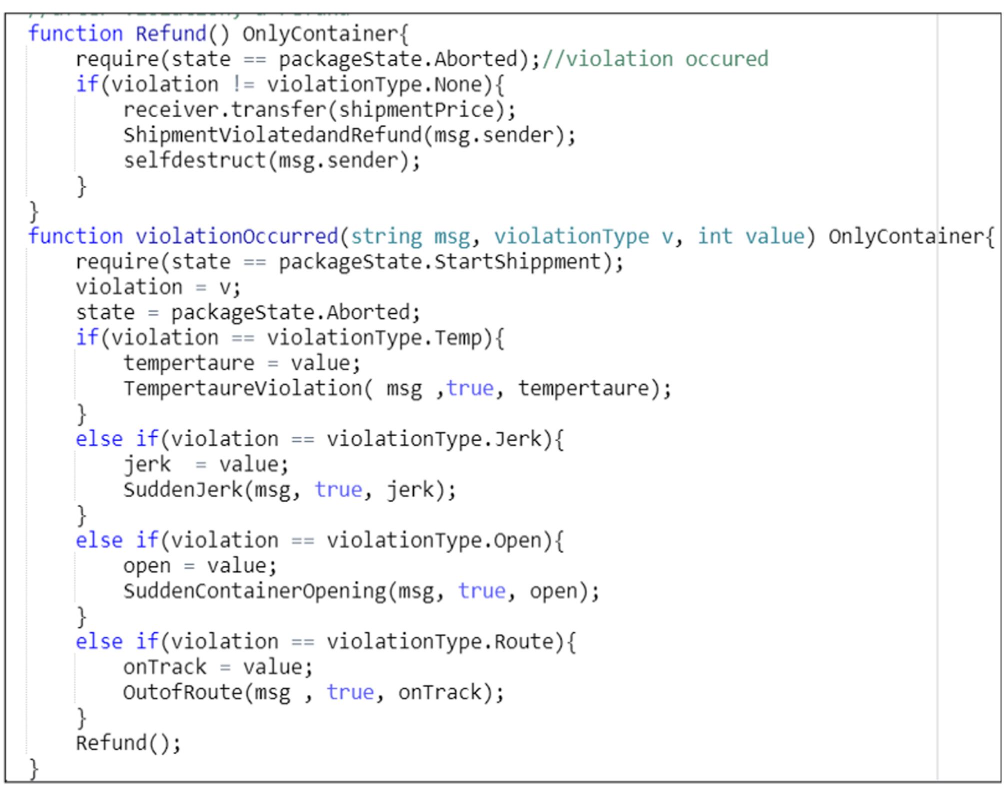


```
pragma solidity ^0.4.0;//version 0.4 or higher

contract IoTContainer{

    //participating entities with Ethereum addresses
    address container;
    address public sender_owner;
    address public receiver;
    string public content;//description of container content
    bytes32 public passphrase; //recived passphrase when money is deposited
    string public receivedCode; //recived code to be hashed
    enum packageState {
        NotReady, PackageContainerReadyforSelfCheck, ReadyforShipment,
        MoneyDeposited, StartShippment,WaitingforPassphrase, ReceiverAuthentiated,
        WaitingForCorrectPasscode, ShipmentReceived,
        AuthenticationFailureAborted,Aborted }
    packageState public state;
    uint startTime;
    uint daysAfter;
    uint shipmentPrice;
    //sensors
    enum violationType { None, Temp, Open, Route, Jerk}
    violationType public violation;
    int selfcheck_result;//1 or 0 indicating the self check result of IoTContainer
    int tempertaure; //track the tempertaure any integer
    int open; //if the container opens 1 , 0
    int onTrack; //to track the route 1 , 0
    int jerk;//sudden jerk 1, 0
```

Fig. 5. Attributes of the smart contact (The full code of the smart contract is available at https://github.com/smartcontract694/project.git).

48 h to provide the correct passphrase. If the receiver fails to provide the right passphrase within 48 h, 50% of the shipment price would be refunded to the receiver and the shipment is aborted. Figs. 7 and 8 illustrate the message sequence diagram of the logical flow of this interaction with function calls and events.

In Fig. 7, we present two scenarios with no violations. The first scenario is when the passphrase is provided within 48 h are incorrect. The other scenario shows the logical flow when the correct passphrase



```
function Refund() OnlyContainer{
    require(state == packageState.Aborted);//violation occured
    if(violation != violationType.None){
        receiver.transfer(shipmentPrice);
        ShipmentViolatedandRefund(msg.sender);
        selfdestruct(msg.sender);
    }
}
function violationOccurred(string msg, violationType v, int value) OnlyContainer{
    require(state == packageState.StartShippment);
    violation = v;
    state = packageState.Aborted;
    if(violation == violationType.Temp){
        tempertaure = value;
        TempertaureViolation( msg ,true, tempertaure);
    }
    else if(violation == violationType.Jerk){
        jerk  = value;
        SuddenJerk(msg, true, jerk);
    }
    else if(violation == violationType.Open){
        open = value;
        SuddenContainerOpening(msg, true, open);
    }
    else if(violation == violationType.Route){
        onTrack = value;
        OutofRoute(msg , true, onTrack);
    }
    Refund();
}
```

Fig. 6. Smart contract functions in case of violation (The full code of the smart contract is available at https://github.com/smartcontract694/project.git).

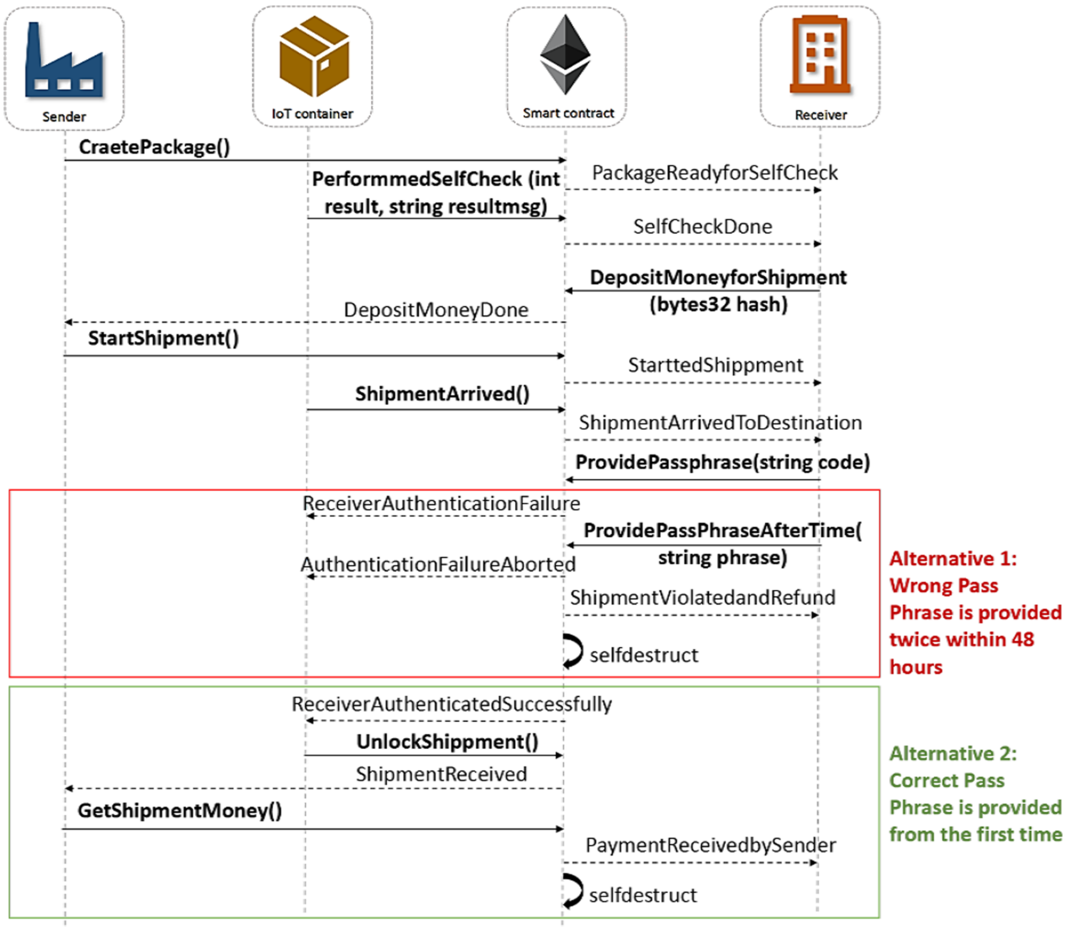

Sender
IoT container
Smart contract
Receiver
CraetePackage()
PerformmedSelfCheck (int result, string resultmsg)
PackageReadyforSelfCheck
SelfCheckDone
DepositMoneyforShipment (bytes32 hash)
DepositMoneyDone
StartShipment()
StarttedShippment
ShipmentArrived()
ShipmentArrivedToDestination
ProvidePassphrase(string code)
ReceiverAuthenticationFailure
ProvidePassPhraseAfterTime(string phrase)
AuthenticationFailureAborted
ShipmentViolatedandRefund
selfdestruct
Alternative 1: Wrong Pass Phrase is provided twice within 48 hours
ReceiverAuthenticatedSuccessfully
UnlockShipment()
ShipmentReceived
GetShipmentMoney()
PaymentReceivedbySender
selfdestruct
Alternative 2: Correct Pass Phrase is provided from the first time

Fig. 7. Message sequence diagram with no violations.

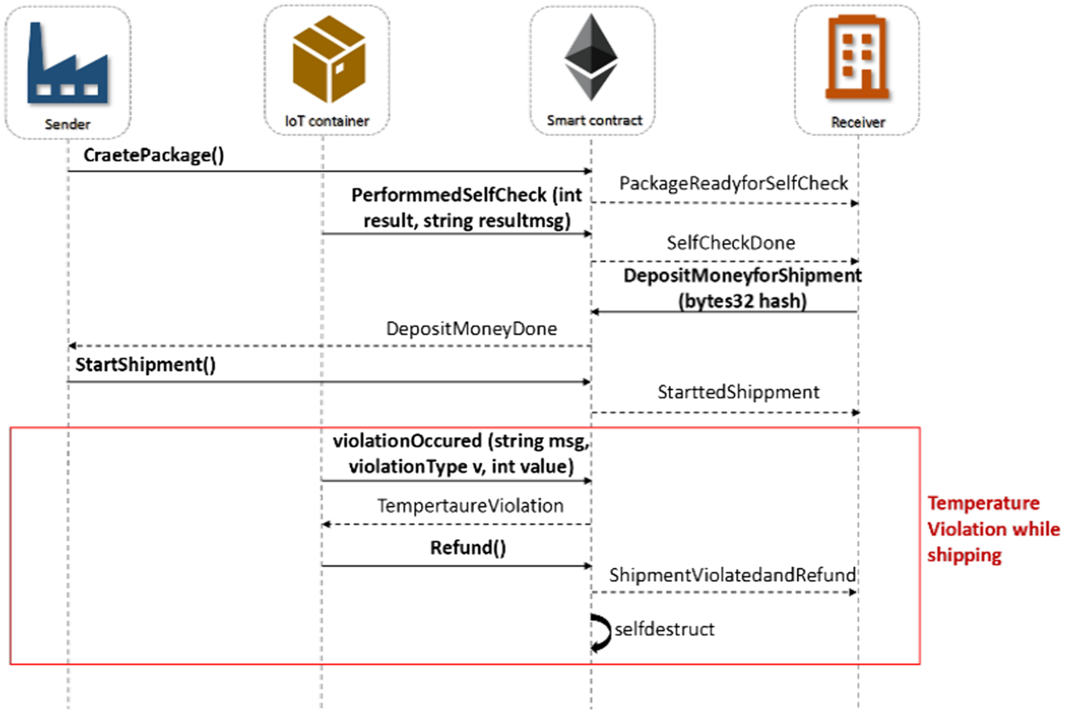
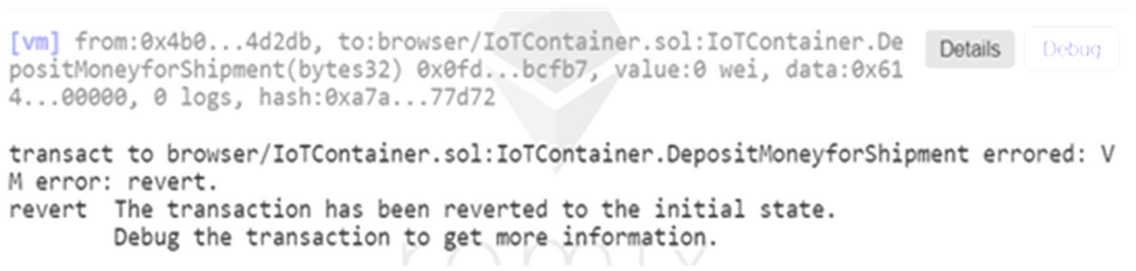
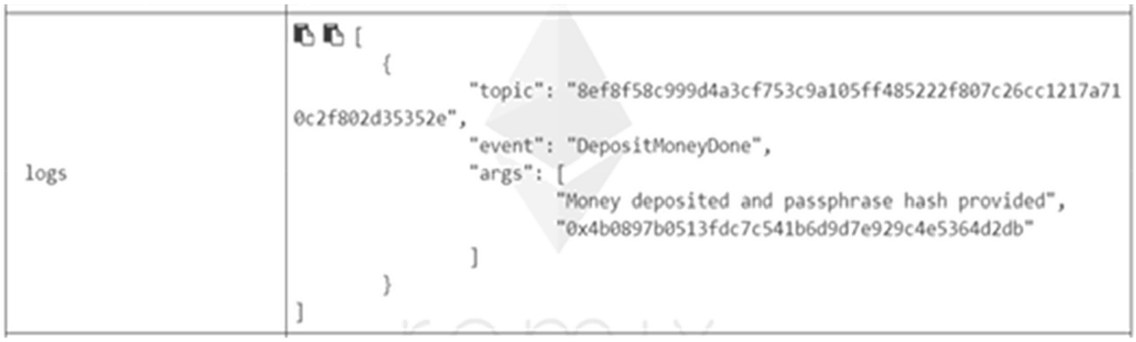

Sender
IoT container
Smart contract
Receiver
CraetePackage()
PerformmedSelfCheck (int result, string resultmsg)
PackageReadyforSelfCheck
SelfCheckDone
DepositMoneyforShipment (bytes32 hash)
DepositMoneyDone
StartShipment()
StarttedShippment
violationOccured (string msg, violationType v, int value)
TempertaureViolation
Refund()
ShipmentViolatedandRefund
selfdestruct
Temperature Violation while shipping

Fig. 8. Message sequence diagram with a temperature violation.



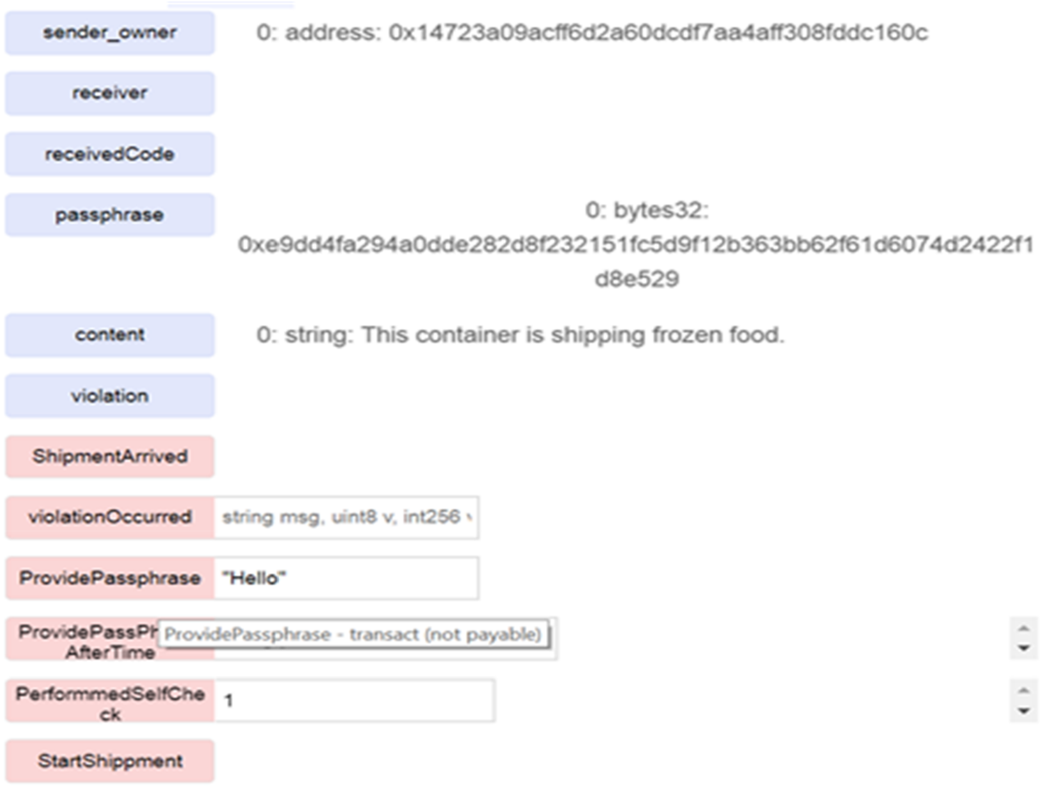
```
[vm] from:0x4b0...4d2db, to:browser/IoTContainer.sol:IoTContainer.De
positMoneyforShipment(bytes32) 0x0fd...bcfb7, value:0 wei, data:0x61
4...00000, 0 logs, hash:0xa7a...77d72

transact to browser/IoTContainer.sol:IoTContainer.DepositMoneyforShipment errored: V
M error: revert.
revert  The transaction has been reverted to the initial state.
        Debug the transaction to get more information.
```

Fig. 9. Error message when the receiver deposits money prior to self-check.



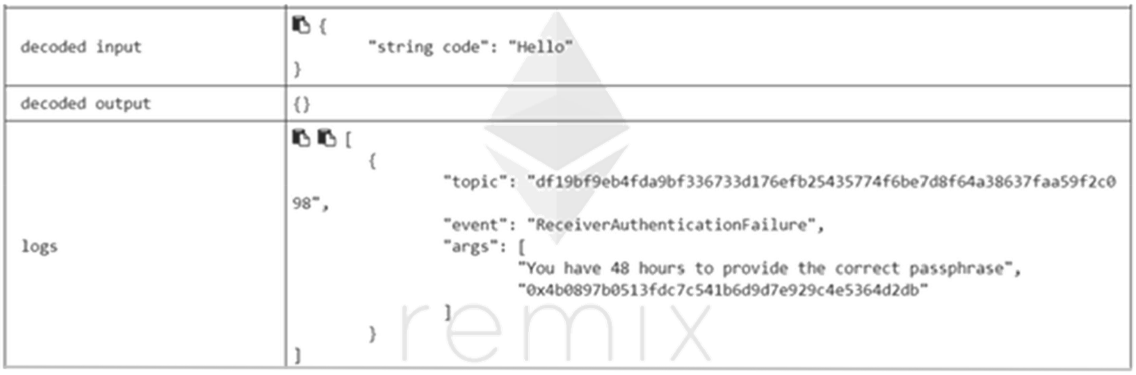
```
logs    [
            {
                    "topic": "8ef8f58c999d4a3cf753c9a105ff485222f807c26cc1217a71
0c2f802d35352e",
                    "event": "DepositMoneyDone",
                    "args": [
                            "Money deposited and passphrase hash provided",
                            "0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db"
                    ]
            }
        ]
```

Fig. 10. Log after calling `DepositMoneyforShipment(bytes32)` function (The full code of the smart contract is available at https://github.com/smartcontract694/project.git).



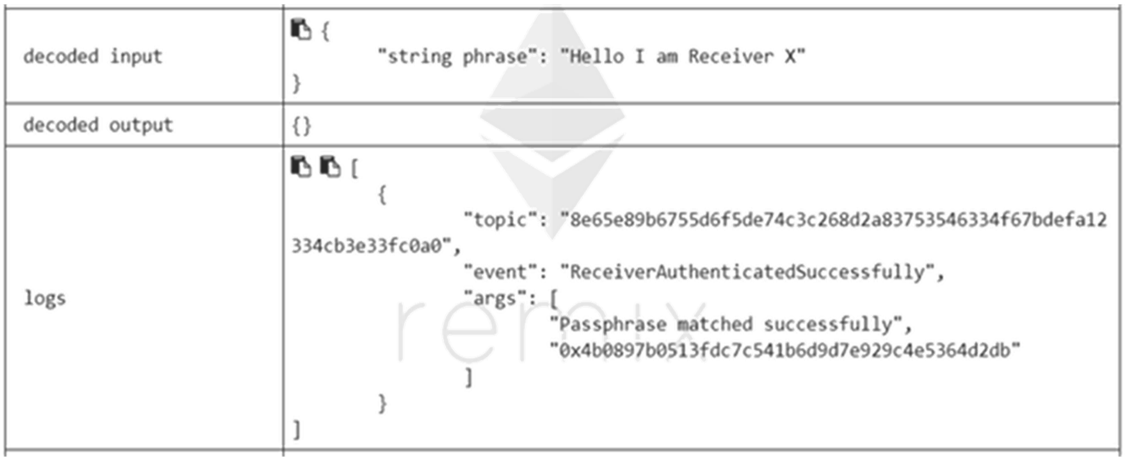
```
sender_owner              0: address: 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
receiver
receivedCode
passphrase                0: bytes32:
                          0xe9dd4fa294a0dde282d8f232151fc5d9f12b363bb62f61d6074d2422f1
                          d8e529
content                   0: string: This container is shipping frozen food.
violation
ShipmentArrived
violationOccurred         string msg, uint8 v, int256
ProvidePassphrase         "Hello"
ProvidePassPhraseAfterTime   ProvidePassphrase - transact (not payable)
PerformmedSelfCheck       1
StartShippment
```

Fig. 11. Receiver entering a wrong passphrase.



```
decoded input    {
                         "string code": "Hello"
                 }
decoded output   {}
logs             [
                         {
                                 "topic": "df19bf9eb4fda9bf336733d176efb25435774f6be7d8f64a38637faa59f2c0
                 98",
                                 "event": "ReceiverAuthenticationFailure",
                                 "args": [
                                         "You have 48 hours to provide the correct passphrase",
                                         "0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db"
                                 ]
                         }
                 ]
```

Fig. 12. Event `ReceiverAuthenticationFailure` is triggered.



```
decoded input    {
                         "string phrase": "Hello I am Receiver X"
                 }
decoded output   {}
logs             [
                         {
                                 "topic": "8e65e89b6755d6f5de74c3c268d2a83753546334f67bdefa12
                 334cb3e33fc0a0",
                                 "event": "ReceiverAuthenticatedSuccessfully",
                                 "args": [
                                         "Passphrase matched successfully",
                                         "0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db"
                                 ]
                         }
                 ]
```

Fig. 13. Passphrase is correct and `ReceiverAuthenticatedSuccessfully` is triggered.

is provided from the first time. Fig. 8 shows a scenario when a temperature violation has occurred.

5.2. Testing

We discuss the testing plan to demonstrate the correct functionality of three features. First, to ensure the status of package is updated correctly with the shipment flow. Second, appropriate events are triggered and logged in a logical sequence based on function calls. Third, the code verifies the legitimate receiver using the passphrase. For testing in Remix, three Ethereum addresses were used, each having 100 Ether as initial balance.

State of the Package: We tested the correct change of the state which is required prior to executing certain functions to ensure the proper execution flow and sequence. For instance, if the receiver tries to execute the `DepositMoneyforShipment(bytes32)` function before the self-check was performed by the IoT-enabled shipment, an error occurs as shown in Fig. 9. A similar error also occurs if any function is executed by the wrong party. An error would show if the `DepositMoneyforShipment(bytes32)` function was called by the IoT-enabled shipment container instead of the receiver.

Order of Events and Logs. We tested the correct order and sequence of the events and logs. For example, if the `DepositMoneyforShipment(bytes32)` function is executed successfully, the log will have the message "Money deposited and passphrase hash provided" as illustrated in Fig. 10. This message will only be in the log, if the money deposited is equal to the shipment price which was 10 Ether and if the receiver also entered the passphrase. After the event was triggered, the address of the receiver would have 10 Ether less. It is also noted that the passphrase is of the type `bytes32` and it should be entered as an array of one bytes while testing.

Passphrase Keccak256 Hash: The passphrase used while testing the code was the keccak256 hash "`Hello I am Receiver X`" which results an output hash of '`0xe9dd4fa294a0dde282d8f232151fc5d9f12b363bb62f61d6074d2422f1d8e529`'. The legitimate receiver should enter the correct passphrase to be hashed after the shipment arrives at the destination. This is done through the function `Providepassphrase(string)`. As mentioned earlier, the receiver has 48 h to enter the right passphrase if it was missed from the first time. As depicted in Fig. 11, the receiver will enter the passphrase "`Hello`" instead of "`Hello I am Receiver X`" in the first attempt. This will result in the event `ReceiverAuthenticationFailure` to be triggered as seen in Fig. 12. A successful subsequent attempt within 48 h with the right passphrase as a parameter to the function `ProvidePassPhraseAfterTime(string)` will trigger the event `ReceiverAuthenticatedSuccessfully` with the message "`Passphrase matched successfully`" shown in the logs, as depicted in Fig. 13.

6. Conclusions

The application of blockchain technology in supply chain and logistics has tremendous advantages in terms of product visibility, tracking and process automation. In this paper, we propose a blockchain-based solution for shipment supply chain management using smart containers. Our proposed solution utilizes the features of smart contracts in Ethereum blockchain to govern and manage the interaction between the sender and receiver of the shipment. We provide details of system interaction among the supply chain participants, and tested various functionalities of the system using Remix IDE. Although our solution is focused on single echelon supply chain; the presented system architecture, design, and code are generic enough and can be applied to multi-echelon shipment management. The program code has been made publicly available for academic, research and practice community. As future work, we plan to implement a fully functional system consisting of an IoT-enabled container equipped with Raspberry Pi board connected to various sensors, and develop front-end DApps (decentralized applications) and wallets for the sender and the receiver to interact with the Ethereum smart contracts.

Acknowledgement

This publication is based upon work supported by the Khalifa University of Science and Technology under Award No. CIRA-2019-001.

References

Angeles, R. (2005). RFID technologies: Supply-chain applications and implementation issues. *Information Systems Management, 22*(1), 51–65.
Bahga, A., & Madisetti, V. K. (2016). Blockchain platform for industrial internet of things. *Journal of Software Engineering and Applications, 9*(10), 533.
Banerjee, A. (2018). Blockchain technology: supply chain insights from ERP. *Advances in computers: Vol. 111*, (pp. 69–98). Elsevier.
Bocek, T., Rodrigues, B. B., Strasser, T., & Stiller, B. (2017). Blockchains everywhere-a use-case of blockchains in the pharma supply-chain. *2017 IFIP/IEEE symposium on integrated network and service management (IM)* (pp. 772–777). IEEE.
Buterin, V. (2014). A next-generation smart contract and decentralized application platform. *White Paper*.
Chou, P. H., Lee, C. T., Peng, Z. Y., Li, J. P., Lai, T. K., Chang, C. M., ... Lai, L. Y. (2013). A Bluetooth-Smart insulating container for cold-chain logistics. *2013 IEEE 6th international conference on service-oriented computing and applications* (pp. 298–303). IEEE.
Christidis, K., & Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access, 4*, 2292–2303.
Huh, S., Cho, S., & Kim, S. (2017). Managing IoT devices using blockchain platform. *2017 19th international conference on advanced communication technology (ICACT)* (pp. 464–467). IEEE.
IBM Blockchain, Available at https://www.ibm.com/blockchain/platform/ [Accessed: Oct. 29, 2017].
Jayaraman, R., Salah, K., & King, N. (2019). Improving opportunities in healthcare supply chain processes via the internet of things and blockchain technology. *International Journal of Healthcare Information Systems and Informatics (IJHISI), 14*(2), 49–65.
Khan, M. A., & Salah, K. (2018). IoT security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems, 82*, 395–411.
Kim, H. M., & Laskowski, M. (2018). Toward an ontology-driven blockchain design for supply-chain provenance. *Intelligent Systems in Accounting, Finance and Management, 25*(1), 18–27.
Ko, D., Kwak, Y., Choi, D., & Song, S. (2015). Design of cold chain application framework (CCAF) based on IOT and cloud. *2015 8th international conference on u-and e-service, science and technology (UNESST)* (pp. 11–13). IEEE.
Lee, D., & Park, J. (2008). RFID-based traceability in the supply chain. *Industrial Management & Data Systems, 108*(6), 713–725.
Levinson, M. (2016). *The box: How the shipping container made the world smaller and the world economy bigger-with a new chapter by the author.* Princeton University Press.
Li, F., & Chen, Z. (2011). Brief analysis of application of RFID in pharmaceutical cold-chain temperature monitoring system. *Proceedings 2011 international conference on transportation, mechanical, and electrical engineering (TMEE)* (pp. 2418–2420). IEEE.
Lin, L. C. (2009). An integrated framework for the development of radio frequency identification technology in the logistics and supply chain management. *Computers & Industrial Engineering, 57*(3), 832–842.
Mackey, T. K., & Nayyar, G. (2017). A review of existing and emerging digital technologies to combat the global trade in fake medicines. *Expert Opinion on Drug Safety, 16*(5), 587–602.
Musa, A., Gunasekaran, A., & Yusuf, Y. (2014). Supply chain product visibility: Methods, systems and impacts. *Expert Systems with Applications, 41*(1), 176–194.
StaTwig: Improving food and vaccines distribution systems more efficiently through blockchain- UNICEF Innovations (2019). Available at: https://www.unicef.org/innovation/stories/statwig-improving-food-and-vaccines-distribution-systems-more-efficiently-through. [Accessed: May. 29, 2019].
Tian, F. (2016). An agri-food supply chain traceability system for China based on RFID & blockchain technology. *2016 13th international conference on service systems and service management (ICSSSM)* (pp. 1–6). IEEE.
Tian, F. (2017). A supply chain traceability system for food safety based on HACCP, blockchain & Internet of things. *2017 International conference on service systems and service management* (pp. 1–6). IEEE.
Toyoda, K., Mathiopoulos, P. T., Sasase, I., & Ohtsuki, T. (2017). A novel blockchain-based product ownership management system (POMS) for anti-counterfeits in the post supply chain. *IEEE Access, 5*, 17465–17477.
Urien, P., & Piramuthu, S. (2013). Internet Smart Card for perishable food cold supply chain. *2013 IEEE Eighth international conference on intelligent sensors, sensor networks and information processing* (pp. 83–88). IEEE.
Yue, X., Wang, H., Jin, D., Li, M., & Jiang, W. (2016). Healthcare data gateways: Found healthcare intelligence on blockchain with novel privacy risk control. *Journal of medical Systems, 40*(10), 218.
Zhao, J. L., Fan, S., & Yan, J. (2016). Overview of business innovations and research opportunities in blockchain and introduction to the special issue.
Zobel, A. (2016). The complete journey clinical trial supply. *International Clinical Trials,* 14–16.