AFRL-RI-RS-TR-2020-106

END-TO-END SUPPLY CHAIN MANAGEMENT TROUGH BLOCKCHAIN PREVENTING COUNTERFEIT AND MALICIOUS PRODUCTS

JACKSON STATE UNIVERSITY

JUNE 2020

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2020-106 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
TODD N. CUSHMAN
Work Unit Manager

/ S /
JAMES S. PERRETTA
Deputy Chief, Information Exploitation
And Operations Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JUNE 2020	FINAL TECHNICAL REPORT	MAR 2019 – DEC 2019

4. TITLE AND SUBTITLE

END-TO-END SUPPLY CHAIN MANAGEMENT TROUGH BLOCKCHAIN PREVENTING COUNTERFEIT AND MALICIOUS PRODUCTS

5a. CONTRACT NUMBER
N/A

5b. GRANT NUMBER
FA8750-19-1-0020

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Célestin Wafo Soh

5d. PROJECT NUMBER
BC2S

5e. TASK NUMBER
CM

5f. WORK UNIT NUMBER
TB

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Jackson State University
1400 J. R. Lynch Street
Jackson MS 39127

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2020-106

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We have built a scalable pull supply chain management system which allows a secure and trusted recording of events, the verification of authenticity and provenance of products, the proof of ownership of goods and complete traceability of assets. Our ultimate goal is to detect counterfeits or malicious actors/components as assets move gradually through manufacturing processes of separately owned businesses and possibly multiple national or international shipments, before they are delivered to the USAF.

15. SUBJECT TERMS

Blockchain, supply chain management, smart contracts, parts traceability

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**TODD N. CUSHMAN**
U	U	U	UU	30	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

Table of Contents

List of Figures

Acknowledgments

Gratefully acknowledge the contributions of Dr. Bassirou Diatta (Summer Research Scholar), Dr. Jules K. Waku (Visiting Research Scholar), and Greg Offah (Student Researcher). Also, would like to thank Mrs. Gretta Ashley, our Office Manager, for helping with several administrative aspects of this project.

1.0 SUMMARY

We have leveraged the Ethereum blockchain [1] and Solidity [2] smart contract technologies together with novel algorithms to build a pull supply chain management system. Specifically, we have

1. modeled the business network involved in a pull supply chain,
2. modeled the interaction between participants and assets in a pull supply chain using appropriate transactions,
3. ensured the security, traceability and immutability of transactions by employing Ethereum blockchain and MetaMask wallet,
4. emitted relevant events to authorized network participants as they occur,
5. and packaged our solution as a web-based distributed application[3] with detailed documentation.

2.0 INTRODUCTION

The electronic component supply chain is complex and non-integrated. It is plagued by shortages due primarily to economic or political uncertainties, tough national and international regulations, the scarcity of raw materials and the fast speed of innovations in consumer electronics' businesses. These challenges create opportunities for malicious agents in the electronic component supply chain. As a result, there is a steady flow of counterfeit electronic components in the supply chain. This parallel flow adversely affects consumer electronics. To which, there are many counterfeit electronic components entering our defense supply chain. They imperil our troops and endanger electronic systems and sensors in which the breakdown of a single part could undermine a mission or cause severe injuries and even deaths. In 2011, a Senate Armed Services Committee (SASC) investigation found at least 1,800 cases of counterfeit parts in US weapons and one million suspected fake components in the defense supply chain. These compromised parts were found for instance in the SH-60B helicopter's Forward Looking Infrared System, a memory chip in the L-3 Display System on the USAF's (United Sates Air Force) C-130j and C-27j cargo planes, and the ice detection module on a Navy P-8A Poseidon commercial airplane. The report of the 2011 SASC investigation was publicized on May 21, 2012. It resulted in congressional responses contained in theFY2012 and FY2013 National Defense Authorization Acts (NDAAs). In order to comply with these NDAAs, the DoD (Department of Defense) issued several Defense Federal Acquisition Regulation Supplements (DFARS). Perhaps the most pertinent and consequential ones to the electronic component supply chain are DFARS 252.246-7007 (Contractor Counterfeit Electronic Part Detection and Avoidance System) and DFARS 252.246-7008 (Sources of Electronic Parts). They ensure the traceability of electronic parts entering the DoD supply chain while defining a hierarchy for procurement by authorized contractors.

We have built a scalable pull supply chain management system which allows a secure and trusted recording of events, the verification of authenticity and provenance of products, the proof of ownership of goods and complete traceability of assets. Our ultimate goal is to detect counterfeits or malicious actors/components as assets move gradually through manufacturing processes of separately owned businesses and possibly multiple national or international shipments, before they are delivered to the USAF. We have leverage the Ethereum blockchain[1] and Solidity [2] smart contract technologies together with novel algorithms to build the proposed management system. Specifically we have

a. modeled the business network involved in a pull supply chain,

b. modeled the interaction between participants and assets in a pull supply chain using appropriate transactions,

c. ensured the security, traceability and immutability of transactions by employing Ethereum blockchain and MetaMask wallet,

d. emitted relevant events to authorized network participants as they occur,

e. and packaged our solution as a web-based distributed application [3] with a detailed documentation.

This technical report comprises five chapters including the summary and this introduction. Chapter 3 has five sections. Section 3.1 is concerned with a summary of supply chain management with an emphasis on management strategies. This section is important for understanding supply chain flows (information and materials) and our choice of the pull strategy. In Section 3.2, we review blockchain technologies and their application to supply chain management. Sections 3.3 deals with a detailed description of the data structures and state variables of our smart contract. In Section 3.4, we present functions that are used for reading and writing data. These are functions that are employed for interacting with the blockchain. Section 3.5 describes the events that are emitted as participants interact with our smart contract. Perhaps Chapter 4 is technically the most important one. It deals with the deployment and the graphical user interface (GUI) of our smart contract. In the last chapter i.e. Chapter 5, we recapitulate our findings.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Supply Chain Management

Supply chain management (SCM) is a collection of processes that are involved in producing goods and bringing them cost efficiently and effectively to consumers. It has five fundamental stages: planning, sourcing, manufacturing, delivery and return. Within SCM, there are material, data and cash flows. These flows are complex and must be digitally visible to enable and facilitate decision-making.

3.1.1 Supply Chain Flows and Administration

A supply chain [4] is a network used to provide goods or services from raw materials to end- consumers through a properly crafted flow of information, physical distribution and payments. It is the linkage between producers of goods and their consumers. It is a complex network that necessitates a proper management in order to optimally satisfy all its participants (consumers, retailers, distributors, manufacturers and raw materials suppliers). Arguably, the backbone of a supply chain is supply chain management (SCM) [4] i.e. a collection of processes and strategies employed by companies to ensure that their supply chain is efficient and cost effective. In order to operationalize a business strategy, a supply chain plan must be adopted. This is the first step of SCM. Typically, a supply chain strategy aims at driving down operational cost while maximizing efficiency. It establishes how to work with suppliers, distributors, customers and customers' customers. The second step in SCM is sourcing. It is concerned with building strong relationships with suppliers of raw materials needed for production. It determines different planning methods for shipping, delivery and payments. The third step of SCM deals with making the actual product demanded by the customer. It comprises designing, producing, testing, packaging and synchronizing products for delivery. The fourth step of SCM plans the delivery of products to the customer at the destined location by the supplier. This stage is mainly concerned with logistics. It deals with receiving customer’s orders, establishing a network of warehouses, picking carriers to deliver goods to customers and setting up an invoicing system to receive payments. The fifth and last step of SCM is the return. It deals with defective or damaged products that are returned by customers.

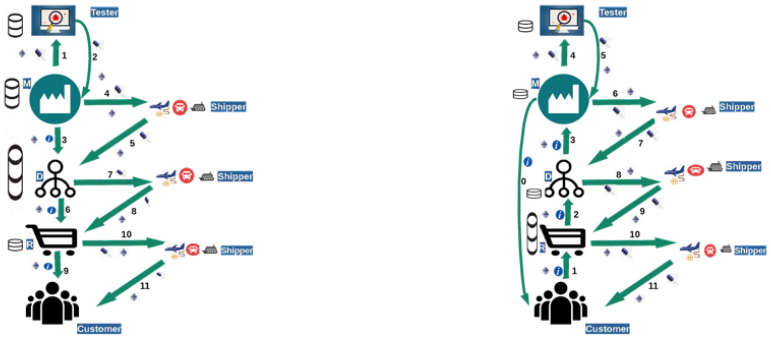
There are three main types of flow in SCM: (1) material flow (goods from producers to consumers and vice versa), (2) information/data flow (request for quotation, purchase orders, engineering changes requests etc.), and (3) money flow. These flows are spaghetti-like yet they are critical for decision-making. Thus, they must be visible at a click of a button in the management system. The components of SCM flows include shipment, warehousing, sourcing and procurement, returns management, and post-sale services.

3.1.2 Pull and Push Management Strategies

Pull or push strategies are used to accommodate the supply and demand of a product. When we shop for an item, we expect to find it in the appropriate store. How the retailer manage to keep up with the demands of customers must be properly planned. Indeed having too much or too little of a given product may affect negatively a retailer's bottom line. This is the inventory risk problem. Take for instance a furniture retailer. Does he stock on furniture or simply order furniture as customers demand? On one hand, if he just stock on furniture from wholesalers hoping that customers might buy them, this is a push strategy. One advantage of push strategy is cost: the prices of products are fixed by wholesalers long in advance. These product themselves are made by manufactures and sold to wholesalers without knowledge of demands from customers. The problem with push strategy is that retailers might end up with too many goods that are not consumed. Thus, they run a high inventory risk. The same type of inventory risk may apply to wholesalers too: They may be unable to convince enough retailers to buy their products. On the other hand, retailers may stock on product based solely on consumers' demands. In their turn, wholesalers may order from manufactures only products demanded by retailers. Thus, manufacturers produce only on demand. This is a pull strategy: it is demand driven. Besides reducing inventory risk, pull strategy may remediate raw material shortage. Indeed, there is reduced waste and almost certain guarantee that the product will be sold. Pull strategy is manufacturer driven while push strategy is consumer driven. We have illustrated these two strategies in Figures1a and 1b.

3.1.3 Electronic Components Supply Chain: Pull Strategy Use Case

The electronic component supply chain is plagued by shortages mainly caused by economic and political qualms, rapid products update and the scarcity of raw materials. These challenges create opportunities for malicious agents to enter the supply chain. In 2011, an investigation by the Senate Armed Service Committee (SACS) uncovered several counterfeit parts in our defense supply chain that originated mostly from China. As a result, FY2012 and FY2013 National Defense Authorizations Acts (NDAAs) had provisions to comprehensively address this problem. The DoD implemented these regulations through several Defense Federal Acquisition Supplements (DFARS) which comprises DFARS 252.246 7007 (Contractor Counterfeit Electronic Part Detection and Avoidance System) and DFARS 252.246 (Sources of Electronic Parts). They ensure the traceability of electronic parts entering the DoD supply chain while defining a hierarchy for procurement by contractors.



(a) Push strategy. (b) Pull strategy.

Figure 1: Push Strategy vs. Pull Strategy

We propose the usage of pull strategy in DoD supply chain in order to address some of the problems described above. Such strategy will be implemented through a smart contract running on a blockchain. Indeed such strategy addresses for instance the shortage of raw material problem. Also since pull strategy is consumer driven, we can control products fabrication and distribution to avoid entries of malicious goods.

3.2 Application of Blockchain to Supply Chain Management

A blockchain is an immutable data structure relying on cryptographic algorithms and protocols to securely record transactions among possibly untrusted nodes of a peer-to-peer network [5]. It relies on onsensus algorithms for validating transactions that are then emorialized in blocks which are accessible to all or authorized nodes. These blocks are chained using cryptographic hashes resulting in a temper-evident log.

3.2.1 Brief Introduction to Blockchain Technologies

Arguably, Bitcoin [5, 6], a digital currency is the most successful implementation of block-chain technologies. The success of Bitcoin is quite amazing if one considers the lifespans and multiple failures of similar enterprises post-Bitcoin. The emergence of Bitcoin may be attributed to several technological innovations comprising the use of a blockchain and a de- centralized model of governance that permeates secure and almost anonymous user-to-user transactions, even in the absence of trust among participants or failure of few nodes in its network.

A blockchain is a set of protocols and cryptographic algorithms that allows a clique of computers to securely record data using a shared open database called a keyledger. The latter is often call a blockchain because of the data structure it uses to save information. In this distributed public database, data are recorded in blocks that are sequentially linked using hash pointers pointing backward to previous blocks. The job of a hash is twofold: It indicates where the data is stored and includes the cryptographic hash of that data at some point in time. A classical pointer allows us to

quickly retrieve data. However, a hash pointer additionally provides a way to verify whether the value pointed to has changed. Thus, if an adversary modifies data anywhere in the blockchain, it will result in the hash pointer in the next and subsequent blocks being incorrect. This property allows the blockchain to be a tamper-evident log. A typical block in a blockchain comprises information about several transactions that have occurred on the network. This information is summarized into a Merkle tree [5], a special binary tree, which facilitates the secure and fast verification of whether a particular transaction belongs to a given block.

Unlike traditional databases, every transaction on a blockchain is made public and every node of the network can write onto it. This necessitates users to be anonymous to prevent a transaction being tied to a specific identity. Anonymity is achieved by using a pair of cryptographically secure keys: a public key and a private key. The public key is a proxy for the address used for sending and receiving transactions (on the Ethereum blockchain, the first 20 bytes of the KECCAK-256 of the public key serves as the address). The private key is kept secret and is employed for digitally signing transactions from the corresponding address. Anonymity poses the problem of trust: How can we be sure that anonymous nodes are trustworthy when they add information to the ledger? This problem is solved through consensus algorithms (Proof of Work [6], Proof of Stake [7], Proof of Activity [7] , Proof of Burn [8], etc.) that are predicated on the fact that the majority of users of a blockchain have an incentive to be honest. It is important to stress that if anonymity is not required, like on private or permissioned blockchains, permissions may be employed to control who can read and write onto the blockchain. A typical blockchain platform comprises: (1) a ledger which is a distributed, immutable historical record of all validated transactions occurring over the network, (2) a peer-to-peer network which stores, updates and maintains the ledger, (3) membership services for user authentication, authorization and addresses management, (4) smart contracts i.e. programs made of conditional statements that run on the blockchain in response to certain events, (5) events notification services for updates and actions on the blockchain, (6) wallets for keeping users credentials, (7) systems management for components creation, modification and monitoring , and (8) system integration for the connection of the blockchain with external systems.

3.2.2 Tracking Products Using Smart Contracts

We have implemented the pull strategy as a smart contract on the Ethereum blockchain. Such contract allows participants to tract products from ordering to delivery. The contract is implemented in Solidity, a language for writing smart contract that is inspired by C++, Python and JavaScript. The governance of our smart contract is ensured by an owner and administrators. The owner of the smart contract can give administrative rights to ad-dresses (administrators) or transfer ownership of the contract. Administrators can register participants which comprise consumers, retailers, distributors (wholesalers), manufactures and testers. Only consumers can order products. Also, the creation of assets is reserved to manufacturers. Once an order is created, it must be transferred from the consumer

(creator of the order) to a manufacturer through a retailer and a distributor. Once a product is manufactured, it must be tested before shipment. When an order is transferred between participants, its recipient is the new owner. Only the owner of an order can cancel it. An asset must be destroyed only by its owner. In the subsequent chapters, we described in details data structures, functions, deployment and the graphical user interface (GUI) of our smart contract.

3.3 Data Structures and Properties of Our Smart Contract

In this chapter we present the data structures associated with our smart contract. There are two categories of data structures: Enumeration (enum) and structure (struct). Category is of type enum whereas Participant, Order and Asset belong to the type struct. In addition, we introduce public and private properties of our smart contract.

3.3.1 Data Structures

Category Type: enum

Description:

- 0: Customer
- 1: Retailer
- 2: Distributor
- 3: Manufacturer
- 4: Tester
- 5: Shipper

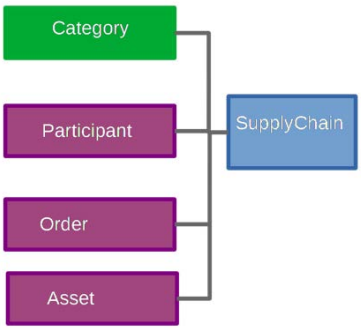


Figure 2: Data structure of our smart contract

Participant

Type: struct Fields:

- status bool: determines whether a participant is registered or not
- account address: address of the participant
- category Category: participant's scategory

Order

Type: struct Fields:

- origin address: origin of the order
- destination address: destination of the order
- description bytes32: description of the order

Asset

Type: struct Fields:

- uid uint: serial number of an asset
- tested bool: determines whether an asset has been tested
- manufacturer address: address of the manufacturer
- description bytes32: description of the asset

3.3.2 State Variables

- owner address : owner's address
- numAdmin uint : number of administrators
- numSuspendedAdmin uint : number of suspended administrators
- currentOrder uint : serial number of the latest order
- numPart uint : total number of participants
- numCustomers uint : total number of customers
- numRetailers uint : total number of retailers
- numDistributors uint : total number of distributors
- numManufacturers uint : total number of manufacturers
- numTesters uint : total number of testers
- numShippers uint : total number of shippers
- assetCount uint : total number of assets created
- assetDestroyed uint : total number of assets

- partStore address = > Participant : stores participants
- suspendedAccounts address = > bool : stores the addresses of suspended participants
- orderStore uint = > Order : stores orders
- orderWallet address = > (uint = > bool) : stores owners of orders
- tracker uint = > address[2] : stores the addresses of originators and targets of the latest transfers of orders or assets corresponding to orders
- pathLength uint => uint : stores the number of steps taken by an order.
- assetStore uint = > (uint = > Asset) : stores assets
- assetWallet address => (uint => (uint => bool)) : used for proving ownership of an asset
- orderNum2Serial uint = > uint : correspondence between order's serial number and asset's serial number
- serialNum2Order uint = > uint : correspondence between asset's serial number and order's serial number
- destroyedAssets uint = > (address = > bool) : used for proving that an asset has been destroyed
- adminRight address = > bool : used to determine if an address has administrative rights
- suspendedAdminQ address = > bool b: used to determine whether an administrator has been suspended

Note: A public mapping automatically generates a public getter function.

3.4 Functions: Calls and Operations of our Smart Contract

The goal of this chapter is to describe functions of the SupplyChain smart contract. There are two types of functions in this smart contract: calls and operations. Calls are functions that do not modify the state of the smart contract (and by ricochet that of the blockchain). They just read data from the blockchain. They are symbolized with the icon C *after* their names. Operations are functions that modify the state of the smart contract and are indicated with O after their names. Calls and operations are invoked through transactions. Transactions that modify the state of the blockchain tend to be more expensive than those that do not. Thus, in general, operations will be more costly than calls.

3.4.1 Calls and Operations

constructor C

Parameters:

- _owner address: address of the owner of the smart contract

Usage:

- Instantiates the smart contract

changeContractOwner O

Parameters:

- _newOwner address: new owner of the contract Usage:
- Changes the ownership of the contract.
- Only the contract's owner can execute this transaction.

Usage:

- Changes the ownership of the contract.

addAdmin O

Parameters:

- _account address: an administrator's address

Usage:

- Adds an administrative address.
- Only the contract's owner can execute this transaction.

suspendAdmin O

Parameters:

- _account address: administrator's address to be suspended Usage:
- Suspends an administrator.
- Only the contract's owner can sign this transaction.

readmitAdmin O

Parameters:

- _account address: suspended administrator's address

- Usage:
- Readmits a suspended administrator.

registerParticipant O Parameters:

- _account address: address to be registered
- _category Category: category of the participant (e.g Category.MANUFACTURER) Usage:
- This function is used by contract's administrators for registering participants (cus- tomers, retailers, distributors, manufacturers, testers and shippers)
- A participant may be registered only once
- Only an administrator can execute this transaction

getRegistrationStatus O Return

Parameters:

- bool: status of the participant; true means registered and false means not registered Usage:
- This function is used for checking whether an address is registered.

suspendParticipant O Parameters:

- _account address: account to be suspended Usage:
- This function is used by an administrator for suspending a participant

suspendedParticipant O

Parameters:

- _account address: account of a registered participant Return

Parameters:

- _suspQ bool: true if the participant has been suspended and false otherwise Usage:
- This function is for checking whether a participant is suspended.

readmitParticipant O Parameters:

- _account address: account of the participant to be readmitted Usage:
- This function is used by the administrator for readmitting a suspended participant.
- This transaction must be executed by an administrator.

createOrder O

Parameters:

- _destination address: retailer's address
- _description bytes32: description of the order Usage:
- This function is employed by the customer for creating an order targeting a specific retailer.
- Orders are created only by customers.

orderOwnership C Parameters:

- _order uint: serial number of the order
- _account address: owner's address

Return Parameters:

- _status bool: true if the address belongs to the owner of the asset and false otherwise

Usage:

- This function determines whether a certain participant owns a given order.
- The order is identified by its serial number and the participant with its address.

transferOrder O

Parameters:

- _destination address: destination of the order
- _order uint: order's serial number

Usage:

- This function is used to transfer orders between customers, retailers and distributors.
- The originator of the transfer must be the current owner of the order.

cancelOrder O

Parameters:

- _order uint: order's serial number

Usage:

- This function is used for canceling an order.
- An order can be canceled only by its owner.

createAsset O

Parameters:

- _order uint: order's reference number
- _description bytes32: order's description

Usage:

- This function is used by a manufacturer for creating an asset.
- An asset is created as a response to a given order.
- Only manufacturers can create assets.

assetOwnership C Parameters:

- _currentOwner address: current asset owner's address
- _serial uint: serial number of the asset
- _order uint: reference number of the order that lead to the creation of the asset Return

Parameters:

- _status bool: true if the address is that of the owner and false otherwise

Usage:

- This function determines whether an asset belongs to a given participant.
- The participant is identified by its address.
- The asset is identified by the order which lead to its creation and its serial number.

transferAsset2Tester O

Parameters:

- _destination address: tester's address
- _order uint: reference number of the order
- _assetCount uint: serial number of the asset Usage:
- This function is used by a manufacturer for transferring an asset to a tester.
- This transaction can be executed only by a manufacturer.

transferAsset2Manufact O

Parameters:

- _destination address: manufacturer's address
- _order uint: reference number of the order
- _assetCount uint: serial number of the asset Usage:

  This function is used by the tester for transferring a tested asset to the manufacturer who requested the test.
  •

transferAsset O

Parameters:

- _destination address: new asset owner's address
- _order uint: reference number of the order that lead to the creation of the asset
- _assetCount uint: asset's serial number

Usage:

- This function is used for transferring asset that has been tested and passed.
- Only the current owner of the underlying asset can execute this transaction.

isDelivered C

Parameters:

- _destination address: destination of the asset.
- _order uint: reference number of the order.
- _assetCount uint: serial number of the asset. Return

Parameters:

- _delvStatus bool: true if the asset has been delivered and false otherwise.

Usage:

- This function tells whether an order has been delivered to the customer. Usage:
- This function determines the length of the path of an order up to the last transfer.
- The order must be delivered.

trackOrder C

Parameters:

- _order uint: serial number of the order Return

Parameters:

- (_from, _to) (address, address): end points of the latest link in the path of an order

Usage:

- This function provides the latest link on the path of an order.
- When the order is created, the end points of the link are the same and equal to the address of the customer who created the order.

getSerialNum C

Parameters:

- _order uint: reference number of an order Return

Parameters:

- _serialNum uint: serial number of the asset Usage:
- This function tells the serial number of an asset created in response to an order.
- If the asset has not yet been created, this function returns 0.

getOrderNum C

Parameters:

- _serial uint: serial number of the asset Return

Parameters:

- _orderRef uint: reference number of the order

Usage:

- This function determines the serial number of the order which lead to the creation of the asset with serial number _serial.

destroyAsset *C*

Parameters:

- _order uint: reference number of the order that lead to the asset's creation
- _assetCount uint: serial number of the asset

Usage:

- This function is used by the owner of an asset for destroying it.

3.5 Events

Events are messages broadcast by the blockchain in response to certain transactions. Since saving information on the blockchain can be costly, events which are comparatively cheaper are used as a trade-off. In this chapter, we describe all the events of our smart contract.

OrderCreated

Parameters:

_to address: destination of the order

_orderRef uint: serial number of the order

_ description bytes32: description of the order

OrderTransferred

Parameters:

_from address: origin of the order

_to address: destination of the order

_orderRef uint: serial number of the order

AssetCreated

Parameters:

_creator address: manufacturer's address

_serialNumuint: asset's serial number

_orderRef uint: order's serial number

AssetTesting

Parameters:

_from address: manufacturer's address

_to address: tester's address

_serialNum uint: asset's serial number

_orderRef uint: order's serial number

AssetTested

Parameters:

_from address: tester's address

_to address: manufacture's address

_serialNumuint: asset's serial number

_orderRef uint: order's serial number

_result bytes32: test's result

AssetDestroyed

Parameters:

_owner address: asset owner's address

_serialNum uint: asset's serial number

_orderRef uint: order's serial number

NewAdminAdded

Parameters:

_newAdmin address: newly added administrator's address

AdminSuspended

Parameters:

_suspAdmin address: suspended administrator's address

AdminRestore

Parameters:

_admin address: restored administrator's address

NewOwner

Parameters:

_newOwner address: new contract owner's address

xRegisteredBy

Parameters:

_partAddress address: registered participant's address

_category uint: participant's category

_admin uint: registering administrator's address

xSuspendedBy

Parameters:

_partAddress address: registered participant's address

_category uint: participant's category

_admin uint: suspending administrator's address

xReadmittedBy

Parameters:

_partAddress address: registered participant's address)

_category uint: participant's category

_admin uint: readmitting administrator's address

4.0 RESULTS AND DISCUSSION

In this chapter, we start by describing the tools employed for deploying and generating a graphical user interface (GUI) for our smart contracts. They comprise the following three tools:

- Remix IDE,
- MetaMask,
- OneClickDapp.

Next, we explain how to deploy and interact with our smart contract through its GUI.

4.1 Tools

4.1.1 Remix IDE

Remix IDE is an in-browser editor for writing, compiling, debugging, testing and de-ploying smart Ethereum contracts. It also has several plugins for performing several specialized tasks associated with Solidity contracts. It has an extensive documentation available at https://remix- ide.readthedocs.io/en/latest/. The IDE is available at https://Remix.ethereum.org and requires no installations. After the IDE is launched the user must select the environment to use. There are three options to choose: Solid-ity, Vyper and Workshops. We used the first environment for our development. Solidity is the most popular language for writing Ethereum smart contracts. It is inspired by C++, Python and JavaScript. The documentation of the Solidity language is available at https://solidity.readthedocs.io/en/v0.5.13.

4.1.2 MetaMask

MetaMask is an Ethereum wallet in the browser. It allows its users to access Ethereum enabled distributed applications (dApps). It injects Web3 API (a JavaScript library for interacting with the Ethereum blockchain) into every website's JavaScript context to allow dApps to interact with the blockchain. It holds cryptographic key-pairs (public and private keys) that are used to securely perform transactions (read and write operations) on the blockchain. The step by step installation of MetaMask is provided on its official website https:// metamask.io/. Once MetaMask is installed, its logo appears in the top right corner of your browser. To open it click on the icon and log in with the password set during installation. Then, connect to the appropriate network by clicking on an item in the provided list. Note that it is possible to add new networks. In order to conduct tests on our smart contract we will mainly use Ropsten Test Network. The default account will show as Account 1. In order to customize your accounts, click on the avatar on the top right corner and follow the instructions for naming an account and adding new accounts. To each account corresponds a unique key-pair: a public key and a private key. The first 20 bytes of the KECCAK-256 of the

public key is also known as the address. The private key is used for signing transactions on the blockchain and must be kept secret. Should you need ETH from the Ropsten Test Network, select an account and click on the Deposit icon. Thereafter, click on Get Ether to request some test ETH from the MetaMask Ether Faucet. In order to log out of MetaMask, click on the avatar on the to right corner. A new window will open with the Log Out button on the top right corner.

4.1.3 OneClickDapp

OneClickDapp is a service for building a simple web interface for interacting with smart contracts. It allows you to instantly generate a unique URL for sharing your dApp. It is fast, free and open source. It requires the ABI (Application Binary Interface) of the contract and the address where the contract has been deployed. These artifacts may be obtained from the Remix IDE as follows. After setting up the appropriate environment (Solidity in our case), load your smart contract into the Remix IDE. Then, log into your MetaMask and select the account to be used for the underlying deployment. Next, compile your contract by clicking the compile button (the one with the Solidity logo). Next click on the Deploy and Run Transactions button and select as Evironment the option Injected Web3. Then, click on the Deploy Button (after possibly filling the appropriate constructor's parameters). After this, you will be prompted by MetaMask to confirm the transaction. When the transaction is confirmed your contract will show in Remix as deployed. Now, you are ready to generate a dApp from your smart contract. Go to https:// oneclickdapp.com/and click the Create dApp for free to fill the necessary information: address where the contract has been deployed (get it from Remix), ABI (get it from Remix) and the selection of the appropriate network (Ropsten in our case). Finally, click on the icon to generate your dApp which comes with a unique URL that can be shared.

4.2 Interacting with Our Smart Contract

We created a dApp from our Solidity smart contract by following the steps explained in the previous section. Here we explain how to use it step by step. There are two types of interaction that one can do on the blockchain through our contract: write and read. Write interactions or operations are those that modify the state of the contract and by ricochet that of the blockchain. They are expensive compare to read interactions or calls which consist in reading data from the blockchain through the contract. There are in total 15 (see Fig. 3) operations and 25 calls (see Fig. 4) . The GUI is available at https:// oneclickdapp.com/pony-basket/. For conversion from ASCII to bytes32, use the URL https:// eth-toolbox.com/?target=address&address=#bp-address

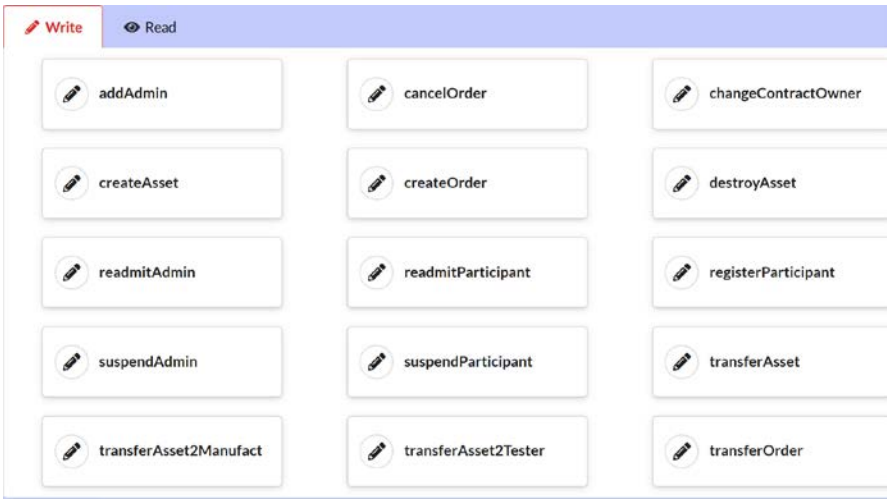


Figure 3: GUI for operations

Figure 4: GUI for calls

5.0 CONCLUSION

We have built a permissioned blockchain [1] powered by a Solidity [2] smart contract for tracking assets from fabrication to delivery. Our smart contract is generic enough such that we can used it for tracking any product on a pull supply chain. The inter-face of our blockchain was automatically generated using a third party service (https: //oneclickdapp.com/) for generating a distributed application (dApp) available at https: //oneclickdapp.com/pony-basket/. The main use case of our dApp is for tracking electronic components from fabrication to delivery and beyond. On our permissioned blockchain, assets' serial numbers are controlled internally and available to all participants in such a way that counterfeiting is almost impossible. Besides, if a counterfeit product is introduced, its origin can easily be traced since all transactions are immutable.

6.0 REFERENCES

[1] https://ethereum.org/

[2] https://solidity.readthedocs.io/en/v0.5.13/

[3] https://oneclickdapp.com/pony-basket/

[4] Mentzer, J.T., 2004. Fundamentals of supply chain management: twelve drivers of competitive advantage. Sage.

[5] Narayanan, A., Bonneau, J., Felten, E., Miller, A. and Goldfeder, S., 2016. Bitcoin and cryptocurrency technologies: a comprehensive introduction. Princeton University Press.

[6] Nakamoto, S., 2008. Bitcoin: A peer-to-peer electronic cash system.

[7] Bentov, I., Lee, C., Mizrahi, A. and Rosenfeld, M., 2014. Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake. IACR Cryptology ePrint Archive, 2014, p.452.

[8] https://github.com/slimcoin-project/slimcoin-project.github.io/raw/ master/whitepaperSLM.pdf

LIST OF ACRONYMS

ABI:	Application Binary Interface.
API:	Application Programming Interface.
ASCII:	American Standard Code for Information Interchange.
dApp:	distributed Application.
DFARS:	Defense Federal Acquisition Regulation Supplements.
DoD:	Department of Defense.
ETH:	Ether
FY:	Financial Year.
GUI:	Graphical User Interface.
IDE:	Integrated Development Environment.
NDAA:	National Defense Authorization Act.
SASC:	Senate Armed Services Committee.
SCM:	Supply Chain Management.
URL:	Uniform Resource Locator.
US/USA:	United States of America.
USAF:	United States Air Force.