Introducing Multi Shippers Mechanism for Decentralized Cash on Delivery System

Hai Trieu Le[1], Ngoc Tien Thanh Le[2], Nguyen Ngoc Phien[*,3], Nghia Duong-Trung[4], Ha Xuan Son[5], Thai Tam Huynh[6], and The Phuc Nguyen[7]

[1,2,4,5]Cantho University of Technology, Can Tho city, Vietnam
[*,3]Center for Applied Information Technology, Ton Duc Thang University, Ho Chi Minh city, Vietnam
[*,3]Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh city, Vietnam
[4,5]Can Tho University of Technology, FPT University, Can Tho city, Vietnam
[6]Transaction Technologies PTE. LTD., Singapore
[7]University of Trento, Trento, Italy

***Abstract*—One of the major problems of e-commerce globally is the selling and buying of goods among the parties over the Internet in which the traders may not trust their partners. Cash on delivery allows customers to pay in cash when the product is delivered to their home or a location they choose. This is sometimes called a payment system because customers receive goods before making a payment. This paper investigates a critical verification process issue in the cash on delivery system. In particular, we propose a multi shippers mechanism, which consists of blockchain technology, smart contracts and hyperledger fabric platform to achieve distributed and trustworthy verification across participants in the decentralized markets. Our proposed mechanism is given to not only ensure the benefits of the seller but also prevent shipper's fraudulent. The solution leverages the consistency and robustness of decentralized markets where trust is flexible and effectively controlled. To demonstrate the application and implementation of the proposed framework, we conduct several case studies on real-world transaction datasets from a local computer retailer. We also provide our sources codes for further reproducibility and development. Our conclusion is that the continued integration of multi-shipper mechanism and blockchain technology in the decentralized markets will cause significant transformations across several disciplines.**

***Keywords*—Blockchain; smart contract; Cash on Delivery (COD); hyperledger fabric**

I. Introduction

A regular delivery transaction includes buyers and sellers. The seller needs to transport the goods to the buyer, while the buyer gives payment to the seller. However, both sellers and buyers can cheat. Specifically, the seller is skeptical that the buyer will receive the goods without payment, and the buyer will doubt that the seller will receive the money and not deliver the goods. In international transactions, this situation is exacerbated by a slow and complicated transaction process. Ordinary goods must be transported long distances and must go through import and export procedures. The payment must end up with barriers similar to currency changes and legal regulations of each country.

Cash on Delivery (COD) allows customers to pay in cash when the product is delivered to their home or a location they choose. This is sometimes called a payment system because customers receive goods before making a payment. COD has become increasingly popular in recent years. However, most published documents about COD have appeared in reports or magazines or on the web, with a few scientific studies to date. Among research articles, most investigated payment methods in general, rather than focusing on COD in particular.

The commonly used transfer unit is postage, but usually, consumer and business shipments will be sent to COD by courier companies, commercial truck forwarders or organizations own delivery organizations. COD sales usually involve a fee charged by the shipping agent and is usually paid by the buyer. In retail and wholesale transactions, shipments are made on a COD when the buyer does not have a credit account with the seller and does not choose prepayment. The term is also often used when the small amount involved and the advance credit cost will be high in proportion to the size of the purchase.

The rest of the paper is organized as follow. Section 2 presents some related works. Section 3 briefly describes technical background. Section 4 presents our proposed COD transport process. Section 5-6 describes experimental results. Section 7 gives some conclusions.

II. Related Work

One of the major problems of e-commerce globally is the selling and buying of goods among the parties over the Internet in which the traders may not trust their partners. Krishnamachari *et. al.* [1] proposed the mechanism that executes a transaction with any kinds of assets by using the digital key and these processes do not need a trusted third-party. Additionally, the authors describe a transaction method which signs dual deposit for anti-fraud payment transactions and the delivery between two parties in which the trader can use the digital signature to verify. The seller and the buyer (customer) use a pair of symmetric keys to verify goods. They use smart contracts to decide and handle sellers and buyers by increasing deposits. But this paper has not yet analysis on a problem of shipping, if it is a physical product and the shipper fails to comply with the commitment, then the system is not resolved.

Hasan [2] proposes a delivery process in which participants (sellers, shippers and buyers) must mortgage an amount of

money. The mortgage is double the value of the goods shipped if the contract value will be returned to the parties. They provide a limited time solution to complete the contract. If the delivery time fails, the system will automatically resolve the dispute, based on the contract without the need for people.

Almost existing blockchain decentralized approaches still contain limitations that need to be improved to be effective and complete. Firstly, in [3], [4], [5] there is no incentive for any participating entities to act honestly. Sellers, buyers and carriers are fully believed. Secondly, [4], [5], [6], [7] depend on TTP or trusted arbitrator to act as a deposit and keep all the money from starting the sales process until the end. There is a TTP that holds money that can be considered as one focus point of failure. More, [2], [8] have no resolution mechanism dispute if any happens. Therefore, there will be a loss for the seller, buyer or both for any dishonest behavior.

Other researchers, Le *et. al.* [9] has proposed a mechanism based on the Ethereum Blockchain [10] or Son *et al.* has introduced a mechanism [11] based on Hyperledger Fabric flatform [12] that relates to product transportation between sellers and buyers. In their approach, the carrier plays an important role. In our article, it is recommended that the shipper join the system and mortgage a sum of money to ensure the reliability of the system. Our process is given to not only ensure the benefits of the seller but also prevent shipper's fraudulent. If the shipper has problems, such as loss of goods then the goods of the seller sent at the carrier will still be refunded in cash to the seller.

In this article, we review and summarize related work mentioning delivery solutions and technical implementation use blockchain technology. We also conducted decentralized surveys market based on blockchain. In addition, we use hyperledger fabric in the system to protect the rights of sellers.

III. Materials and Technical Background

A. Blockchain and Blockchain-based Smart Contracts

Blockchain is a list of developing logs, called blocks, linked by encryption. Each block contains the previous block's cryptographic hash function, timestamp, and transaction data. Each block has a block header and a body containing data and hash values of the previous block. The hash value is the result of a hash function. The hash function transforms data of any length into a fixed length string or numeric value, such as 256 bits (32 bytes) with SHA256. Blockchain is a technology that allows secure data transmission based on an extremely complex encryption system, similar to accounting books of a company where cash is closely monitored. In this case, the blockchain is an accounting ledger [13] that works in the digital field. A special feature of blockchain is that transactions are done at a high level of trust without disclosing information.

Blockchain-based smart contracts are proposed contracts that could be partially or fully executed without human interaction [14], [15], [16]. One of the main objectives of a smart contract is an automated escrow. An IMF (International Monetary Fund) staff discussion reported that smart contracts based on Blockchain technology might reduce moral hazards and optimize the use of contracts in general, but "no viable smart contract systems have yet emerged". Due to the lack of widespread use, their legal status is unclear [17]. Smart contract based on blockchain is being considered for many different types of transactions, from ubiquitous devices to real-time operational management structures for industrial products and data transfer in some applications including transaction finance. All types of business and management can participate in the network and use the properties of the Blockchain system to ensure transparency of stakeholders.

B. Smart Contracts

A cryptocurrency is a decentralized platform that a distributed ledger is used to interact with virtual money. A contract is an instance of a computer program that executes on the Blockchain. Users transfer money by publishing transactions and interacting with contracts in the cryptocurrency network where information is propagated, data is stored among miners or network's nodes. An underlying cryptocurrency system supports the utilization of smart contracts. A smart contract contains program code, a stored file and an account balance. Any user can submit a transaction to an append-able-only log. When the contracted is created, its program code cannot be changed. An append-able-only log, called a blockchain, which imposes a partial or total arrangement on submitted transactions is the main interface provided by the cryptocurrency. Fig. 1 presents the idea of a decentralized cryptocurrency system and its components.

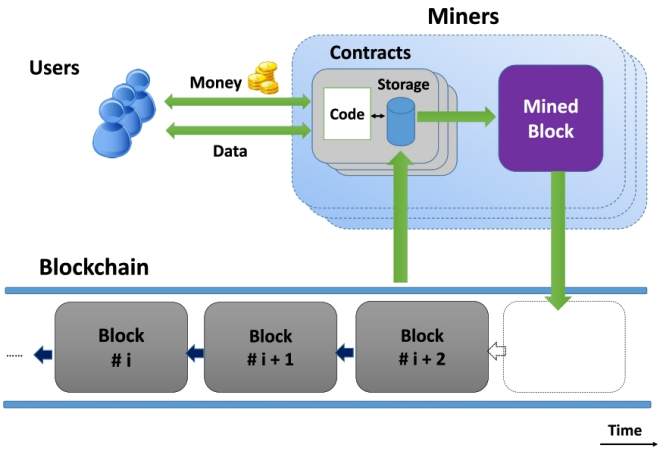

Fig. 1. An illustration of smart contracts and Blockchain in a decentralized cryptocurrency system [18].

C. Hyperledger Fabric

Hyperledger Fabric [19], [20], [21] is an open source distributed ledger platform, designed for developing permission application enterprise-grade. Fabric provides a platform to build instant, efficient and secure enterprise blockchain applications. Hyperledger Fabric is a platform for distributed ledger solutions underpinned by a modular architecture delivering high degrees of confidentiality, resiliency, flexibility, and scalability. It is designed to support pluggable implementations of different components and accommodate the complexity and intricacies that exist across the economic ecosystem.

IV. Proposed COD Transport Process

In this section, the authors introduce a general overview of the architecture with a highlight of the idea of multi-shipper. Then, we discuss the proposed architecture in more details. Finally, we present an important algorithm that serves as the backbone of our proposed architecture.

A. General COD Process

The authors start this session by presenting a general description of COD transport process. The general procedure is illustrated in Fig. 2. First, buyers create an order and send to the seller (2), meanwhile the order details are encrypted and sent to the delivery company which and the seller began to agree on time constraints and delivery prices (3). Before the order can be delivered, the shipper of a delivery company will carry out identity verification by an ID and a hash code order (4). If the first step finishes successfully, the shipper authentication process is notified and verified to the delivery company and the buyer (5). Next, the order's goods will be shipped via a shipper and/or various shippers. Each time the carrier is changed, the two shippers carry out the authentication by an ID and a hash code of the order. The authentication information is returned to allow the next shipper or notify whether the order is illegally intervened. If the authentication process is successful, the next shipper continues to ship the order to another shipper (if any) (6). Finally, the remaining shipper will authenticate the order with the buyer by encrypting the details of the order. In addition, the buyer must verify the identity by the ID to prove that he or she has ordered (7).

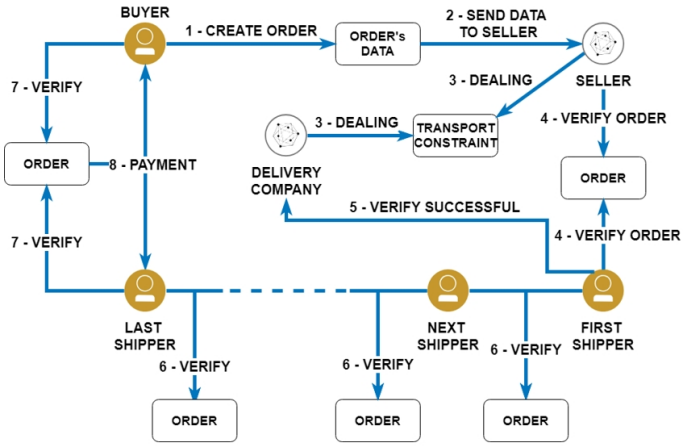

Fig. 2. General description of our proposed COD transport process.

B. Detailed COD Design

In this section, we describe how the authentication process between multi-shippers is done. The detailed procedure of multi-shippers authentication is illustrated in Fig. 3. The previous shipper will provide an ID and an asset hash code or each asset in the order (in the case of multiple asset orders) (1). The next shipper will enter the ID and the hash code into the system (2). Then, the system compares the hash code provided by the shipper with the hash code has been previously stored in the database (3). If the hash code does not match, the system will store the authentication information with the status "FAILED" (4) - (5.1) - (6). Consequently, the next shipper refuses such products (7) - (8.1). In case two hash codes match, the system will store authentication information with the status "SUCCESSFUL" (4) - (5.1) - (6). After that, the shipper continues to process the order delivery (7) - (8).

Algorithm 1 describes the authentication process between two shippers. First, the ID and hash of the asset are provided, the system checks the hash code stored in the database to make sure that it matches the hash code provided with the same ID of the asset. If the pair information does not match, the actual status will be immediately stored with "FAILED" state. Hence, the next shipper has the right to refuse the order's shipment. Otherwise, if the two hash codes match each other, the authentication information will be stored with the status "SUCCESSFUL", the next shipper accepts the assets and continues the order delivery process.

Algorithm 1 verifyShipper

```
for each asset given by the previous shipper do
    Get the asset to encrypt's data
    if The hash strings do not match then
        Store authentication information with "FAILED" status
        Next shipper refuses to get the asset
    else
        Store authentication information with "SUCCESSFUL" status
        Next shipper gets asset
        Next shipper transports asset to the buyers or to another shipper(s) on the chain if any
    end if
end for
```

V. Experimental Setup

A. Dataset Collection

The experimental datasets, see Table I, have been crawled from a local computer score that the authors use to demonstrate how the system works. There is one thing to note is that the price of products is in Vietnamese Dong (VND). The currency exchange rate when the paper is conducted is that 100 USD equals 2.300.000 VND.

B. Experimental Scenarios

1) Scenario 1: A buyer creates an order; successful verification between seller - shipper, shipper - shipper and shipper - buyer pairs.

Step 1: The buyer initiates the order information, the *createOrder()* function is called to perform this task and stores the information of the order into the distributed ledger. At this step, the *createAssetHash()* function is also called to encrypt the order and store the encrypted string into the database. An example of the *createOrder()* and *createAssetHash()* functions are presented in Tables II and III.

Step 2: Next, the process of verifying information between the seller and shipper begins. The *encryptAsset()* function is called to encrypt the commodity information. The encrypted string will be compared to the hash code and shipper holding

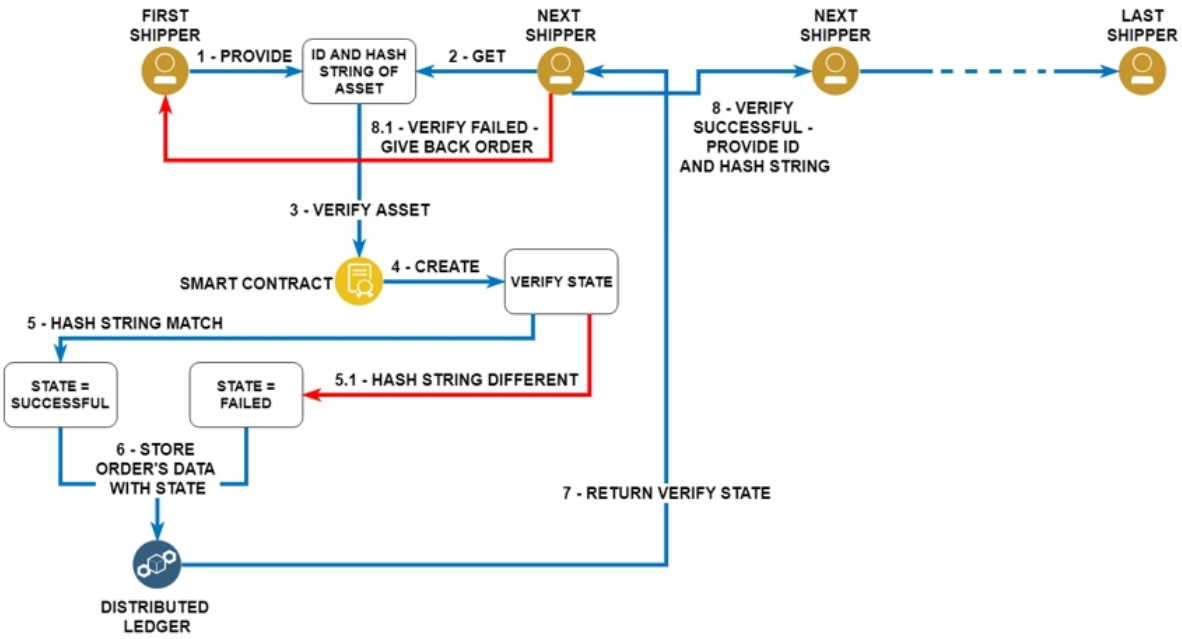

Fig. 3. Detailed design of our proposed COD transport process.

to confirm the order. The result of calling the *encryptAsset()* function is described in Table IV.

Step 3: After completing the authentication process with the seller, the shipper proceeds to authenticate each other during the shipment process. The *verifyshipper()* function is used to store the authentication information with "SUCCESSFUL" status from which the shipper can decide to continue shipping packages. The result of calling the *verifyShipper()* function is described in Table V.

Step 4: Delivery of the goods after being verified and shipped to the buyer. The shipper will eventually use the *encryptAsset()* function to encrypt the product information. The hash code after being generated coincides with the hash code that the buyer has. The authentication process is successful, and hence the buyer proceeds to pay for the delivered product. The result of calling the *encryptAsset()* function is presented in Table VI.

2) Scenario 2: A buyer creates an order; successful verification between seller - shipper pair; unsuccessful verification between shipper - shipper pair.

Step 1: The buyer initiates the order information, the *createOrder()* function is called to perform this task and stores the information of the order into the distributed ledger. At this step, the *createAssetHash()* function is also called to encrypt the order and store the encrypted string into the database. An example of the *createOrder()* and *createAssetHash()* functions are presented in Tables VII and VIII.

Step 2: Next, the process of verifying information between the seller and shipper begins. The *encryptAsset()* function is called to encrypt the commodity information. The encrypted string will be compared to the hash code and shipper holding to confirm the order. The result of calling the *encryptAsset()* function is described in Table IX.

Step 3: At this step, the shipper proceeds to authenticate orders together during exchanging the goods. The previous shipper will not be able to change the delivered items because this information is kept private. In addition, the shipper is completely unaware of the product inside the package. Any behaviors that deliberately change the details will cause the hash code to be generated because the last shipper's hash code is completely different from the hash code held by the buyer. The shipper authenticates the items with the ID and hash code of the product. If the hash codes do not match, the system will store authentication information with the "FAILED" status. The next shipper will refuse to accept the goods. The process of calling the *verifyShipper()* function is presented in Table X.

3) Scenario 3: A buyer creates an order; unsuccessful verification between seller - shipper pair.

Step 1: The buyer initiates the order information, the *createOrder()* function is called to perform this task and stores the information of the order into the distributed ledger. At this step, the *createAssetHash()* function is also called to encrypt the order and store the encrypted string into the database. An example of the *createOrder()* and *createAssetHash()* functions are presented in Tables XI and XII.

Step 2: If the seller deliberately changes the product structure after the buyer has ordered, the hash code generated during the authentication period between the shipper and the seller will be different from the hash code that shipper is keeping (note that the seller does not know this hash code before). Since then, the shipper can determine that the product code has been changed by the seller and has the right to refuse to receive the package. The authentication information is also stored with "FAILED" status. The process of calling the *encryptAsset()* function is presented in Table XIII.

TABLE I. EXPERIMENTAL DATASET TAKEN FROM A LOCAL RETAIL STORE.

No.	Product name	Product details	Price (in VND)
1	AsusX441MAN5000 (GA024T)	display:14inch,HD; CPU:PentiumN5000,1.10GHz; RAM:4GB; HDD:1TB; graphics:IntelHDGraphics; OS:Windows10; DVD:no	7.190.000
2	AsusVivoBookX441MAN5000 (GA004T)	display:14inch,HD; CPU:PentiumN5000,1.10GHz; RAM:4GB; HDD:500GB; graphics:IntelHDGraphics; OS:Windows10; DVD:no	7.490.000
3	AsusVivoBookX407UAi36006U (BV129T)	display:14inch,HD; CPU:i3-6006U,2.0GHz; RAM:4GB; HDD:1TB; graphics:IntelHDGraphics; OS:Windows10; DVD:no	9.690.000
4	AsusX407UAi37020U (BV351T)	display:14inch,HD; CPU:i3-7020U,2.3GHz; RAM:4GB; HDD:1TB; graphics:IntelHDGraphics; OS:Windows10; DVD:no	10.090.000
5	AsusX507UAi37020U (EJ727T)	display:15.6inch,FullHD; CPU:i37020U,2.3GHz; RAM:4GB; HDD:1TB; graphics:IntelHDGraphics620; OS:Windows10,DVD:no	10.590.000
6	AsusVivoBookX540UBi36006U (DM024T)	display:15.6inch,FullHD; CPU:i3-6006U,2.0GHz; RAM:4GB; HDD:1TB; graphics:NVIDIAMX110,2GB; OS:Windows10; DVD:support	10.790.000
7	AsusVivoBookA411UAi38130U (EB688T)	display:14inch,FHD; CPU:i3-8130U,2.2GHz; RAM:4GB; HDD:1TB; graphics:IntelUHDGraphic620; OS:Windows10; DVD:no	11.290.000
8	AsusA510UAi38130U (BR333T)	display:15.6inch,HD; CPU:i3-8130U,2.2GHz; RAM:4GB; HDD:1TB; graphics:IntelUHDGraphics620; OS:Windows10; DVD:no	11.390.000
9	AsusVivoBookX507UFi38130U (BR203T)	display:15.6inch,HD; CPU:Corei3CoffeeLake,2.20GHz; RAM:4GB; HDD:1TBSATA3; graphics:NVIDIAMX130,2GB; OS:Windows10HomeSL; DVD:no	11.990.000
10	AsusVivoBookS15S510UAi38130U (BQ222T)	display:15.6inch,FHD; CPU:i38130U,2.2GHz; RAM:4GB; HDD:1TB; graphics:IntelHDGraphics620; OS:Windows10; DVD:no	12.890.000
11	AsusA411UAi58250U (EB678T)	display:15.6inch,FHD; CPU:i38130U,2.2GHz; RAM:4GB; HDD:1TB; graphics:IntelHDGraphics620; OS:Windows10; DVD:no	13.490.000
12	AsusX407UAi58250U (BV485T)	display:14inch,HD; CPU:Corei5CoffeeLake,1.60GHz; RAM:4GB; HDD:1TB; Optane16GB, graphics:IntelHDGraphics620; OS:Windows10Home	13.690.000
13	AsusA510UAi58250U (EJ1215T)	display:15.6inch,FullHD, CPU:Corei5CoffeeLake,1.60GHz; RAM:4GB; HDD:1TB; graphics:IntelUHDGraphics620; OS:Windows10HomeSL	13.790.000
14	AsusX507UFi58250U4GB1TB (EJ121T)	display:15.6inch,FullHD; CPU:Corei5KabylakeRefresh,1.60GHz; RAM:4GB; HDD:1TBSATA3,slotsupportSSDM.2; graphics:NVIDIAMX130,2GB; OS:Windows10HomeSL	14.590.000
15	AsusVivoBookS15S510UAi58250U (BQ414T)	display:15.6inch,FullHD; CPU:i5KabylakeRefresh,1.60GHz; RAM:4GB; HDD:1TB; graphics:UHDGraphics620; OS:Windows10; DVD:no	15.490.000
16	AsusVivoBookS510UNi58250U (BQ276T)	display:15.6inch,FullHD; CPU:Corei5CoffeeLake,1.60GHz; RAM:4GB; HDD:1TBSATA3; graphics:NVIDIAMX150,2GB; OS:Windows10HomeSL; DVD:no	16.790.000
17	AsusVivobookS15S530UAi58250U (BQ290T)	display:15.6inch,FHD; CPU:i5-8250U,1.6GHz; RAM:4GB; HDD:1TB; graphics:IntelUHDGraphics620; OS:Windows10	17.390.000
18	AsusZenBookUX433FAi58265U (A6061T)	display:14inch,FullHD; CPU:Corei5CoffeeLake,1.60GHz; RAM:8GB; SSD256GBNVMePCIe; graphics:IntelUHDGraphics620; OS:Windows10Home	22.990.000
19	AsusFX504GEi58300H (E4138T)	display:15.6inch,FullHD; CPU:Corei5CoffeeLake,2.30GHz; RAM:8GB; HDD:1TB, supportSSDM.2PCIe; graphics:NVIDIAGeForceGTX1050Ti,4GB; OS:Windows10HomeSL	23.490.000
20	AsusFX505GEi78750H (BQ037T)	display:15.6inch,FullHD; CPU:Corei7CoffeeLake,2.20GHz; RAM:8GB; SSD128GBM2PCIe,HDD:1TBSATA3; graphics:NVIDIAGeForceGTX1050Ti, 4GB; OS:Windows10Home	27.990.000

TABLE II. THE CREATEORDER() FUNCTION: A BUYER CREATES AN ORDER AND THE SYSTEM CREATES ORDER'S DATA.

orderID	buyerID	sellerID	deliverID	Product name	Quantity	Price (in VND)	Status	Execution time
order001	customer001	seller001	delivery001	AsusX441MAN5000(GA024T)	1	7.190.000	waiting	1.008266524s
order002	customer002	seller002	delivery002	AsusVivoBookX441MAN5000(GA004T)	1	7.490.000	waiting	1.003088932s
order003	customer003	seller003	delivery003	AsusVivoBookX407UAi36006U(BV129T)	1	9.690.000	waiting	1.006305223s
order004	customer004	seller004	delivery004	AsusX407UAi37020U(BV351T)	1	10.090.000	waiting	1.003237888s
order005	customer005	seller005	delivery005	AsusX507UAi37020U(EJ727T)	1	10.590.000	waiting	1.002595092s

TABLE III. CREATEASSETHASH(): THE SYSTEM CREATES ORDER'S HASH.

orderID	sellerID	Product name	Quantity	Price (in VND)	Execution time
order001	seller001	AsusX441MAN5000(GA024T)	1	7.190.000	1.003257091s
order002	seller002	AsusVivoBookX441MAN5000(GA004T)	1	7.490.000	1.002689831s
order003	seller003	AsusVivoBookX407UAi36006U(BV129T)	1	9.690.000	1.002950846s
order004	seller004	AsusX407UAi37020U(BV351T)	1	10.090.000	1.003001433s
order005	seller005	AsusX507UAi37020U(EJ727T)	1	10.590.000	1.002540695s

VI. REMARKS

The implementation of our approach is deployed on Ubuntu 19.04 machine with 2.53GHz CPU and 8GB of RAM. Tables II, III, IV, V, and VI show the performance of functions in the Cash on Delivery system, in which the input data and

TABLE IV. ENCRYPTASSET(): THE SELLER ENCRYPTS ASSET TO VERIFY ORDER TO THE SHIPPER.

sellerID	Product name	Quantity	Price (in VND)	Execution time
seller001	AsusX441MAN5000(GA024T)	1	7.190.000	1.000282933s
seller002	AsusVivoBookX441MAN5000(GA004T)	1	7.490.000	1.000271937s
seller003	AsusVivoBookX407UAi36006U(BV129T)	1	9.690.000	1.000289196s
seller004	AsusX407UAi37020U(BV351T)	1	10.090.000	1.000244656s
seller005	AsusX507UAi37020U(EJ727T)	1	10.590.000	1.000368499s

TABLE V. VERIFYSHIPPER(): THE SHIPPER ENCRYPTS ORDER'S DATA TO THE NEXT SHIPPER.

orderID	Hash	Location	Execution time
order001	bdd736bba7e3b336d5bed8b08fe503c7afeb1b6e621b1a439566ef6e46b96e30	cityAlpha	1.024215124s
order002	5f2fe095759f31b2d91819f8df0602601758394d287fb63d92effa84e6679a02	cityAlpha	1.023620744s
order003	d873b49bc5aeee5ee1d4e7cffff02aca09a91c739a72867e77eb70e5a8403499	cityAlpha	1.021422615s
order004	21fbd6ee92b9ca5446409690136f65493572863fe8eea836df424d3328e0a1ac	cityAlpha	1.018266805s
order005	00631c0bdd93ec1014bbf7290d0e43fa189c94b71af5ecef7ea187a548c2fc74	cityAlpha	1.016574424s

TABLE VI. ENCRYPTASSET(): THE SHIPPER ENCRYPTS ORDER'S DATA TO VERIFY TO THE BUYER.

sellerID	Product name	Quantity	Price (in VND)	Execution time
seller001	AsusX441MAN5000(GA024T)	1	7.190.000	1.000329441s
seller002	AsusVivoBookX441MAN5000(GA004T)	1	7.490.000	1.000292817s
seller003	AsusVivoBookX407UAi36006U(BV129T)	1	9.690.000	1.000285833s
seller004	AsusX407UAi37020U(BV351T)	1	10.090.000	1.00029968s
seller005	AsusX507UAi37020U(EJ727T)	1	10.590.000	1.000230017s

TABLE VII. CREATEORDER(): THE BUYER CREATES ORDER AND THE SYSTEM CREATES ORDER'S DATA.

orderID	buyerID	sellerID	deliverID	Product name	Quantity	Price (in VND)	Status	Execution time
order006	customer006	seller006	delivery006	AsusVivoBookX540UBi36006U(DM024T)	1	10.790.000	waiting	1.002387767s
order007	customer007	seller007	delivery007	AsusVivoBookA411UAi38130U(EB688T)	1	11.290.000	waiting	1.002605573s
order008	customer008	seller008	delivery008	AsusA510UAi38130U(BR333T)	1	11.390.000	waiting	1.002959993s
order009	customer009	seller009	delivery009	AsusVivoBookX507UFi38130U(BR203T)	1	11.990.000	waiting	1.002753797s
order010	customer010	seller0010	delivery010	AsusVivoBookS15S510UAi38130U(BQ222T)	1	12.890.000	waiting	1.002916293s

TABLE VIII. CREATEASSETHASH(): THE SYSTEM CREATES ORDER'S HASH.

orderID	sellerID	Product name	Quantity	Price (in VND)	Execution time
order006	seller006	AsusVivoBookX540UBi36006U(DM024T)	1	10.790.000	1.002827145s
order007	seller007	AsusVivoBookA411UAi38130U(EB688T)	1	11.290.000	1.003145603s
order008	seller008	AsusA510UAi38130U(BR333T)	1	11.390.000	1.003106346s
order009	seller009	AsusVivoBookX507UFi38130U(BR203T)	1	11.990.000	1.00231202s
order010	seller0010	AsusVivoBookS15S510UAi38130U(BQ222T)	1	12.890.000	1.002822286s

TABLE IX. ENCRYPTASSET(): THE SELLER ENCRYPTS ASSET TO VERIFY ORDER TO THE SHIPPER.

sellerID	Product name	Quantity	Price (in VND)	Execution time
seller006	AsusVivoBookX540UBi36006U(DM024T)	1	10.790.000	1.000223568s
seller007	AsusVivoBookA411UAi38130U(EB688T)	1	11.290.000	1.000244662s
seller008	AsusA510UAi38130U(BR333T)	1	11.390.000	1.000268549s
seller009	AsusVivoBookX507UFi38130U(BR203T)	1	11.990.000	1.000184843s
seller0010	AsusVivoBookS15S510UAi38130U(BQ222T)	1	12.890.000	1.000358836s

TABLE X. VERIFYSHIPPER(): THE SHIPPER ENCRYPTS ORDER'S DATA TO VERIFY TO THE NEXT SHIPPER.

orderID	hash	Location	Execution time
order006	c40e97a57e66ed09388476a09571bfd513df6f2fd72dfc57b473fd5af2097a00 - a	cityAlpha	1.015158103s
order007	495f2a8e0e61732764c8b03e832cf0735bb07551b6388b89d2c20474e0ba094f - b	cityAlpha	1.013581844s
order008	b85c9b9c7772b17b2c5641a75dc19bd448d386afb5dbdeaca8fbbddebd3ca468 - c	cityAlpha	1.019617637s
order009	29e869390120de9cda8bfd910701225a1ba0de37b9f5b77d5cf7fb1284570d48 - d	cityAlpha	1.013978084s
order010	08b640f9f41e83ce466de42251d19f80253b9af854688375f374f67ab7fa0581 - e	cityAlpha	1.012554071s

TABLE XI. CREATEORDER(): THE BUYER CREATES ORDER AND THE SYSTEM CREATES ORDER'S DATA.

orderID	buyerID	sellerID	deliverID	Product name	Quantity	Price (in VND)	Status	Execution time
order011	customer011	seller011	delivery011	AsusA411UAi58250U(EB678T)	1	13.490.000	waiting	1.003676700s
order012	customer012	seller012	delivery012	AsusX407UAi58250U(BV485T)	1	13.690.000	waiting	1.001928328s
order013	customer013	seller013	delivery013	AsusA510UAi58250U(EJ1215T)	1	13.790.000	waiting	1.001924617s
order014	customer014	seller014	delivery014	AsusX507UFi58250U4GB1TB(EJ121T)	1	14.590.000	waiting	1.002541236s
order015	customer015	seller015	delivery015	AsusVivoBookS15S510UAi58250U(BQ414T)	1	15.490.000	waiting	1.002492703s

TABLE XII. CREATEASSETHASH(): THE SYSTEM CREATES ORDER'S HASH.

orderID	sellerID	Product name	Quantity	Price (in VND)	Execution time
order001	seller011	AsusA411UAi58250U(EB678T)	1	13.490.000	1.002169191s
order002	seller012	AsusX407UAi58250U(BV485T)	1	13.690.000	1.003219684s
order003	seller013	AsusA510UAi58250U(EJ1215T)	1	13.790.000	1.003060215s
order004	seller014	AsusX507UFi58250U4GB1TB(EJ121T)	1	14.590.000	1.003002701s
order005	seller015	AsusVivoBookS15S510UAi58250U(BQ414T)	1	15.490.000	1.002834664s

TABLE XIII. ENCRYPTASSET(): THE SELLER ENCRYPTS ASSET TO VERIFY ORDER TO SHIPPER.

sellerID	Product name	Quantity	Price (in VND)	Execution time
seller011	AsusA411UAi58250U(EB678T)	1	13.490.000	1.000232470s
seller012	AsusX407UAi58250U(BV485T)	1	13.690.000	1.000262189s
seller013	AsusA510UAi58250U(EJ1215T)	1	13.790.000	1.000224077s
seller014	AsusX507UFi58250U4GB1TB(EJ121T)	1	14.590.000	1.000301122s
seller015	AsusVivoBookS15S510UAi58250U(BQ414T)	1	15.490.000	1.000283559s

execution time of the two main functions of the system (Verify and Encrypt) that describe in Tables IV, V, and VI. According to the measured data in the tables, we conclude that our method does not require high-configuration equipment for deploying but it still ensures system performance (approximately 1 second/function). In addition, the level of complexity algorithm *verifyShipper()* is *n* where n is the number of products in a package and that of *encryptAsset()* is 1, this function does not depend on the input data. Moreover, our proposed approach supports the decentralized architecture based on Blockchain, users do not need cryptocurrency units to execute transactions (such as Ethereum systems) that significantly reduce risks such as vector attack. The peer on the network verify the request by identity information of the users and it prevents the act of installing/launching malicious smart contracts to query illegal data. On the other hands, our proposal mechanism does not require complex encryption or authentication algorithms whenever verifying the information of the user, thus saving more than non-distributed Blockchain-based systems. Hence it easily deployed in an enterprise environment. We encourage further reproducibility and implementation by providing our sources codes freely accessed on our Github repository[1].

VII. CONCLUSION

As we have demonstrated, the introduction of multi-shipper mechanism applied in any cash on delivery systems is very beneficial. Our process is given to not only ensure the benefits of the seller but also prevent shipper's fraudulent. We have provided a transparent verification that works across participants. Several case studies have demonstrated the feasibility of the proposed mechanism in achieving trustworthy and transparent verification for the COD systems. The solution leverages the consistency and robustness of decentralized markets where trust is flexible and effectively controlled. To the best of our knowledge, there have not been any research papers that exploit and implement a mechanism of multi-shipper in the COD system. We believe that the continued integration of multi-shipper mechanism and blockchain technology in the decentralized markets will cause significant transformations across several disciplines, bringing about new business applications and having us reconsider how the existing COD systems are developed.

[1]https://github.com/xuansonha17031991/CashOnDelevery-Chaincode

REFERENCES

[1] A. Asgaonkar and B. Krishnamachari, "Solving the buyer and seller's dilemma: A dual-deposit escrow smart contract for provably cheat-proof delivery and payment for a digital good without a trusted mediator," *arXiv preprint arXiv:1806.08379*, 2018.

[2] H. R. Hasan and K. Salah, "Blockchain-based solution for proof of delivery of physical assets," in *International Conference on Blockchain*. Springer, 2018, pp. 139–152.

[3] "Two party contracts," Feb 2015. [Online]. Available: https://dappsforbeginners.wordpress.com/tutorials/two-party-contracts/

[4] "How our escrow smart contract works," Oct 2017. [Online]. Available: https://blog.localethereum.com/how-our-escrow-smart-contract-works/

[5] "Sites like ebay or etsy but decentralized - our features." [Online]. Available: https://openbazaar.org/features/

[6] [Online]. Available: https://www.syscoin.org/home.html

[7] J. Sidhu, "Syscoin: A peer-to-peer electronic cash system with blockchain-based services for e-business," in *2017 26th international conference on computer communication and networks (ICCCN)*. IEEE, 2017, pp. 1–6.

[8] "Double deposit escrow." [Online]. Available: https://bitbay.market/double-deposit-escrow

[9] N. T. T. Le, Q. N. Nguyen, N. N. Phien, N. Duong-Trung, T. T. Huynh, T. P. Nguyen, and H. X. Son, "Assuring non-fraudulent transactions in cash on delivery by introducing double smart contracts," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 5, 2019. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2019.0100584

[10] "Ethereum project." [Online]. Available: https://ethereum.org/

[11] H. X. Son, M. H. Nguyen, N. N. Phien, H. T. Le, Q. N. Nguyen, V. D. Dinh, P. T. Tru, and P. Nguyen, "Towards a mechanism for protecting seller's interest of cash on delivery by using smart contract in hyperledger," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 4, 2019. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2019.0100405

[12] "Hyperledger fabric." [Online]. Available: https://hyperledger-fabric.readthedocs.io/

[13] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[14] T. Hamid, "Cash on delivery the biggest obstacle to e-commerce in uae and region," May 2014. [Online]. Available: https://www.thenational.ae/business/technology/cash-on-delivery-the-biggest-obstacle-to-e-commerce-in-uae-and-region-1.604383

[15] F. Idelberger, G. Governatori, R. Riveret, and G. Sartor, "Evaluation of logic-based smart contracts for blockchain systems," in *International Symposium on Rules and Rule Markup Languages for the Semantic Web*. Springer, 2016, pp. 167–183.

[16] M. Alharby and A. van Moorsel, "Blockchain-based smart contracts: A systematic mapping study," *arXiv preprint arXiv:1710.06372*, 2017.

[17] D. He, K. F. Habermeier, R. B. Leckow, V. Haksar, Y. Almeida, M. Kashima, N. Kyriakos-Saad, H. Oura, T. S. Sedik, N. Stetsenko *et al.*, "Virtual currencies and beyond: initial considerations," 2016.

[18] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 79–94.

[19] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 30.

[20] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2018, pp. 264–276.

[21] F. Benhamouda, S. Halevi, and T. T. Halevi, "Supporting private data on hyperledger fabric with secure multiparty computation," *IBM Journal of Research and Development*, 2019.