Hindawi
Security and Communication Networks
Volume 2021, Article ID 8884478, 18 pages
https://doi.org/10.1155/2021/8884478

*Research Article*

# SESCF: A Secure and Efficient Supply Chain Framework via Blockchain-Based Smart Contracts

**Menghui Lou**,[1] **Xiaolei Dong**,[1] **Zhenfu Cao**,[1,2] **and Jiachen Shen**[1]

[1]*Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China*
[2]*Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen and Shanghai Institute of Intelligent Science and Technology, Tongji University, Shanghai, China*

Correspondence should be addressed to Xiaolei Dong; dongxiaolei@sei.ecnu.edu.cn and Jiachen Shen; jcshen@sei.ecnu.edu.cn

Received 25 September 2020; Revised 7 December 2020; Accepted 17 December 2020; Published 15 January 2021

Academic Editor: Debiao He



The demands for the fairness, security, and efficiency of the supply chain have grown significantly due to the rise of globalization. However, some problems of the information flow, logistics, and capital flow in the supply chain remain a challenge, such as the information asymmetry between upstream and downstream, substandard quality of goods, difficulty in traceability, and default of payment. Therefore, this paper proposes a blockchain-based supply chain framework (SESCF), which solves the supply chain problems securely and efficiently. First, the use of blockchain and smart contracts ensures the information symmetry in the supply chain system. Second, the radio frequency identification (RFID) provides a unique identity of goods, which helps in real-time quality monitoring. Additionally, the immutability and distributed storage of the blockchain play an important role in tracking the origin of goods. Third, the efficient payment channel is used to solve the problem of payment defaults. Furthermore, simulations of smart contracts along with the security analyses are presented in this paper. We also implement a blockchain-based supply chain system (*SescfDapp*), which is built upon a Consortium blockchain. Large-scale experiments and detailed analysis prove the feasibility and efficiency of our proposed system.

## 1. Introduction

In recent years, with the deepening of the global division of labor, the supply chain of modern enterprises has been continuously extended, resulting in the fragmentation, complexity, geographical dispersion, and other characteristics, which has brought great challenges to supply chain management [1–3]. The supply chain's operations not only affect the core enterprise's interests but also affect the whole supply chain and the relevant enterprises' interests, which makes it increasingly important to propose a fair, secure, and efficient supply chain system [4–6]. Information flow, logistics, and capital flow are the three lifebloods of the supply chain. Information flow directs logistics, and logistics drives capital flow. However, in actual work, situations such as inadequate information flow, low logistics efficiency, and constant capital flow often occur. First, due to asymmetric and delayed information in the upstream and downstream, it is difficult for enterprises to control the flow of goods in real time. There is also the risk that information will be tampered to cause fraud, which will lead to a sharp increase in costs and a significant decrease in efficiency. Additionally, the modern supply chain encompasses all aspects of the production and distribution of goods, from raw materials to finished products delivered to consumers. However, the supply chain involves a vast range and weak supervision which has led buyers to lack a reliable way to verify the authenticity of goods. Therefore, the global counterfeit market is hugely flooded. A report released by the European Union states that the global market value of counterfeit trade is as high as more than 400 billion US dollars. About 5% of all goods imported into EU countries are fake and shoddy products over $100 billion [7], and monitoring the development of goods and efficient logistics management in the goods supply chain is critical to ensure goods quality. Meanwhile, the existing supply chain management systems

have weak constraints and on-chain enterprises have a large operating space, resulting in the final cash settlement between the supplier and the buyer being highly dependent on the contractual spirit of the two parties, especially when multiparty transactions and multilevel transactions are involved. The above issues become more challenging when there is a lack of transparency and mutual trust in the current supply chain systems.

Blockchain technology is a feasible way to solve the above issues; in other words, blockchain and supply chain are a natural pair [8–10]. First of all, the structure of the blockchain is a kind of time-series data that can store information, which is similar to the form of goods circulation in the supply chain. Secondly, the relatively low frequency of information updates in the supply chain avoids the shortcomings of current blockchain technology in terms of processing performance. Every transaction information on the blockchain, such as transaction parties, transaction time, and transaction content, will be recorded on a block and stored on the distributed ledger of each node on the chain, which ensures the integrity, reliability, and high transparency of the information. According to [11], for a variety of use cases in capital markets, distributed ledger technology (DLT) has the potential to reduce transaction latency, operational risk, process friction, liquidity requirements, and more. As mentioned in the 2018 China's Blockchain Industry White Paper [12], blockchain technology has begun to be implemented in many areas of the real economy. The report introduces blockchain applications in the real economy from areas including goods traceability, copyright, and supply chain application scenarios. However, there are still many significant challenges when implementing blockchain-based systems in the real world.

It is very challenging to propose an efficient system to ensure the fairness and security of the information flow, logistics, and capital flow of the supply chain at the same time, as the supply chain is fragmented, complicated, and geographically dispersed. Additionally, the feasibility and implementation of such a system remain to be solved. Therefore, in this paper, we propose a blockchain-based framework (SESCF) to ensure the fairness and security of the supply chain system efficiently. First, the use of blockchain and smart contracts in the supply chain enables information to be disclosed between upstream and downstream enterprises, since blockchain is a distributed ledger, that is, the information on the blockchain is recorded and shared by various participants at the same time. Besides, the radio frequency identification (RFID) provides a unique identity of each goods, which helps to monitor quality and trace the origin of the goods in real time [13, 14]. Simultaneously, we take advantage of the payment channel to solve the problem of payment defaults [15–18]. Here, SESCF allows the two parties in the transaction to put a certain amount of coins into the payment channel. Once a default or timeout occurs, the smart contract will be triggered to ensure the security of the coins of both parties. In our design, blockchain and smart contracts enable the verification of information, goods, and money, while they travel in the supply chain. We assume that honest users run a bug-free system, and smart contracts must be executed correctly. The contributions can be summarized as follows:

(1) As far as we know, this is the first efficient supply chain framework based on blockchain, which guarantees the fairness and security of the information flow, logistics, and capital flow. Meanwhile, the corresponding smart contracts are given in this paper.

(2) We have improved the efficiency of the supply chain by separating goods transactions and capital transactions, specifically, placing goods transactions on the chain and, conversely, putting capital transactions off the chain by opening payment channels.

(3) We run smart contracts in a simulation environment and evaluate the gas cost on the Ethereum network, and the experimental results proved the feasibility of our proposed solution. On this basis, we establish an efficient blockchain-based system *SescfDapp*. Furthermore, we also conduct a detailed vulnerability assessment of SESCF and discuss how the system remains secure against malicious attacks.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 introduces the background techniques and related knowledge. Section 4 proposes a secure and efficient blockchain-based framework. Section 5 describes the implementation details. Section 6 shows the experimental results of our proposed system. Section 7 analyses the security of smart contracts in this work. Finally, Section 8 concludes this paper and proposes directions for future work.

## 2. Related Work

The blockchain was born in the Bitcoin system [19], and now more and more industries are proposing their blockchain solutions. Azaria et al. [20] proposed a decentralized system MedRec that uses blockchain processing EMRs. Zhang and Deng [21] proposed a secure billing protocol during taxi rides. Passengers and drivers use publicly verifiable two-party blockchains to agree on taxi ride fees. Biswas and Muthukkumarasamy [22] applied blockchain to smart cities and integrated blockchain with smart devices to provide a secure communication platform for smart cities.

Furthermore, blockchain has been introduced in supply chain areas to make the chain more transparent, authentic, and trustworthy. A lot of research has been done to ensure the security and integrity of the supply chain [23, 24]. A shared, consensus-based, and immutable ledger helps track the origin and transformations undergone in the supply chain. The blockchain will create a formal registry to track the possession of goods throughout the supply chain.

In the current research work, most researchers are committed to using blockchain technology and Internet of Things (IoT) devices to achieve transparency and traceability in the supply chain [4, 14, 25–30]. Bocek et al. [30] proposed that blockchain can be used in various physical fields and took the pharmaceutical supply chain as an example to

introduce in detail how to use blockchain in the supply chain. To achieve fair transactions between producers and suppliers, Alahmadi and Lin [4] proposed a blockchain-based supply chain management system, which uses tamper-proof features of blockchain to make fair trade of goods. In [28], Marc Pilkington proposed that blockchain provides a groundbreaking method that can recreate transparency and establish new relationships with consumer products. To overcome the excessive addition of preservatives and harmful chemicals in the wine production process, Biswas et al. [26] proposed a blockchain-based wine supply chain traceability system to track counterfeit wine in the supply chain. In particular, there are many studies [14, 25, 27, 29] combined with the development of RFID and blockchain technology to establish a supply chain traceability system. Tian [25] proposed a novel agro-food supply chain traceability system and assessed the advantages and disadvantages of the system in the article. Toyoda et al. [27] proposed a product ownership management system (POMS) that attaches RFID to goods, which can be utilized to trace counterfeit products in the post supply chain system. Cui et al. [29] implemented a traceability system for the electronic parts supply chain based on the blockchain, tracing each chip in a nondestructive manner. Mondal et al. [14] proposed a blockchain-based food supply chain traceability system, combined with RFID and blockchain to build a complete system architecture to ensure the transparency of food in the supply chain system.

## 3. Background Techniques

In this section, we first introduce the technologies used in this paper, which contain blockchain, smart contract, RFID, and payment channel. Besides, we give brief analyses on how these technologies are used in SESCF and how effective they are.

*3.1. Blockchain.* Blockchain is a new application model of computer technology, such as distributed data storage, point-to-point transmission, consensus mechanisms, and encryption algorithms [9, 31]. In short, it is a list of records that are linked by cryptographic techniques. Unlike centralized databases, it is a decentralized transaction and data management technology and is in an ideal state for a low trust (or trustless) exchange system, that is, information in this system does not rely on the third party. Moreover, it utilizes peer-to-peer networks to perform peer-to-peer verification on items and distributes transaction details in the ledger. As the underlying technology of Bitcoin, blockchain has been a hot topic in the field of security and finance.

According to the degree of network centralization, blockchain is divided into three types: (i) Public blockchain. People around the world can read data, send transactions, and "mine" in Public blockchain. Thereby, Public blockchain is also considered to be a completely decentralized existence, as no organization or institution is able to control it. As we know, Bitcoin and Ethereum are typical Public blockchain [19, 32]. (ii) Consortium blockchain: several organizations or institutions jointly participate in management in Consortium blockchain. It should be noted that, unlike the Public blockchain, only organizations in it have permission to access data. (iii) Private blockchain: an organization or institution controls the write right in Private blockchain, and nodes participating in it will be strictly restricted.

In this paper, we implement an evaluation system for SESCF, which is built upon a Consortium blockchain. If a node wants to join SESCF, it needs to apply to Certification Authority (CA). After the node passes the CA verification successfully, it can join SESCF. Each node in SESCF has the power to compete for mining, and a certification composed of multiple entities can effectively prevent problems such as illegal transactions. If a transaction entity attempts to tamper with transaction records alone or with other transaction entities, other entities may prove their misconduct based on their transaction records and remove them from SESCF.

*3.2. Smart Contract.* Smart contract, as one of the Turing-complete programming languages, was first proposed by Nick Szabo in 1994 [31]. It is digital, and the protocol is enforced using encrypted code. In other words, smart contract is a computer protocol designed to facilitate, verify, and enforce negotiation digitally. Therefore, smart contract can perform reliable transactions without third parties, and these transactions are trackable and irreversible. In addition, smart contract cannot be tampered with because it has been stored and recorded on the blockchain. If anyone wants to tamper with it, he must tamper with the entire blockchain's ledger, which is costly.

Usually, an ordinary contract hinders the development of the supply chain because it takes much labour. What is more, due to the supply chain being highly open, theft, loss, and fraud are quite common. Fortunately, smart contract overcomes these shortcomings by providing parties with secure, transparent digital versions. Furthermore, it can automate tasks and transactions and even restrict behavior based on rules stored in code.

*3.3. Radio Frequency Identification.* Internet of Things (IoT), or "Internet of Everything," is a huge network formed by combining various information sensing devices with the Internet. There are some IoT-based tracking and tracing infrastructures, such as electronic article surveillance (EAS), radio frequency identification (RFID), and QR codes, which are primarily targeted for automatic package-level tracking [33]. Among the IoT devices mentioned above, RFID is low load and small, so we use RFID to solve the problems in supply chain logistics [13, 17, 34–38]. Sensor data and RFID information from these devices can achieve near-real-time asset tracking, monitoring, and alerting. More broadly, the data generated by these devices help produce actionable results, informs business intelligence, and helps businesses improve operations.

Furthermore, since blockchain is a distributed system, it cannot directly sense the status of logistics, such as the temperature of food during transportation. Therefore, the

application of blockchain to logistics must be completed in conjunction with IoT. In the actual production process, it is necessary to attach an RFID tag to each goods to store information, such as ID and product information. In particular, when transporting goods with high requirements on temperature and other conditions, such as fruits, medicines, and frozen foods, the real-time information of the RFID tag of the cargo box is read through the handheld terminal to ensure the validity of the goods. According to the goods specification information (including the inspection and quarantine information of agricultural goods) stored in RFID tags, the quality inspection can be performed on the goods. Besides, the RFID tags can be used as the carrier of anticounterfeiting traceability information.

*3.4. Payment Channel.* In the current blockchain applications, such as Bitcoin [19], a new block needs to be propagated to all nodes in the network immediately before the next block is generated. This will ensure the security of the Bitcoin system. Therefore, to meet this characteristic, Bitcoin uses the proof of work (POW) to control the rate of block production to one block in 10 minutes. Furthermore, to ensure the block's propagation rate, the block size is also limited [39].

Using the payment channel to achieve off-chain payment has become a promising approach to solve Bitcoin's efficiency and scalability issues [15–18]. Both parties' transactions in the payment channel do not have to be chained. In short, the payment channel is like a secure deposit box, where they lock coins into a deposit secured by a smart contract. After the transactions are completed, the channel is closed, and all transactions between them are submitted to the network. In this step, the two parties get back their own coins according to their status after the last transaction.

## 4. SESCF

In this section, we first introduce the secure and efficient blockchain-based supply chain framework (SESCF), which solves the existing problems in the information flow, logistics, and capital flow. Immediately afterward, we provide the process of generating transactions in SESCF.

*4.1. Blockchain-Based System Model.* Figure 1 illustrates a general overview of the system architecture to solve the existing problems of information flow, logistics, and capital flow. Our system introduces a competitive bidding mechanism to ensure the transparency of information between entities in SESCF. In addition, payment channels are used to allow the security of coins of these entities. The proposed system model follows a layered architecture and is divided into three layers. The first layer is the user layer that handles the offline transportation between entities of SESCF. These interactions involve goods along with RFID tags that store goods information. The second layer, i.e., the transaction layer, handles the online transactions of SESCF, including (i) trading event, in which the goods demander firstly publishes an order demand, then the goods suppliers bid based on it, and finally, the goods demander chooses the winning bidder to conduct the transaction, (ii) payment event, in which two parties conduct capital transactions by opening a payment channel, and (iii) delivery event, which is inspecting goods information when receiving the goods using the information stored in RFID. The third layer is the blockchain layer that stores transactions and data. Also, it tracks the source of the problematic goods involved in SESCF. It is worth noting that the blockchain layer enforces strict access control strategies to prevent unauthorized reading and writing of data.

SESCF only allows registered users to perform specific transactions. If a node wants to join SESCF, it needs to be authenticated by Certificate Authority first. That is, Certificate Authority is responsible for registering the identity of all entities in SESCF. In addition, each function is allowed to be performed by specific entities. The entities are described as follows: *Supplier*, *Producer*, *Retailer*, and *Certificate Authority*.

*Supplier (S).* Suppliers are the enterprises that supply various required resources to producers and their competitors, which include the provision of raw materials, equipment, energy, and labor.

*Producer (P).* Producers are the enterprises that supply goods to retailers. They use raw materials or components to make a series of daily consumer goods through relatively automated machinery and production processes. When P and S reach a demand agreement, P will apply to open an off-chain channel with S.

*Retailer (R).* Retailers are the enterprises whose sales are mainly from retail, that is to say, retailers sell goods directly to the end consumers. Compared with producers, they are in the final stage of goods circulation. Similarly, when R and P reach a demand agreement, R will apply to open an off-chain channel with P.

*Certificate Authority (CA).* Certification Authority is an organization that issues digital certificates. Furthermore, as a trusted third party in e-commerce transactions, it is responsible for checking the legality of public keys in the public-key system.

In the following paper, since the scenario between retailers and producers is similar to that between producers and suppliers, we only take the raw material transaction between suppliers and producers as an example. In addition, the system model is divided into three parts: information flow, logistics, and capital flow.

*4.2. Information Flow.* In goods circulation, the flow of all information is called information flow. Moreover, information sharing helps improve the efficiency of the system. However, due to the information asymmetry of upstream and downstream caused by factors such as time, space, and technology, the traditional supply chain information system is distributed in the hands of various participants. So far, no platform can gather all goods information with complete trust. Therefore, we propose to use blockchain to solve the problems of the information flow. Here, the information flow mainly refers to supply and demand information.

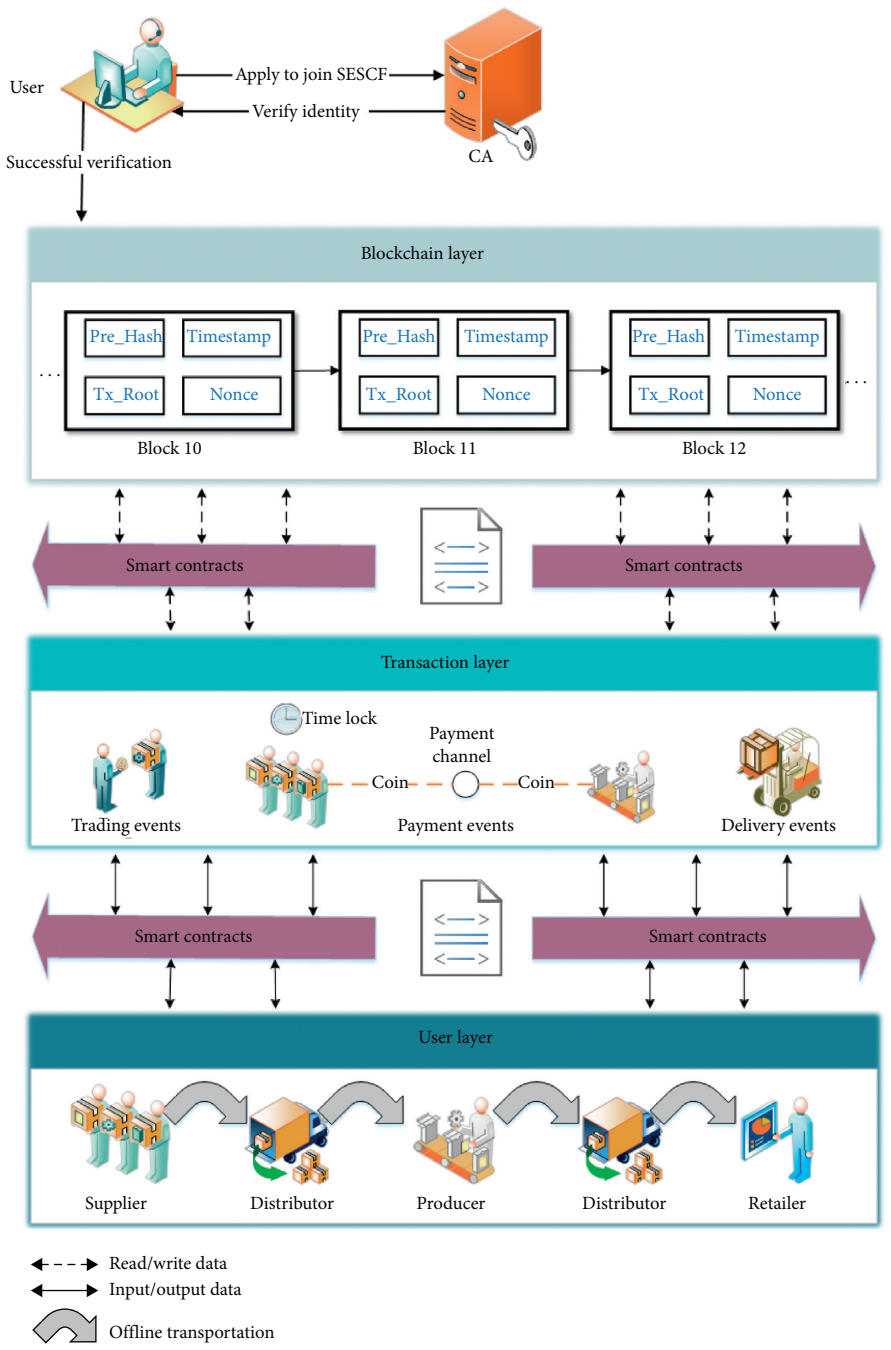


Figure 1: Blockchain-based supply chain model.

Each participating entity has a pair of public and private keys and participates by invoking functions within the smart contract. To address the opacity of supply and demand information between upstream and downstream, a competitive bidding method is introduced in SESCF. Figure 2 outlines the sequence flow for a scenario where the goods demander, i.e., producer, and the goods suppliers, i.e., suppliers, complete the trading event by invoking the smart contract. When P needs a batch of raw materials, he executes the *SendTender* $T(P, S_1, \ldots, S_N, T_{\text{tender}})$ function to release the tender announcement to $S_1, \ldots, S_N$, i.e., $N$ suppliers in SESCF. Here, $T_{\text{tender}}$ is a demand order for raw materials signed with $P$'s private key, and its format is shown in (1). Then, the event *RequestedSendTender* is invoked and is made available to all active participants (i.e., $P$ and $S_1, \ldots, S_N$). $S_1, \ldots, S_N$ execute the *SendBid* $S_1, \ldots, S_N, P$, eprice, $T_{\text{tender}}$ function to send bid price before the deadline for bidding, where the attribute *eprice* refers to the bid price encrypted with $P$'s public key. It is worth noting that if entities do not participate in this bid, there is no need to send bid price. Once the bid deadline is reached, $P$ uses his private key to decrypt $S_1$, *eprice*, ..., $S_N$.*eprice* to get, *price*, ..., $S_N$.*price*, and then, the *ComputeWinBidder* ($P$, $S_1$, *price*, ..., $S_N$·*price*, $T_{\text{tender}}$) function is executed to choose the supplier with the

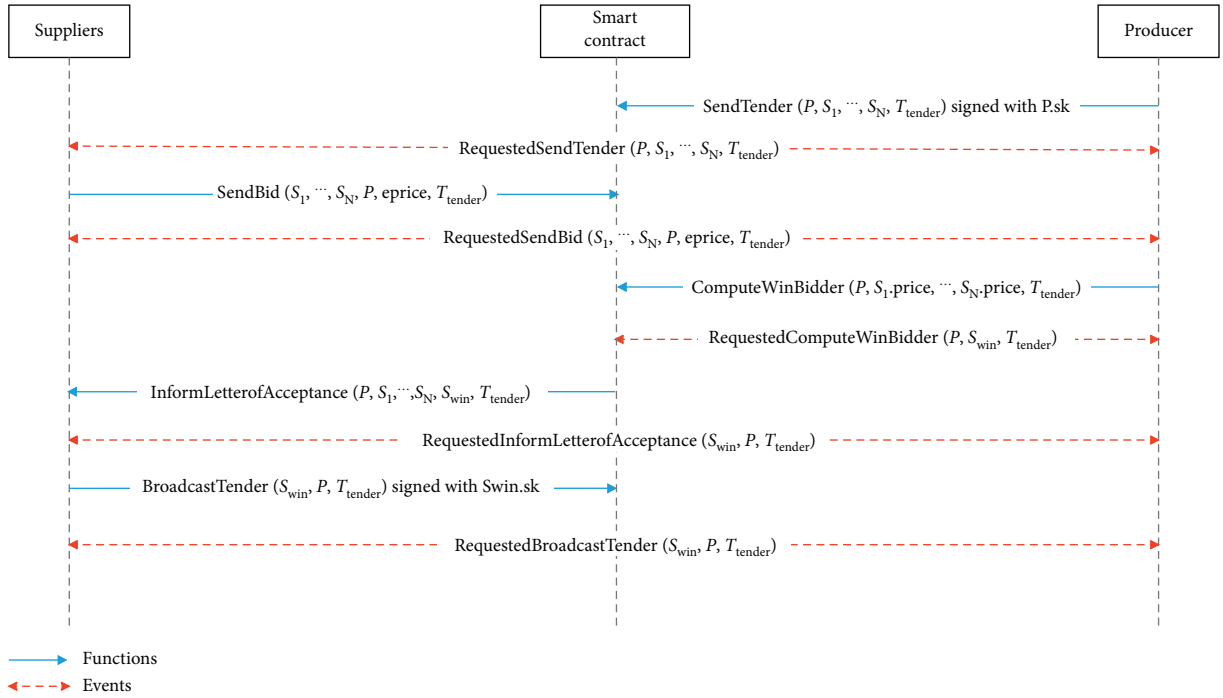


Figure 2: Sequence diagram showing interactions between suppliers, smart contract, and producer to address the opacity of supply and demand.

lowest bid price as $S_{win}$, i.e., the winning bidder. The *RequestedComputeWinBidder* event triggers the smart contract, which then automatically executes *InformLetterofAcceptance* ($P$, $S_1$, …, $S_N$, $S_{win}$, $T_{tender}$) to notify the participating entities of the winning bidder. After $P$ and $S_{win}$ reach an agreement through the event *RequestedInformLetterofAcceptance*, $S_{win}$ signs $T_{tender}$ with his private key and executes the *BroadcastTender* ($S_{win}$, $P$, $T_{tender}$) function to broadcast this transaction. The event *RequestedBroadcastTender* is activated to notify the interacting entities of this transaction (i.e., $P$ and $S_{win}$) at that specific time point:

$$T_{tender} = (Oname \parallel Osize \parallel Qspec \parallel \mathrm{Num}_{max} \parallel Txtype \parallel PD \parallel LD \parallel T_1 \parallel T_2 \parallel T_3), \quad (1)$$

where *Oname* represents the order name of the tender, specifically, it means the raw materials required by *P*. *Osize* refers to the order quantity of the tender, which represents the number of raw materials required by *P*. *Qspec* refers to the quality specifications of the goods and $\mathrm{Num}_{max}$ refers to the maximum number of substandard goods. *Txtype* refers to the type of transaction, and it should be noted that *Txtype* is *txptos* in $T_{tender}$, which refers to the transaction sent by *P* to *S*. *PD* refers to the deposit, which is part of the price that *P* needs to pay to *S* in advance. *LD* represents liquidated damages, which is the amount payable to *P* if *S* does not deliver before the delivery deadline. The three times in $T_{tender}$ represent three deadlines: $T_1$ is the deadline for bidding, $T_2$ is the deadline for computing $S_{win}$, and $T_3$ is the deadline for delivery.

*4.3. Logistics.* Logistics is the process of integrating the functions of transportation, storage, and distribution from the supplier to the receiver. Because the structure of the supply chain is extremely complex, there are many counterfeit goods, and it is a very time-consuming and challenging task to track goods in the logistics. Therefore, in SESCF, we mainly solve the problems of substandard quality and traceability in logistics.

Figure 3 represents the message sequence diagram for quality inspection. That is, perform quality inspection when the goods are delivered, and refuse to accept them if the quality is unqualified. Following the offline delivery from $S_{win}$ to $P$, the function *SendDelivery* ($S_{win}$, $P$, $T_{delivery}$) is executed and the event *RequestedDelivery* is activated. Here the parameter $T_{delivery}$ is a delivery note signed with the private key of $S_{win}$, and its format is shown in (2). After receiving the goods offline, $P$ calls the function *InspectGoods* ($P$, $S_{win}$, *RFIDInfo*, *Qspec*, $\mathrm{Num}_{max}$) to inspect the quality, and the parameters *RFIDInfo* represents the goods information that is stored in the RFID. The *RequestedInspectGoods* event triggers the smart contract to automatically execute *SendFlag* ($P$, $S_{win}$, *count*, *flag*) to notify the inspection result, where the attribute *count* indicates the total number of unqualified goods, and *flag* indicates whether it has passed the test, i.e., true or false. If *flag* is true, $P$ signs $T_{delivery}$ with his private key and executes *BroadcastDelivery* ($P$, $S_{win}$, $T_{delivery}$) to broadcast this transaction. Otherwise, $P$ returns the goods to $S_{win}$.

On the other hand, although fakes often appear in our lives, tracking fakes in the supply chain is very troublesome. Therefore, Figure 4 shows the message sequence diagram in which the user uses the smart contract to track the origin of

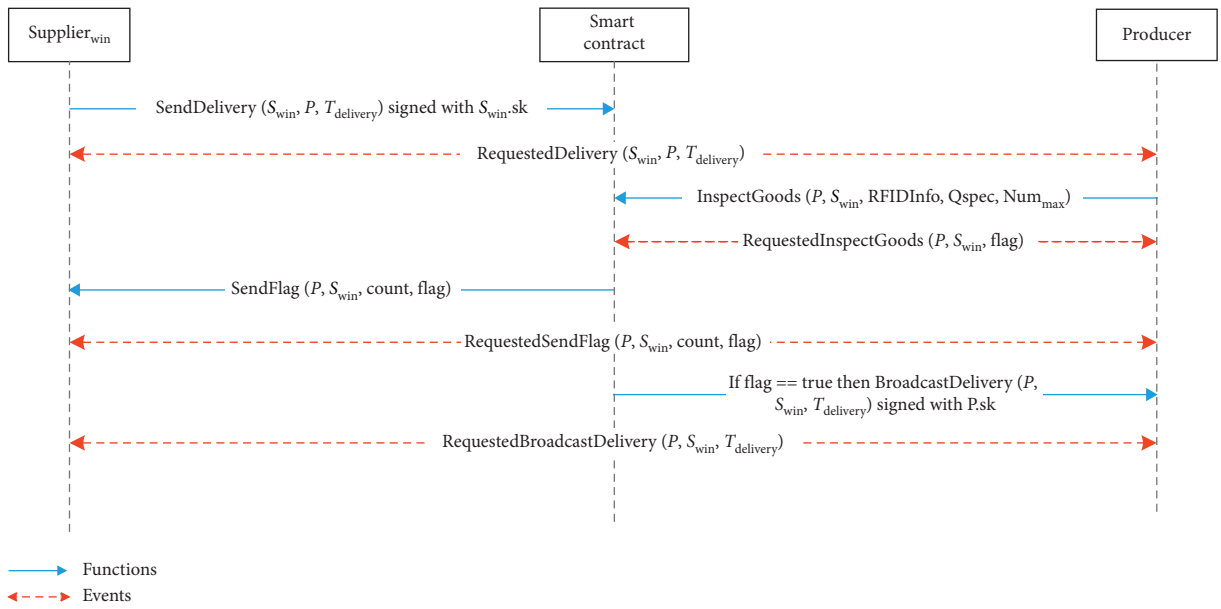


Figure 3: Sequence diagram showing interactions between Supplier$_{win}$, smart contract, and producer to inspect the goods quality.

goods. Once the user finds a fake, he first scans the RFID tag of it and passes *Gid* and *id_set* which are stored in the RFID as parameters to the function *TraceGoods* (*user*, *Gid*, *id_set*). By doing so, users can trace not only the producer of the fake but also the suppliers who provide raw materials for the fake. The *RequestedTraceGoods* event triggers the smart contract to query transactions in the blockchain, and then, the query results are returned to the user through the function *SendTraceResult* (*user*, *tx_set*), where *tx_set* is the transaction set.

$$T_{\text{delivery}} = (Exhash \parallel Txtype \parallel Tx \parallel Goods), \quad (2)$$

where *EXhash* represents the hash value of the related tender transaction. *Txtype* refers to the type of transaction, and *Txtype* is *txstop* in $T_{\text{delivery}}$, which refers to the transaction sent by *S* to *P*. *Tx* is the delivery order, and *Goods* specifically refers to the content of the RFID tag. Moreover, the RFID tag contains *Gid* (goods id), *id_set* (the raw material id set of goods), goods name, origin, source, production date, goods specification information (including expiration date), etc. It should be noted that each *Gid* is unique in our system. For agricultural goods, *Qspec* refers to the inspection and quarantine information. Especially for goods with timeliness, such as meat and fruits, special attention should be paid to the expiration date.

*4.4. Capital Flow.* Capital flow is the process of currency circulation. The efficiency and dynamic optimization of upstream and downstream capital operations are directly related to the operation quality of corporate capital circulation. However, payment defaults are common problems in the supply chain. To solve payment defaults, we consider opening a payment channel on both sides of the transaction. Moreover, placing capital transactions off-chain reduces the waste of on-chain resources.

Figure 5 outlines the sequence flow for a scenario where the producer and the winning bidder pay the deposit, balance, and liquidated damages in the payment channel. Following $S_{\text{win}}$ broadcasts $T_{\text{tender}}$, the function *OpenChannel* ($P$, $S_{\text{win}}$, $T_{\text{tender}}$, *timelock*) is executed to apply for opening a payment channel between $P$ and $S_{\text{win}}$, where *timelock* refers to the active time of the channel. After the channel is successfully opened, $P$ and $S_{\text{win}}$ first sign it with their private keys through the event *RequestedOpenChannel* and then put some coins from the wallet into it by executing the functions *InputCoins* ($P$, *coin*, *Channel*) and *InputCoins* ($S_{\text{win}}$, *coin*, *Channel*). In particular, in the multisignature address controlled by both parties, $P$ is required to store not less than the price of the order, and $S_{\text{win}}$ is required to store not less than *LD*. Now, $P$ executes the function *PayDeposit* ($P$, $S_{\text{win}}$, *PD*, *Channel*) to pay the deposit to $S_{\text{win}}$. If $S_{\text{win}}$ delivers on time, *PayBalance* ($P$, $S_{\text{win}}$, *balance*, *Channel*) is executed by $P$ to pay the balance. Otherwise, $S_{\text{win}}$ executes *PayLiquidatedDamages* ($S_{\text{win}}$, $P$, *LD*, *Channel*) to pay the liquidated damages. In particular, if $S_{\text{win}}$ delivers before the deadline, both $P$ and $S_{\text{win}}$ can execute the function *CloseChannel* ($P$, $S_{\text{win}}$, $T_{\text{tender}}$, *Channel*) to apply for closing the payment channel in advance. The event *RequestedCloseChannel* is activated to notify the interacting entities that the channel is closed successfully, and $P$ and $S_{\text{win}}$ will get back their respective coins according to the status of the last transaction. Unlike digital currencies such as Bitcoin, the form of payment transaction $T_{\text{pay}}$ is shown in the following equation:

$$T_{\text{pay}} = (Exhash \parallel Txin \parallel Txout \parallel Txtype \parallel Txcount). \quad (3)$$

*EXhash* represents the hash value of the associated tender transaction. *Txin* is the input address, and *Txout* is the output address. *Txtype* is the type of payment

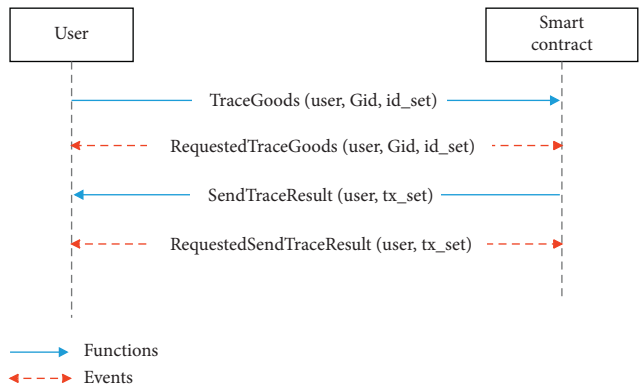


Figure 4: Sequence diagram showing interactions between user and smart contract to track the origin of goods.

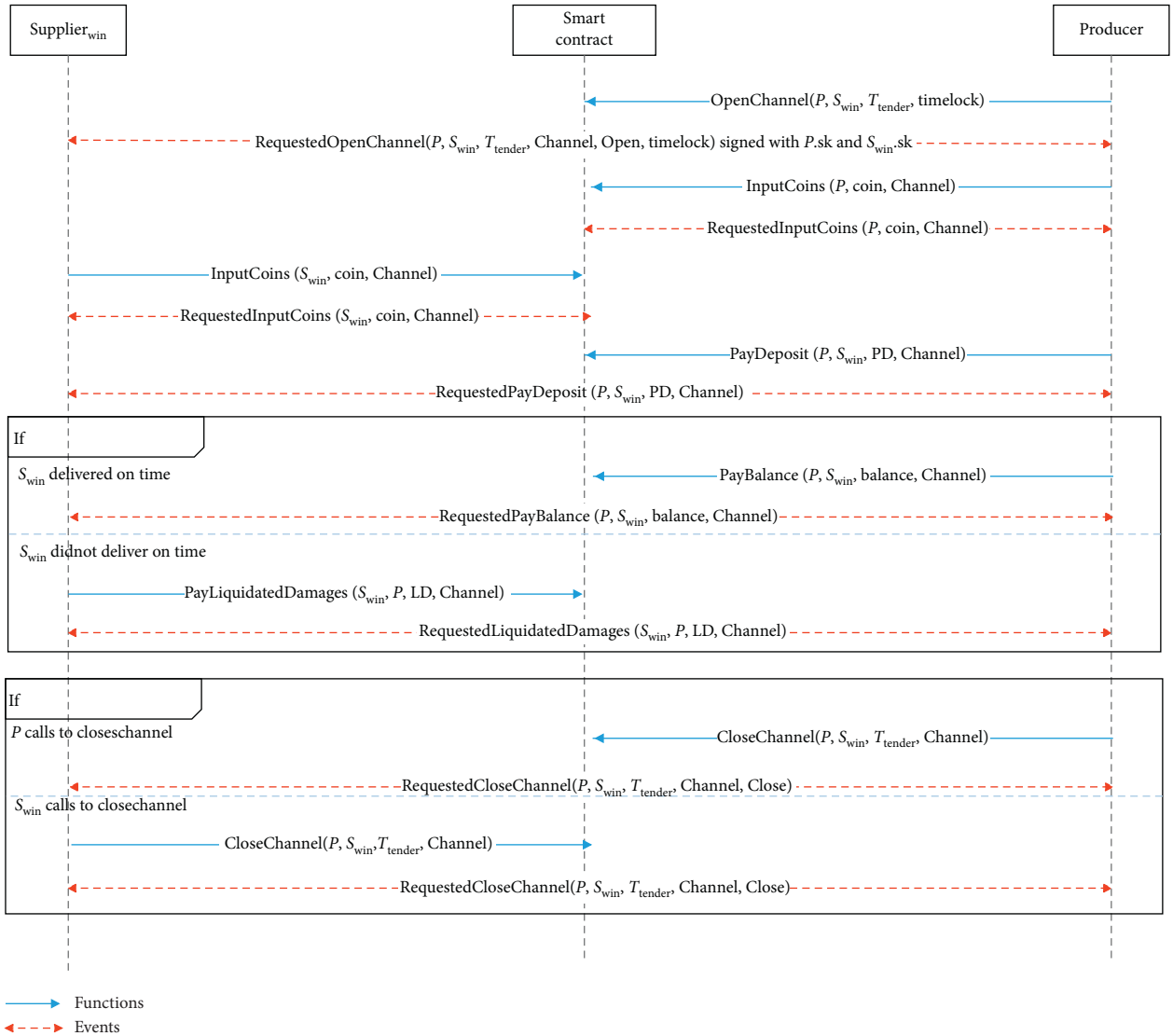


Figure 5: Sequence diagram showing interactions between Supplier$_{win}$, smart contract, and producer to pay deposit, balance, and liquidated damages.

transaction, and we determine it based on the increase or decrease of $P$'s coins and $S_{win}$'s coins after the channel is closed: (i) if $P$'s coins increase and $S_{win}$'s coins decrease, it means that $S_{win}$ did not deliver on time, and $S_{win}$ pays $P$ the liquidated damage, and then *Txtype* is *LD*; (ii) if $P$'s coins decrease and $S_{win}$'s coins increase, it means that $S$ delivered on time, and $P$ pays $S_{win}$ the price of the goods, and then *Txtype* is the price of the goods. *Txcount* is the amount of $T_{pay}$.

## 5. Implementation Details of SESCF

In this section, we implement SESCF from three parts: information flow, logistics, and capital flow. In particular, specific smart contracts for each part are given.

*5.1. Information Flow.* As discussed earlier, we propose to use a competitive bidding mechanism to ensure the symmetry of upstream and downstream supply and demand information. That is, $P$ issues a tender announcement, then $S_1, \ldots, S_N$ bid, and finally, $P$ chooses the lowest bidder as $S_{win}$. According to these processes, three primitives are defined: *Tender*, *Bid*, and *Compute*. The process is described in Algorithm 1.

*Tender*. In the tender stage, $P$ first sends $T_{tender}$ signed with $P$'s private key $sK_P$ to the smart contract. Then, the smart contract forwards $T_{tender}$ to $S_1, \ldots, S_N$ after receiving $T_{tender}$.

*Bid*. In the bid stage, after receiving $T_{tender}$ forwarded by the smart contract, $S_1, \ldots, S_N$ choose to bid or give up according to $T_{tender}$. If they choose to bid, they need to use $P$'s public key, i.e., $pK_P$, and a random number $r$ to encrypt the bid price and then use NIZK to compute a zero-knowledge proof $\pi$ to prove that *eprice* is constructed correctly. It is worth noting that each user does not know others' bids before committing to their own, so that the user's bid is independent of others'. As long as $P$ does not disclose the bid information, users' bids remain private even after bidding. After receiving the message from $S_1, \ldots, S_N$, the smart contract first determines the current time is less than $T_1$, ensures that $S_1, \ldots, S_N$ conduct bidding exercises for the first time, and then verifies their commitments are correct. If verifications are passed, the smart contract automatically forwards their bid messages to $P$.

*Compute*. In the compute stage, $S_1, \ldots, S_N$ first call for computing. If $S_1$ $(i \in \{1, \ldots n\})$ has not called for computing before $T_2$, it means that it has abandoned the bid. Then, $P$ decrypts their bids with$sK_P$ and compares them with the price cycle to select $S_{win}$ $(S_{win} \in \{S_1, \ldots, S_N\})$ with the lowest price. After that, the smart contract sends the message whether they are the winning bidder to $S_1, \ldots, S_N$. Finally, $S_{win}$ uses $sk_S$ to sign and broadcast $T_{tender}$.

*5.2. Logistics.* At the time of delivery, the buyer first scans the RFID and then uses *RFIDInfo*, *Qspec*, and $\text{Num}_{max}$ as inputs to trigger the smart contract for quality inspection. We define a primitive to complete the quality inspection: *Inspect*. The process is described in Algorithm 2.

*Inspect*. $S_{win}$ first sends $T_{delivery}$ signed with $sk_S$ to the smart contract before the latest delivery time $T_3$, and the smart contract forwards $T_{delivery}$ to $P$. Then, after $P$ receives the goods offline, he scans the RFID tags. Meanwhile, *Qspec*, $\text{Num}_{max}$, and *RFIDInfo* are passed as parameters to the smart contract for quality inspection. Specifically, the smart contract traverses the information of the goods and compares them with *Qspec*. There are two cases in the inspection result: (i) if the number of substandard goods is not greater than $\text{Num}_{max}$, then $P$ will sign $T_{delivery}$ with $sk_P$ and broadcast $T_{delivery}$. (ii) If the number of substandard goods is greater than $\text{Num}_{max}$, the smart contract will notify $P$ and $S_{win}$ of the number of substandard goods, and $P$ will return the goods to $S_{win}$ offline.

Algorithm 3 describes the process of tracking the origin of fakes in SESCF. Similarly, we define a primitive to complete the traceability: *Trace*.

*Trace*. Using a unique *Gid* to identify the goods can achieve traceability. In Section 4.3, we have introduced *Gid* and *id_set*. Once a problem is found with the goods, the user can scan the RFID to obtain the unique *Gid* and *id_set*. The goods information contained in $T_{delivery}$ is stored on the blockchain so that users can locate the corresponding $T_{delivery}$ and further locate the supplier of the problem goods according to *Gid* and *id_set*.

*5.3. Capital Flow.* Algorithm 4 describes the process of solving payment defaults in the supply chain. Specifically, after $S_{win}$ broadcasts $T_{tender}$, a payment channel $C < P, S_{win} >$ with a time lock of $T_3$ is opened between $P$ and $S_{win}$ immediately. We define three primitives to complete the payment: *Open channel*, *Payment*, and *Close channel*. Here, *txhash* refers to $T_{tender}$'s hash value.

*Open channel*. In the open channel stage, $P$ releases the coins not less than the order price into the channel, and $S_{win}$ releases the coins not less than *LD* into the channel. As well as, the time lock is set to $T_3$. Finally, both parties use their private keys to sign the channel $C < P, S_{win} >$. The role of the time lock is to set the latest closing time of the channel. In simple terms, after $T_3$, if $C < P, S_{win} >$ is not closed yet, it will automatically close, at the same time, the coins in the channel are redistributed and submitted to the blockchain.

*Payment*. In the payment stage, $P$ pays $S_{win}$ *PD* after the channel is opened. If $S_{win}$ fails to deliver to $P$ before $T_3$, he will pay *LD* to $P$ automatically through the smart contract at $T_3$. On the contrary, if $S_{win}$ successfully delivers to $P$ before $T_3$, $P$ will pay $S_{win}$ the balance through the smart contract.

*Close channel*. After successful delivery before$T_3$, $P$ or $S_{win}$ applies for closing the payment channel. If $C < P, S_{win} >$ has not been closed at $T_3$, it indicates that $S_{win}$ failed to deliver before $T_3$; then $S_{win}$ pays *LD* to $P$. $P$ and

(1) **tender**: $P$ inputs $T_{\text{tender}}$ signed with $sK_P$
(2) Smart contract forwards $T_{\text{tender}}$ to $S_1, \ldots, S_N$
(3) **bid:** $S_1, \ldots, S_N$ receive $T_{\text{tender}}$
(4) sample encryption randomness $r$
(5) eprice := ENC ($pk_P$, $r$, \$price)
(6) $\pi$ := NIZK.prove(($P$, eprice), (\$price, $r$))
(7) $S_1, \ldots, S_N$ send (eprice, $\pi$)
(8) Smart contract receives (eprice, $\pi$) from $S_i$ ($i \in \{1, \ldots N\}$)
(9) assert current time $T < T_1$
(10) assert this is the first bid input
(11) assert NIZK.verify ($R_{\text{compute}}$, $\pi$, ($P$, eprice))
(12) Smart contract sends ($S_i$, eprice) to $P$
(13) store eprice, $T_{\text{tender}}$ to use later in
(14) **compute:** on input (compute) as Supplier $S_1, \ldots, S_N$
(15) assert current time $T_1 < T < T_2$
(16) $P$ decrypts and stores(\$ price) := DEC($sk_P$, eprice)
(17) **if** this is the last compute received **then**
(18) for $i \in \{1, \ldots, n\}$ such that $S_i$ has not called for compute
(19) \$price := $+\infty$
(20) **end if**
(21) winsupplier := $-1$
(22) bestprice: = $+\infty$
(23) **for** $i \in \{1, \ldots, n\}$ **do**
(24) **if** $S_i$.price < bestprice **then**
(25) bestprice := $S_i$.price
(26) $S_{\text{win}} := S_i$
(27) **end if**
(28) **end for**
(29) store and output $S_{\text{win}}$
(30) $S_{\text{win}}$ signs $T_{\text{tender}}$ with $sk_S$
(31) $S_{\text{win}}$ broadcasts $T_{\text{tender}}$
(32) Relation (statement, witness) $\in R_{\text{compute}}$ is defined as
(33) parse statement as ($P$.eprice)
(34) parse witness as (\$price, $r$)
(35) assert eprice = ENC($pk_P$, $r$, \$price)

Algorithm 1: *Competitive bidding()* for $P$ and $S_1, \ldots, S_N$.

(1) **inspect:** $S_{\text{win}}$ inputs $T_{\text{delivery}}$ signed with $sk_S$
(2) assert current time $T <$ Exhash.$T_3$
(3) $N$ := Goods.count
(4) assert $N$ = Exhash.Osize
(5) Smart contract forwards $T_{\text{delivery}}$ to $P$
(6) $P$ scans and inputs the RFID tag of the goods
(7) $P$ inputs (Qspec, $\text{Num}_{\text{max}}$)
(8) count := 0
(9) **for** $i \in [N]$ **do**
(10) compare $\text{Goods}_i$.information with Qspec
(11) **if** the quality of $\text{Goods}_i$ is not up to standard **then**
(12) count++
(13) end if
(14) end for
(15) if count > $\text{Num}_{\text{max}}$ then
(16) Smart contract sends count to $P$ and $S_{\text{win}}$
(17) else
(18) $P$ signs $T_{\text{delivery}}$ with $sk_P$
(19) $P$ broadcasts $T_{\text{delivery}}$
(20) end if

Algorithm 2: *Inspect()* for $P$ and.$S_{\text{win}}$.

```
(1) trace: N_i scans the RFID tag of the goods
(2) N_i inputs Gid and id_set
(3) total := id_set.length +1
(4) num = 0
(5) for tx.Txtype = txstop and num = total do
(6)    if Gid = tx.Goods_j.Gid or id_set.id = tx.Goods_j.Gid then
(7)       send tx to N_i
(8)       num++
(9)    end if
(10) end for
```

Algorithm 3: *Trace()* for user $N_i$.

```
(1) open channel: P inputs (open, C < P, S_win >, S_win)
(2) Smart contract sends (C < P, S_win >, txhash) to S_win
(3) P inputs Coin_P to C < P, S_win >
(4) P.account := Coin_P
(5) assert Coin_P > txhash.price
(6) S_win inputs Coin_S to C < P, S_win >
(7) S_win.account := Coin_S
(8) assert Coin_S > txhash.LD
(9) t := tx.hash.T_3
(10) P and S_win sign C < P, S_win >
(11) payment: after P and S_win sign the channel
(12) P pays txhash.PD to S_win
(13) P.account := P.account—txhash.PD
(14) S_win.account := S_win.account + txhash.PD
(15) if S_win did not deliver the goods before t then
(16)    S_win pay txhash.LD to P
(17)    P.account := P.account + txhash.LD
(18)    S_win.account := S_win.account—txhash.LD
(19) else
(20)    txhash.balance := S_win.price—txhash.PD
(21)    P pay txhash.balance to S_win
(22)    P.account := P.account − txhash.balance
(23)    S_win.account := S_win.account + txhash.balance
(24) end if
(25) P or S_win call to close channel
(26) close channel: on input (close, C < P, S_win >, txhash) from P and S_win
(27) P.wallet := P.wallet + P.account
(28) S_win.wallet := S_win.wallet + S_win.account
(29) P and S_win sign off
```

Algorithm 4: *Payment()* for $P$ and $S_{win}$.

$S_{win}$ sign off $C < P, S_{win} >$ and submit the balance of their coins to the blockchain.

## 6. Evaluation

In this section, we first discuss the assumptions of SESCF and use simulation tools to prove the feasibility of the smart contracts. Then we implement an efficient blockchain-based supply chain system. Our system runs on the Windows operating system. Finally, we give a comparison between SESCF and the existing blockchain-based supply chain frameworks.

*6.1. Simulation and Results.* We test the smart contracts proposed in this paper in the simulation mode. Specifically, we use Remix, MetaMask, and Ganache to test the feasibility and performance of smart contracts. Remix is the official online integrated development environment (IDE) of Ethereum. The Solidity language can be used to complete the online development, compilation, testing, deployment, debugging, and interaction of Ethereum smart contracts in the web page. Ganache provides virtual users with a certain number of virtual cryptocurrencies. Through the visual interface, users can intuitively set various parameters, browse, and view accounts and transaction data. MetaMask is a Chrome plug-in, so it is very convenient to use it to

complete transfer and other operations in the browser. The specifications of the system are as follows: Intel Core i7-10510U, 2.4 GHz processor, 16 GB RAM, and 500 GB storage.

Ethereum stipulates two accounts: ordinary accounts and smart accounts. For ordinary transfer transactions, that is, calling ordinary accounts, the required gas is a fixed $2.1 \times 10^4$. However, if users call smart accounts, the required gas is different because the complexity of the smart contracts is different. The more resources the transaction takes up, the more gas is required. Following are the performances we tested to evaluate the proposed solution: (i) the gas cost required to deploy smart contracts, (ii) the total amount of gas consumed by different numbers of nodes on the competitive bidding mechanism, (iii) the total amount of gas consumed by different quantities of goods when inspecting the quality of goods, and (iv) the total amount of gas consumed by different numbers of transactions when tracing goods.

Figure 6 shows the gas consumption of the four smart contracts deployed in SESCF. As we can see from the figure, the *Competitive bidding()* consumes the most gas and the *Inspect()* consumes the least gas. In Ethereum, gas needs to be converted into ETH for payment. Here, we set gas price = 1 Gwei = $10^{-9}$ Eth, and then, the total cost of deploying smart contracts is $7.69 \times 10^{-3}$ Eth.

In Figure 7, the amount of gas consumed by different numbers of suppliers in the competitive bidding mechanism is shown. The competitive bidding mechanism consists of three functions, i.e., *Tender()*, *Bid()*, and *Compute()*. In SESCF, as the number of suppliers increases, the resources (calculation, memory, etc.) occupied during bidding also increase. It is obvious in the histogram that as the number of supplier nodes increases, the more gas is required to run the smart contract. In addition, the *Bid()* function consumes the maximum amount of gas for execution and transaction as compared to the other functions because the *Bid()* function not only needs to encrypt bids but also needs to calculate and verify $\pi$.

In the delivery stage, $P$ needs to perform quality inspection on the goods first. Figure 8 shows the amount of gas consumed by different numbers of goods in the *Inspect()* function. Through a large number of experiments, we have found that the gas consumption in the inspect phase is proportional to the number of goods, that is, as the number of goods increases, the more gas is required.

To compare the number of transactions in SESCF with the amount of gas required for tracing goods, we add different numbers of transactions in SESCF for testing. The results are shown in Figure 9. We stipulate in the smart contract *Trace()* that once the transactions are queried, it will stop running. Therefore, the gas consumed by the *Trace()* function is not only related to the number of transactions in SESCF but also related to the ID of the block where the goods is located. However, by drawing these histograms, we concluded that more transactions in SESCF will cost more than fewer transactions.

*6.2. Construction and Results.* The experimental results in the previous section show that the smart contracts we proposed are feasible. Next, we develop an efficient blockchain-based supply chain system *SescfDapp*. *SescfDapp* runs on the Windows operating system, and it is implemented using the C++ language. Furthermore, we assume that no entity on the network has enough computing power to destroy more than half of the network nodes, and only registered entities can buy and sell goods in *SescfDapp*.

First, the user runs *SescfDapp* and then enters the port number and selects the identity on the login interface. After the login is successful, *SescfDapp* automatically obtains the current user's IP address and generates a pair of a public key and private key and initializes a wallet for the user. Next, *SescfDapp* sends the user's IP address, port number, identity, and public key to the CA for verification. If the verification is successful, the system will receive a certificate binding IP address, port number, identity, and public key. But in our simplified supply chain system, the CA server verification step is omitted. Therefore, we consider the IP address, port number, and public key as certificates.

After the user logs in *SescfDapp*, the home page displays the IP, port number, and identity of the user. Further, he can operate on the competitive bidding, inspect, trace, and payment pages. It is worth noting that nodes have only three different identities: supplier, producer, and retailer. Following are the running times of *SescfDapp*. We proved the efficiency of *SescfDapp* by analyzing them.

Figure 10 shows the time cost on the competitive bidding mechanism. We provide different numbers of nodes in *SescfDapp* and run multiple times to study the impact on bidding time cost. It is clearly visible from the graph that the *Compute()* function consumes the maximum amount of time cost for execution as compared to the *Tender()* function. This is because the *Compute()* function not only needs to broadcast the calculation result to $S_i$ but also needs to perform decryption and logically complex operations. The *Tender()* function only needs to broadcast $T_{\text{tender}}$ to $S_i$. Therefore, the time cost of *Tender()* is relatively low. When running the *Bid()* function, we found that the average running time of bidding for a user is $3.08 \times 10^{-1}$ seconds. The execution time of the *Bid()* function mainly includes two parts: (i) the time for $S_i$ ($i \in \in \{1, \ldots, N\}$) to send eprice and $\pi$ and the size of each proof $\pi$ is 1.25 MB; (ii) the time for the smart contract to verify $\pi$.

We first open a payment channel between $P$ and $S_{\text{win}}$ in *SescfDapp* and perform the payment operations. We observe that the channel can be opened between any two users in *SescfDapp*, and it only takes $4 \times 10^{-2}$ seconds to make a payment operation on the path of $P$ and $S_{\text{win}}$. This greatly reduces the overhead on the blockchain.

In Figure 11, the time cost on inspecting and tracing goods in the logistics phase is shown. We test the changes in the execution time of the *Inspect()* function as the number of goods increased, and the changes in the execution time of the *Trace()* function as the number of transactions increased. It is observed from experiments that the *Inspect()* function is


Figure 6: The gas cost required to deploy smart contracts.

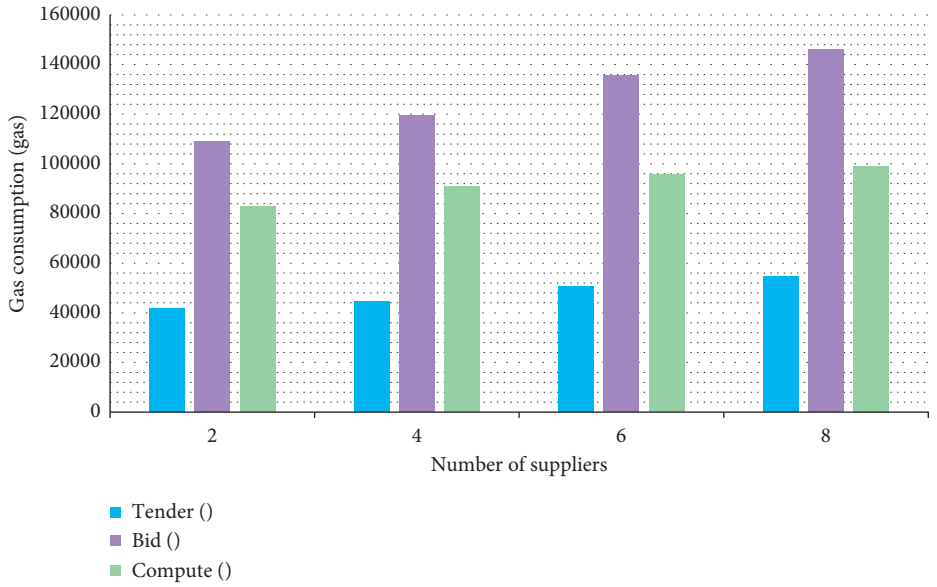


Figure 7: The total amount of gas consumed by different numbers of nodes on the competitive bidding mechanism.

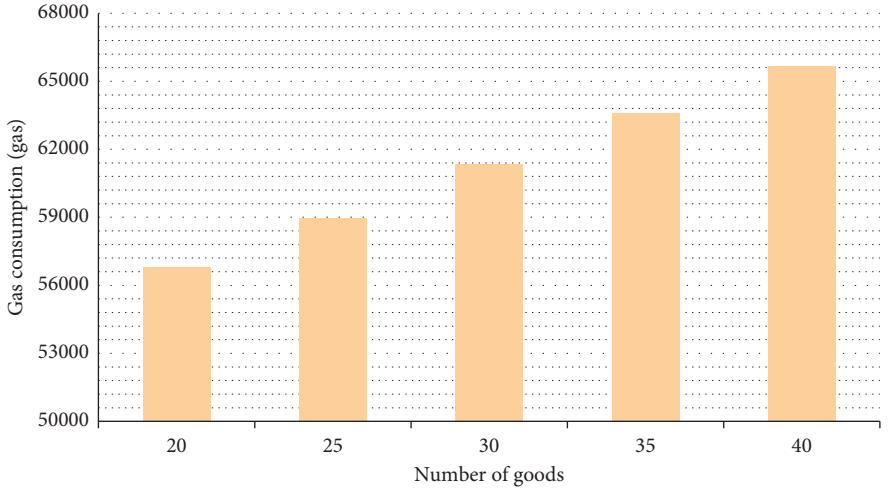


Figure 8: The total amount of gas consumed by different numbers of goods when inspecting the quality of goods.

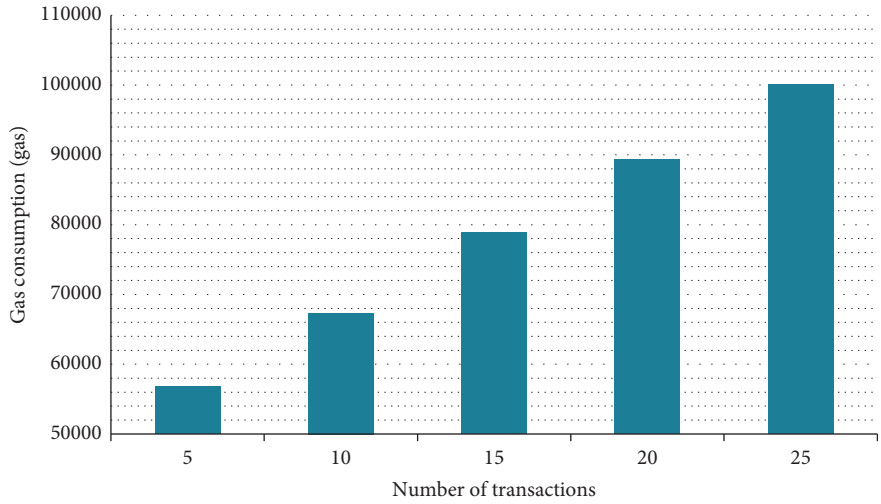


Figure 9: The total amount of gas consumed by different numbers of transactions when tracing goods.

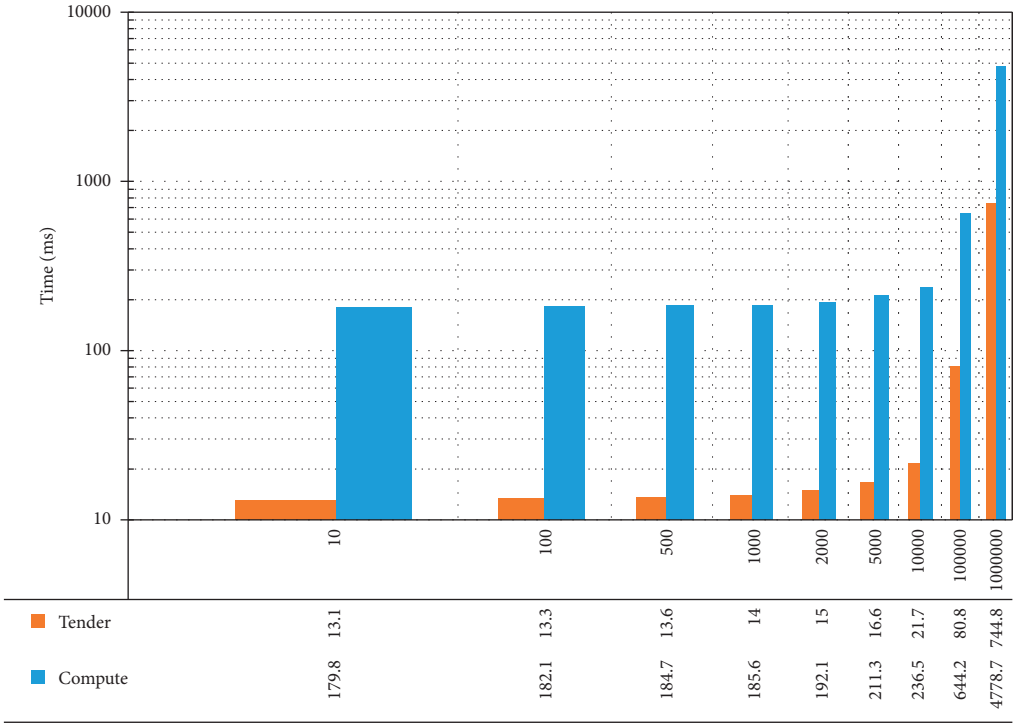


Figure 10: Time cost of tendering and computing.

less affected by the number of goods. When the number of goods reaches $10^6$, the running time of *Inspect()* is only $1.99 \times 10^{-1}$ seconds. However, the *Trace()* function is greatly affected by the number of transactions in the system. When the number of transactions is 1, the running time of *Trace()* is $10^{-3}$ seconds, but when the number of transactions is $10^6$, its running time reaches 3.45 seconds.

As time goes on, there will be more and more nodes in *SescfDapp*, resulting in more and more transactions. According to the experimental results, (i) the number of nodes affects the time cost of *Tender()* and *Compute()*, but when the number of nodes reaches the order of millions, the running time of *Tender()* is only $7.44 \times 10^{-1}$ seconds and that of *Compute()* is 4.78 seconds, and (ii) the number of transactions affects the time cost of *Trace()*. When the number of transactions reaches the order of millions, it can be observed that the running time of *Trace()* is 3.45 seconds. The above time is sufficient for the provenance scenario.

*6.3. Comparison.* In Table 1, we give a comparison between SESCF and the existing blockchain-based supply chain systems in terms of functionalities [14, 23, 25, 29]. Although current work uses blockchain to solve problems in the

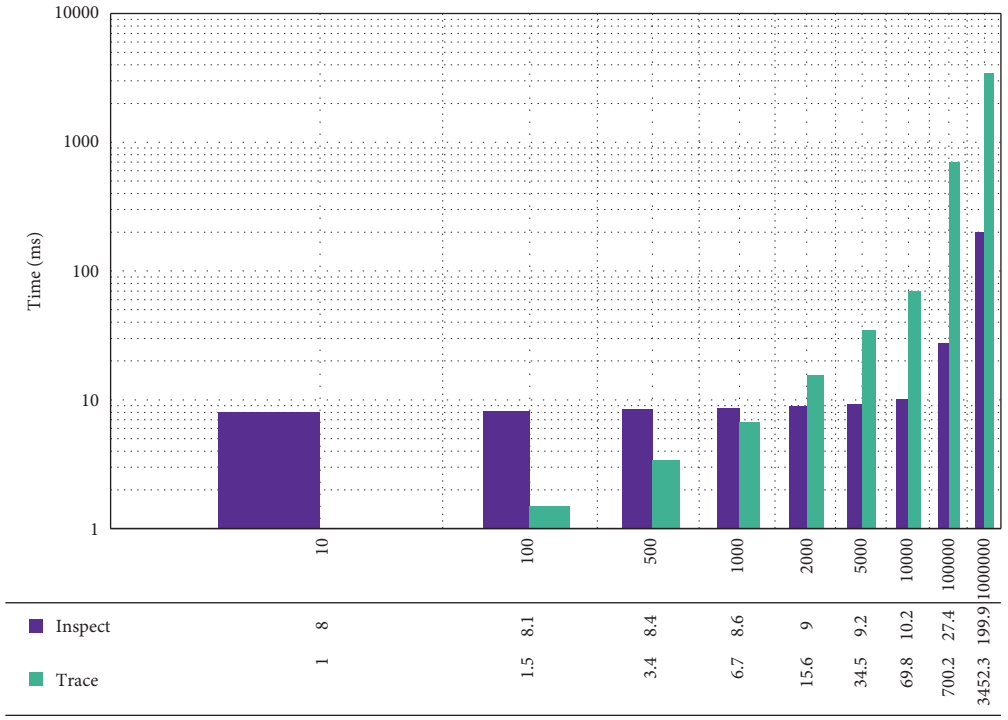


Figure 11: Time cost of inspecting and tracing goods.

Table 1: Functionality comparison.

| Functionality | SESCF | Mondal et al. [14] | Zhang et al. [23] | Tian et al. [25] | Cui et al. [29] |
|---|---|---|---|---|---|
| Symmetry of demand information | ✓ | × | × | × | × |
| Quality inspection | ✓ | ✓ | × | × | × |
| Traceability | ✓ | ✓ | ✓ | × | ✓ |
| Payment | ✓ | × | × | × | × |

supply chain, most research work has focused on only one aspect of the supply chain. As we know, the supply chain is a complex system consisting of information flow, logistics, and capital flow. Therefore, we first proposed an efficient supply chain framework based on blockchain, which guarantees the fairness and security of the information flow, logistics, and capital flow of the supply chain at the same time. In addition, we compare the efficiency of SESCF and Scheme [23]. In comparison, we only consider primary operations with significant overhead and ignore operations with low complexity.

We implement Scheme [23] based on *SescfDapp*. Figure 12 indicates the evaluation of the quality inspection and traceability function in the experiment. As we can see, the quality inspection time of both schemes increases as the number of goods and SESCF achieves more optimized performance. Both two schemes must check whether the parameters of the goods meet the requirements of production indexes. Scheme [23] needs to verify the current node information first and then upload and package the goods information. Meanwhile, SESCF does not need to perform these operations.

It should be noted that the goods ID is unique in both schemes. For the traceability function, the difference between the two schemes is the way to search using *ID*. Scheme [23] needs to traverse all transactions, while SESCF sets a keyword *total*, that is, when the number of query results equals to *total*, the smart contract will terminate. As a result, SESCF is faster for the same number of transactions. The difference in their efficiency has become more and more evident as the number of transactions increases.

## 7. Security Analysis

In this section, we first introduce the vulnerability and security issues of smart contracts which allow malicious users to exploit smart contracts. Then, how SESCF is resilient against these attacks is proposed.

*7.1. Vulnerability and Attack.* As we all know, blockchain data are secure and immutable, and the enforceability of the code on smart contracts makes it impossible for accounts to be breached. Moreover, the use of smart contracts can update in real time and reduce human intervention to improve the efficiency of the supply chain. However, if there are security bugs in smart contracts, the adversary can manipulate the execution of smart contracts to gain profit.

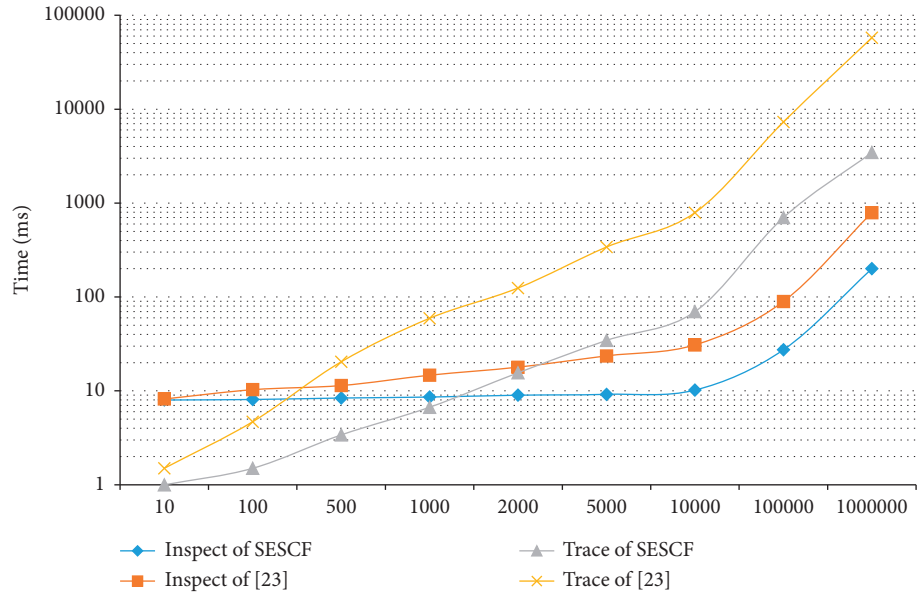


Figure 12: The performance comparison of the time cost of inspecting and tracing goods.

As a result, the security of the entire system will be undermined. Here, we introduce several vulnerabilities in smart contracts which are proposed in paper [40]: *Transaction-Ordering Dependence (TOD)*, *Timestamp Dependence*, *Mishandled Exceptions*, and *Reentrancy Vulnerability*.

*Transaction-Ordering Dependence* is a bug that may be exploited by malicious users. Usually, it takes some time for a transaction to be broadcasted and packaged in a block. Assume that an attacker $\mathscr{A}$ monitors the transaction of the corresponding contract in the network and then immediately sends a transaction to change the current contract status. By doing so, the later-released transaction may be ranked before the first-released transaction. For example, $\mathscr{A}$ submits a prize-winning quiz contract $C_{\text{quiz}}$ and a transaction $TX_a$ that promises to give generous rewards, allowing other nodes to find the correct answer. After submitting $C_{\text{quiz}}$ and $TX_a$, $\mathscr{A}$ continues to monitor the network to see if anyone submits the transaction $TX_b$ with the correct answer. Once someone submits $TX_b$, $\mathscr{A}$ immediately submits a transaction $TX_c$ that reduces the bonus before $TX_b$ is confirmed. Moreover, $\mathscr{A}$ provides a higher gas so that $TX_c$ is processed by the miner first. Finally, when the miner processes $TX_b$, the bonus becomes extremely low, and $\mathscr{A}$ can get the correct answer almost for free.

Ethereum stipulates that when a miner packs a new block $b_1$, if $b_1$'s timestamp is greater than $b_0$'s and the difference between the timestamps is less than 900 seconds, $b_1$'s timestamp is also legal. *Timestamp Dependence* means that the execution of smart contract depends on the current block's timestamp. Different timestamps will result in different execution results of the contract. Assuming the following scenario, there is a lottery contract $C_{\text{lottery}}$ that requires a value $n$ to be calculated from the current timestamp and other variables which can be known beforehand. Moreover, $C_{\text{lottery}}$ indicates that participants with the same number as $n$ will receive prizes. While mining a block, $\mathscr{A}$ tries different timestamps in advance to calculate $n$, so as to give prizes to the designated participants.

In Ethereum, smart contract $C_1$ can call another contract $C_2$ through the *send()* command. If an exception is thrown in $C_2$ (low battery or call stack limit exceeded, etc.), $C_2$ is terminated. However, exceptions in $C_2$ may not be propagated to $C_1$. The above are *Mishandled Exceptions*. For example, $\mathscr{A}$ intentionally exceeds the depth limit of the call stack to attack smart contracts. It is stated that the implementation of the Ethereum Virtual Machine limits the depth of the call stack to 1024 frames. Moreover, if $C_1$ calls $C_2$, the depth of the call stack increases by one. So, $\mathscr{A}$ submits a smart contract $C_{\text{stack}}$ and calls itself 1023 times and then sends the transaction to *Bob*. In doing so, $\mathscr{A}$ ensures that *Bob*'s call stack depth reaches 1024, which causes *Bob* to fail to send instructions.

*Reentrancy Vulnerability*, just as its name implies, is that the adversary can repeatedly enter the smart contract. In Ethereum, smart contracts can call other external contracts's codes. Consider the following contract that sends *ETH* to an external address. This function requires calling the code of the external contract. At this time, by hijacking external calls, $\mathscr{A}$ can repeatedly enter the *transfer()* function of the contract to transfer funds.

*7.2. Security Proof.* In terms of the four attack methods mentioned in Section 7.1, this section proposes security theorems to prove that SESCF is secure and effective.

**Theorem 1.** *SESCF does not rely on Transaction-Ordering.*

*Proof.* In SESCF, $P$ submitted $T_{\text{tender}}$ and reached an agreement with $S_{\text{win}}$ through a competition mechanism. The probability of reducing $S_{\text{win}}$'s contract rewards by releasing another transaction to change the current contract status is

nearly impossible. On the one hand, $T_{\text{tender}}$ is cosigned by $P$ and $S_{\text{win}}$ and broadcasted by $S_{\text{win}}$. On the other hand, once $T_{\text{tender}}$ is broadcast, C < $P$, $S_{\text{win}}$ > is opened immediately. $P$ is required to release coins not less than $T_{\text{tender}}$.price, and $S_{\text{win}}$ is required to release coins not less than $T_{\text{tender}}.LD$ into C < $P$, $S_{\text{win}}$ >. Furthermore, $P$ and $S_{\text{win}}$ will automatically pay $T_{\text{tender}}.PD$, $T_{\text{tender}}$.balance, and $T_{\text{tender}}.LD$ according to $S_{\text{win}}$'s delivery status.

**Theorem 2.** *SESCF does not rely on the timestamp of block.*

*Proof.* In the competition mechanism, $P$ selects the supplier with the lowest bid price as $S_{\text{win}}$ by comparing $S_1$.price, ⋯, $S_N$.price. Consequently, there is no situation in which $P$ selects a designated supplier for a transaction by continuously calculating the block's timestamp. In addition, the bids of $S_1, \ldots, S_N$ are encrypted by $pk_P$, so that the bids of $S_1, \ldots, S_N$ are independent of others'.

**Theorem 3.** *SESCF does not cause Mishandled Exceptions and Reentrancy Vulnerability.*

*Proof.* The exception in the callee's contract is not propagated to the caller can cause Mishandled Exceptions. Besides, in Ethereum, when a contract calls another contract, the current transaction waits for the call to finish. This issue can lead to Reentrancy Vulnerability when a transaction makes use of the intermediate state of the caller. It is quite clear that these two bugs are caused by calls. In SESCF, Competitive bidding(), Inspect(), Trace(), and Payment() are independent of each other, i.e., there is no calling relationship between them.

## 8. Conclusion

We proposed SESCF, a secure and efficient supply chain framework. As far as the authors' knowledge, it is the first system to guarantee the fairness and security of information flow, logistics, and capital flow in the supply chain system simultaneously by exploiting blockchain. Due to space limitation, since the scenario between retailers and producers is similar to that between producers and suppliers, only the latter has been illustrated in this paper. Specifically, we provided the algorithm flow and discussed the smart contracts in detail. Moreover, simulation was performed, and the results showed that the deployment and execution of SESCF only requires a certain amount of gas, which indicates the feasibility of SESCF. Based on this, we developed an efficient blockchain-based supply chain system *SescfDapp*. In the end, we evaluated SESCF and proved that it is robust and secure for entities acting maliciously.

In this paper, we put multiple types of transactions in a single chain. In reality, this may reduce the search efficiency. It is an interesting problem how to extend our framework into a multichain structure, that is, we will consider putting different types of transactions on different side chains in the future. By doing so, not only the search efficiency but also the system throughput will be improved.

## Data Availability

The data used to support the findings of this study are included within the article.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported in part by the National Key Research and Development Program of China (Grant no. 2020YFA0712300), in part by the National Natural Science Foundation of China (Grant no. 61632012), and in part by the Peng Cheng Laboratory Project of Guangdong Province (Grant no. PCL2018KP004).

## References

[1] D. M. Lambert and M. C. Cooper, "Issues in supply chain management," *Industrial Marketing Management*, vol. 29, no. 1, pp. 65–83, 2000.

[2] R. M. Monczka, R. B. Handfield, L. C. Giunipero, and J. L. Patterson, *Purchasing and Supply Chain Management*, Cengage Learning, Boston, MA, USA, 2015.

[3] R. Angeles, "Rfid technologies: supply-chain applications and implementation issues," *Information Systems Management*, vol. 22, no. 1, pp. 51–65, 2005.

[4] A. Alahmadi and X. Lin, *Towards Secure and Fair IIoT-Enabled Supply Chain Management via Blockchain-Based Smart Contracts*, pp. 1–7, Shanghai International Commerce Center Park, Shanghai, China, 2019.

[5] F. Qin, F. Mai, M. J. Fry, and A. S. Raturi, "Supply-chain performance anomalies: fairness concerns under private cost information," *European Journal of Operational Research*, vol. 252, no. 1, pp. 170–182, 2016.

[6] S. Choi and P. R. Messinger, "The role of fairness in competitive supply chain relationships: an experimental study," *European Journal of Operational Research*, vol. 251, no. 3, pp. 798–813, 2016.

[7] OECD, "Trends in trade in counterfeit and pirated goods," 2019, http://oecd.org/governance/risk/trends-in-trade-in-counterfeit-and-pirated-goods-g2g9f533-en.htm.

[8] I. Eyal, "Blockchain technology: transforming libertarian cryptocurrency dreams to finance and banking realities," *Computer*, vol. 50, no. 9, pp. 38–49, 2017.

[9] M. Crosby, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: beyond bitcoin," 2016.

[10] R. Beck, J. S. Czepluch, N. Lollike, and S. Malone, *Blockchain-the Gateway to Trust-Free Cryptographic Transactions*, ECIS, London, UK, 2016.

[11] S. A. Abeyratne and R. P. Monfared, "Blockchain ready manufacturing supply chain using distributed ledger," *International Journal of Research in Enginerring and Technology*, vol. 5, pp. 1–10, 2016.

[12] Ministry of Industry and Information Technology of the People's Republic of China, *China's Blockchain Industry White Paper*, Beijing, China, 2018, http://www.miit.gov.cn/Searchweb/news.jsp.

[13] R. Want, "An introduction to RFID technology," *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 25–33, 2006.

[14] S. Mondal, K. P. Wijewardena, S. Karuppuswami, N. Kriti, D. Kumar, and P. Chahal, "Blockchain inspired RFID-based information architecture for food supply chain," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5803–5813, 2019.

[15] Bitcoin community, bitcoin wiki: payment channels, https://en.bitcoin.it/wiki/Paymentchannels.

[16] C. Decker and R. Wattenhofer, "A Fast and scalable payment network with bitcoin duplex micropayment channels: stabilization, safety, and security of distributed systems," 2015.

[17] G. Malavolta, P. Moreno-Sanchez, and A. Kate, "Concurrency and privacy with payment-channel networks," in *Proceedings of the CCS'17 Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 455–471, Dallas, TX, USA, 2017.

[18] J. Poon and T. Dryja, "The bitcoin lightning network: scalable off-chain instant payments," 2016.

[19] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2018, http://satoshinakamoto.me/2008/11/01/bitcoin-p2p-e-cash-paper/%20and%20http://www.bitcoin.org/bitcoin.pdf.

[20] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, *MedRec: Using Blockchain for Medical Data Access and Permission Management*ÖBB, Vienna, Austria, 2016.

[21] H. Zhang and E. Deng, "Smart contract for secure billing in ride-hailing service via blockchain: peer-to-peer networking and applications," 2018.

[22] K. Biswas and V. Muthukkumarasamy, *Securing Smart Cities Using Blockchain Technology*HPCC/SmartCity/DSS, Sydney, Australia, 2016.

[23] X. Zhang, P. Sun, J. Xu et al., "Blockchain-based safety management system for the grain supply chain," *IEEE Access*, vol. 8, pp. 36398–36410, 2020.

[24] Y. Fu and J. Zhu, "Big production enterprise supply chain endogenous risk management based on blockchain," *IEEE Access*, vol. 7, pp. 15310–15319, 2019.

[25] F. Tian, "An agri-food supply chain traceability system for China based on RFID & blockchain technology," in *Proceedings of the ICSSSM*, Kunming, China, 2016.

[26] K. Biswas, V. Muthukkumarasamy, and W. L. Tan, "Blockchain based wine supply chain traceability system," in *Proceedings of the Future Technologies Conference*, pp. 1–7, Vancouver, Canada, 2017.

[27] K. Toyoda, P. T. Mathiopoulos, I. Sasase, and T. Ohtsuki, "A novel blockchain-based product ownership management system (POMS) for anti-counterfeits in the post supply chain," *IEEE Access*, vol. 5, pp. 17465–17477, 2017.

[28] J.-H. Lee and M. Pilkington, "How the blockchain revolution will reshape the consumer electronics industry [future directions]," *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 19–23, 2017.

[29] P. Cui, J. Dixon, U. Guin, and D. Dimase, "A blockchain-based framework for supply chain provenance," *IEEE Access*, vol. 7, pp. 157113–157125, 2019.

[30] T. Bocek, B. Rodrigues, T. Strasser, and B. Stiller, "Blockchains everywhere—a use-case of blockchains in the pharma supply-chain," in *Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, Portugal, 2017.

[31] Y. M. Kow and X. Ding, "Hey, i know what this is!: cultural affinities and early stage appropriation of the emerging bitcoin technology," in *Proceedings of the the 19th International Conference on Supporting Group Work*, pp. 213–221, Sanibel Island, FL, USA, 2016.

[32] Ethereum, https://www.ethereum.org.

[33] J. Landt, "The history of RFID," *IEEE Potentials*, vol. 24, no. 4, pp. 8–11, 2005.

[34] C. Amador, J.-P. Emond, and M. C. D. N. Nunes, "Application of RFID technologies in the temperature mapping of the pineapple supply chain," *Sensing and Instrumentation for Food Quality and Safety*, vol. 3, no. 1, pp. 26–33, 2009.

[35] E. Smits, J. Schram, and M. Nagelkerke, "Development of printed RFID sensor tags for smart food packaging," in *Proceedings of the 14th International Meeting on Chemical Sensors*, pp. 403–406, Nuremberg, Germany, 2012.

[36] J. Virtanen, L. Ukkonen, T. Björninen, and L. Sydänheimo, "Printed humidity sensor for UHF RFID systems," in *Proceedings of the 2010 IEEE Sensors Applications Symposium (SAS)*, pp. 269–272, Limerick, Ireland, 2010.

[37] K. H. Eom, K. H. Hyun, S. Lin, and J. W. Kim, "The meat freshness monitoring system using the smart RFID tag," *International Journal of Distributed Sensor Networks*, vol. 2014, pp. 1–9, 2014.

[38] R. A. Potyrailo, N. Nagraj, Z. Tang, F. J. Mondello, C. Surman, and W. Morris, "Battery-free radio frequency identification (RFID) sensors for food quality and safety," *Journal of Agricultural and Food Chemistry*, vol. 60, no. 35, pp. 8535–8543, 2012.

[39] Bitcoin community, bitcoin wiki: confirmation, https://en.bitcoin.it/wiki/Confirmation.

[40] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 254–269, Vienna, Austria, 2016.