Master's Thesis

Title

Protection of Confidential Information in Supply Chain System Based on Public Permissionless Blockchain

Supervisor

Professor Masayuki Murata

Author

Takio Uesugi

February 3rd, 2022

Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

Protection of Confidential Information in Supply Chain System Based on Public Permissionless Blockchain

Takio Uesugi

Abstract

Rapid growth of supply chains has caused serious problems, such as counterfeit products and delays in product tracking. To remedy these problems, it is an urgent issue to ensure the product traceability at a high level. Recently, blockchain-based supply chain systems have been proposed. These systems use a blockchain as a shared database to manage distribution information of the supply chain. Blockchain is a transparent and tamper-resistant database, which ensures the integrity of the distribution information. Thanks to these features, blockchain-based supply chain systems can track the product distribution, thereby ensuring the traceability. Considering the entry of new businesses and the growth of secondary markets, it is desirable to design a system to allow anyone to register and verify distribution information at any time. To realize such a supply chain system, it is the optimal choice to use a public permissionless blockchain. However, this supply chain system discloses confidential information, such as business-to-business relationships and transactions information between individuals. Therefore, in this thesis, we propose a supply chain system that can protect confidential information and ensure traceability, using a public permissionless blockchain. We realize the protection of confidential information by hiding the distribution information via attribute-based encryption. We also use zero-knowledge proof to ensure the distribution between legitimate owners and recipients while hiding ownership information. In addition, the proposed system can express aggregation, assembling and packing, hence track relationship between a product and its part as well as the distribution in a grouped unit such as cardboard boxes and shipping containers. We implement the proposed system with smart contracts and evaluate its usage cost based on transaction fees. The result shows the transaction fee is at most $2.6 \times 10^6 gas$ units per distribution. Although the price of the transaction fee varies depending on the blockchain, the lowest fee is 4.5 USD, which indicates that the proposed system can be applied to various products.

Keywords

Public Permissionless blockchain

Confidential protection

Attribute-based encryption

Zero-knowledge proof

Supply chain

Traceability

Ethereum

Contents

List of Figures

List of Tables

1 Introduction

In recent years, supply chains have become increasingly larger and more complex. This has led to serious problems, especially regarding product traceability. For example, the amount of international trade of counterfeit products is estimated to be 446 billion USD in 2019, which is equivalent to 2.5% of world trade [1]. Furthermore, since it was difficult to track ingredients contaminated with *Escherichia coli*, there was an outbreak of food poisoning that spread to a total of 60 people across 14 states in the U.S. [2]. These problems are due to the difficulty in verification of the product's manufacturer and fast product tracking. In the supply chain, there are multiple businesses even in the primary distribution from the manufacturer to the consumer. In addition, secondary markets have also been growing in recent years due to the ease of transactions using apps to trade used stuff. Against this background, it has become increasingly difficult to manage supply chain information. As a provisional measure, each business has managed distribution information by building its own database. However, each business may use different identifiers and data structures to manage the same product. It cannot be denied that this structure may lead to delays and distortions of the information when someone tracks a product. In other words, product traceability is not ensured at a high level.

One of the solutions to these problems is to unify the management of supply chain information among multiple businesses. As a system to realize it, blockchain-based supply chain systems have been proposed [3–7]. In these systems, product distribution information, i.e., ownership history, is recorded in a blockchain. A blockchain is a decentralized and time-series ledger. Multiple nodes in the blockchain network independently verify and update the data based on a common logic, and then store the data in the blockchain. By verifying the stored data against each other, invalid data is automatically detected and eliminated. Thus, the data on the blockchain is extremely difficult to tamper with and has high availability. Accordingly, the distribution information stored in the blockchain cannot be lost due to invalid changes or system failures. It is also possible to prevent the recording of invalid distribution information by using smart contracts to set appropriate conditions for recording and updating distribution information.

There are processes such as assembly and packing in the supply chain. In order to ensure high traceability, the supply chain system should be able to express the operation of aggregation and dis-aggregation. Note that aggregation means to group multiple things into one thing, and dis-aggregation, which is the inverse process of aggregation, means to retrieve the multiple things from the one thing grouped by aggregation. Aggregation makes it possible to track manufacturing processes, thus improving product safety. Suppose that a component of a certain product has a defect. The relationship between the product and the component is recorded through aggregation. In this case, it is possible to quickly identify the product with the defective component, thereby preventing the spread of the defective product. Aggregation and dis-aggregation are also necessary from the perspective of efficiency. When products are distributed in the supply chain, they are packed in cardboard boxes, etc., and then further loaded onto trucks or shipping containers. This means that multiple products are distributed together as a single unit. If the operation of aggregation is not available, the distribution of all the products must be managed one by one, which can be difficult in practice. If it is available, distribution of multiple products can be managed in a grouped unit.

Supply chains are likely to become increasingly larger and more complex as new businesses enter markets and secondary markets flourish. In order to cope with this situation, it is desirable to have an open system, i.e., a system where anyone can register and verify distribution information. This provides a variety of benefits. In such a system, new businesses can freely register distribution information, which does not hinder the establishment of new distribution relationships. General consumers can also register distribution information, making it possible to track transactions between individuals in secondary markets. Furthermore, it is possible to add extra value to products because general consumers can view the distribution information of supply chains. For example, if you can confirm the origin of crops or the manufacturer of products, you can be sure of its safety. Blockchain can be classified as public or private in terms of data access permissions, and as permissionless or permissioned in terms of data update permissions [8]. A blockchain that can be accessed and updated by anyone is a public permissionless blockchain. For the supply chain system, a public permissionless blockchain is desirable.

However, the use of a public permissionless blockchain brings new challenges. One of the most significant challenges is to prevent the leakage of confidential information [9].

A public permissionless blockchain makes all data recorded on it public. That is, even information that should be kept secret will be made public. To be specific, there are the following problems in a supply chain system. Businesses invest significant efforts in investigating business partners and building distribution relationships in order to lower purchase prices or achieve rapid distribution. If this information were made public, it would threaten the competitive advantage of the businesses, as their competitors would be able to identify and establish distribution relationships without investment. As another example, in secondary markets, transaction information between individuals and product ownership information can be identified. Therefore, it is necessary to protect distribution information, such as business-to-business relationships and transactions information between individuals, as confidential information.

In this thesis, we propose a supply chain system that provides both traceability and protection of confidential information, using a public permissionless blockchain. We define *traceability* as the characteristic that anyone can verify the product manufacturer and distribution by verifying the public information and the manufacturer can track its own product distribution. We also define protection of confidential information as data being hidden and not revealed to anyone except those who have the permission to browse it. The target of the protection of confidential information is product ownership information in supply chains. We use attribute-based encryption and zero-knowledge proof to protect confidential information. Attribute-based encryption ensures that information is only disclosed to those who have the permission to browse it. Zero-knowledge proof is to prove that one knows a certain knowledge without revealing any information about that knowledge. We use zero-knowledge proof to prove the knowledge that one is the product owner and has the right to distribute it, without revealing it. This allows for the hiding of distribution information while preventing unauthorized distribution. Therefore, we can realize a supply chain system that satisfies both traceability and protection of confidential information.

The rest of this thesis is organized as follows. Section 2 introduces related works. The design goals of the proposed system are described in Section 3. Section 4 explains the proposed method that can protect confidential information and ensure traceability and Section 5 evaluates it. Lastly, our conclusions and future work are presented in Section 6.

2 Related Work

2.1 Blockchain

2.1.1 Bitcoin and Blockchain

Bitcoin [10] is the world's first cryptocurrency, which started the operation in 2009. Bitcoin has attracted a great deal of attention because it is a system that allows the transfer of data with monetary value without the need for a central administrator. The method proposed to realize this system is blockchain.

Blockchain has a data structure in which a unit of data, called a block, is connected like a chain by having a hash value of the previous block. The block consists of a block header and transaction data. The block header consists of a hash value, a nonce and a timestamp. The hash value is the one generated from the previous block header to point to the previous block, thereby realizing the chain structure. The timestamp is the time when the block is created. The nonce is the value required to execute the consensus algorithm, which is explained later. Even if an attacker tries to tamper with a transaction, the hash value of the block containing the transaction changes, causing a discrepancy of the hash value the next block has. In other words, the chain structure of the blockchain has been broken. Thus, transaction tampering can be easily detected.

A transaction is the smallest unit of data in a blockchain. In the case of Bitcoin, a transaction represents a history of asset transfers. The transaction is added a digital signature by the creator of the transaction, thus it provides validity and non-repudiation. Blockchain is managed in a decentralized manner through P2P networks. The data of the blockchain is shared by all nodes in the blockchain network. Thus, it provides zero-downtime and semi-permanent operation.

Blockchain needs a mechanism to synchronize data across the entire blockchain network. Consensus algorithms realize this mechanism, and Bitcoin utilizes an algorithm called Proof of Work (PoW). The only block generated according to PoW is adopted as the valid one. Block creators, called miners, calculate a cryptographic hash value from the nonce, the previous block's hash value and the current block's transaction data. The hash value needs to be less than or equal to a target value. The target value is automatically adjusted so that the nonce is found once every 10 minutes in the environment where all miners calculate the hash simultaneously. Since cryptographic hash values have preimage resistance, the miners need to repeat calculating hashes while changing the nonce in small increments. Thus, a huge amount of computation power is required to generate a block. Furthermore, Bitcoin adopts the so-called longest chain rule where the chain with the most block is adopted in the blockchain network. In order to tamper with a transaction, an attacker would need to recalculate it again for all blocks from that point to the present. This would be impossible if the attacker did not possess overwhelming computing power. In reality, however, such an attacker does not exist. These mechanisms provide the blockchain with tamper-resistance.

The blockchain used in Bitcoin is a public permissionless blockchain. Blockchains can be classified into public or private in terms of data access permissions, and permissionless or permissioned in terms of data update permissions. Public blockchains allow anyone to access the data, while private blockchains allow only verified participants to reference the data. Permissionless blockchains allow anyone to update the data, while permissioned blockchains allow only verified participants to update the data. Considering the scalability of supply chains and the participation of general consumers, we use public permissionless blockchain in the proposed system.

2.1.2 Turing-Complete Blockchain

Bitcoin has a stack-based programming language, but it is not Turing-complete. In order to perform more complex and general-purpose operations, blockchains with a Turing-complete execution environment have emerged. Such blockchains can execute arbitrary programs, and this mechanism is called a smart contract.

Ethereum [11] is one of the leading blockchains with smart contracts. There are two types of accounts in Ethereum. One type of account, EOA (Externally Owned Accounts), has a pair of public and private keys. EOA is represented by an address, which is a value obtained from the hash value of the public key. Transactions can be issued by signing it with the private key. The transactions are used to transfer ETH, the Ethereum base currency, and also used to send messages to execute smart contracts. The other account, CA (Contract Accounts), manages arbitrary programs and storage. When it receives a message from another CA or EOA, it executes a specific function in the program using the message contents as arguments. The result of the program execution is stored in the CA's storage or used as a message to another CA. Since this account is managed by the blockchain, the program can be executed in a decentralized manner.

In this thesis, we use Ethereum, a public permissionless blockchain with smart contracts.

2.2 Blockchian-based Supply Chain Systems

Several blockchain-based systems have been proposed to improve the product traceability in supply chains.

Some researchers have proposed supply chain systems based on a permissioned blockchain. IBM Food Trust [12] is designed to secure food traceability. Bryatov et al. [13] proposed a system to prevent the distribution of counterfeit drugs in pharmaceutical supply chains. Agrawal et al. [14] proposed a system to prevent the spread of defective drugs and to recall them. Maouchi et al. [15] proposed a traceability system for preserving privacy by setting appropriate permissions for browsing data. However, only verified participants can use these systems. This makes it difficult to widely promote supply chain systems and hinders the unified management of distribution information. Thus, it is difficult to secure product traceability at a high level. In addition, since general consumers cannot participate in these systems, it is practically impossible to apply the system to secondary markets.

Supply chain systems using a public permissionless blockchain have also been proposed. Toyoda et al. [3] proposed a system for using blockchain to manage product ownership transfers to prevent the distribution of counterfeit products in the post supply chain. Kim et al. [4] proposed a system for tracking products from the materials stage through repeated consumption and production of traceable resource units. Huang et al. [5] proposed a system for applying off-chain technologies to food-supply chains, which feature high-frequency distribution. Santos et al. [6] proposed a system to guarantee food safety by representing food with a non-fungible token. Westerkamp et al. [7] proposed a system that can trace manufacturing processes by defining how tokens are created. These supply chain systems are expected to become widely used because anyone can freely participate in the supply chain and browse information. This promotes uniform management of distribution information and contributes to securing product traceability. However, none of these methods consider confidential information, so transaction and ownership information will be widely disclosed. In addition, since the operations such as aggregation and dis-aggregation are not available, these systems inefficiently manage the supply chain.

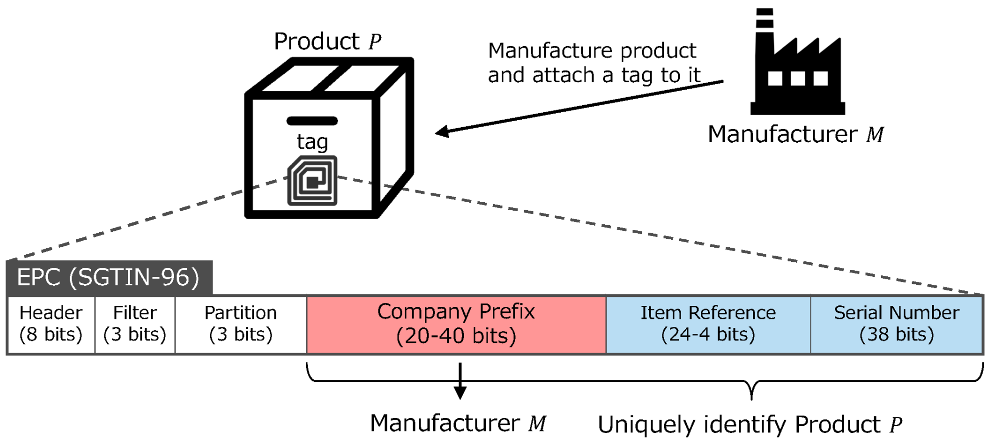


Figure 1: Information recorded on a tag attached to the product.

Table 1: Permission control of the proposed method.

	Manufacturer	Current owner	Next owner	Others
Manufacturing a product	✓			
Packing products		✓		
Unpacking a container		✓		
Shipping a product/container		✓		
Receiving a product/container			✓	

3 Design Goals

This section introduces the system model assumed in the proposed method. Then we explain the design goals in terms of controls of product distribution and information access.

3.1 System Prerequisites

We assumed that a tag such as an RFID is attached to each product. As Figure 1 shows, the tag includes an electronic product code (EPC) in the SGTIN-96 format. SGTIN-96 includes a company prefix that identifies the product manufacturer and a serial number that identifies the product. The proposed system uses the EPC to manage product information. We assume that manufacturers can freely create EPCs and write them to the tags, and that EPCs are correct, that is, the manufacturer's company prefix and the prod-

Table 2: Access control of the proposed method.

	Manufacturer	Previous owner	Current owner	Next owner	Others
Browse manufacturer	✓	✓	✓	✓	✓
Browse previous owner	✓	✓	✓		
Browse current owner	✓	✓	✓	✓	
Browse next owner	✓		✓	✓	
Browse ownership history	✓				

uct's unique serial number are correctly recorded. We also assume that EPCs attached to products cannot be altered by replacing or tampering with the tag.

The systems proposed in [3–7] manage product ownership information with blockchain addresses. In these systems, it can be assumed that a certain mechanism is provided to link blockchain addresses with real-world entities in order to track product owners. Therefore, we also assume that blockchain addresses are linked to real-world entities by the same mechanism as public key certificates.

3.2 Product Distribution Control

The proposed system tracks product distribution by processing enrollments and repeating shipment and receipt processes for the product. In the distribution process, products may be packed and unpacked. To prevent unauthorized product distribution, we control permissions for each process as shown in Table 1. The table shows the processes that can be executed by the corresponding parties. In the table, "manufacturer" is the party that manufactured the product, "current owner" is the party that currently owns the product, and "next owner" is the party that will receive the product from its current owner.

3.3 Information Access Control

In the proposed system, we protect confidential information by restricting information browsing. We control information access as shown in Table 2. "Previous owner" is the party that shipped the product to the current owner, and the rest is the same as in Table 1.

The proposed system allows anyone to browse product manufacturers at will, allowing them to confirm that products were manufactured by a legitimate manufacturer and preventing distribution of counterfeit products. However, only manufacturers can browse ownership histories, because the manufacturer is responsible for its products. Manufacturers can immediately identify owners to recall products and prevent their further distribution when problems occur. No other parties have reasonable grounds for browsing ownership histories. Since the previous owner and the current owner were in a relationship to distribute a product, they can verify identity with each other. On the other hand, the previous owner and the next owner cannot verify the identity of each other because they are not in a relationship of product distribution. Thus, the proposed system protects confidential information through appropriate permission management for browsing that information.

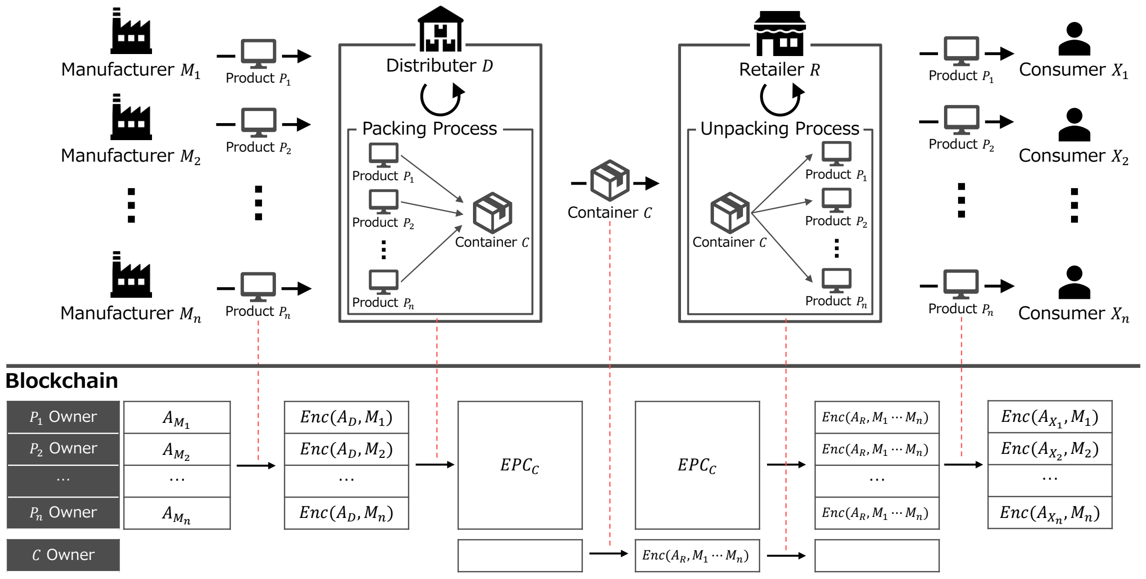


Figure 2: Product distribution using the proposed system

4 Proposed System for Protection of Confidential Information

In this thesis, we propose a system to protect confidential information of product ownership. As mentioned above, the proposed system assumes that real-world entities can be uniquely identified by their blockchain addresses. Therefore, the proposed system records the encrypted blockchain addresses in the blockchain. This hides the blockchain addresses of product owners and ensures the confidentiality of the ownership information. The manufacturer can obtain the blockchain addresses by decrypting the encrypted ones, and track the distribution of the product. Product owners and recipients share a secret token and demonstrate that they know it using zero-knowledge proof. The proposed system thereby guarantees distribution between correct owners and recipients while concealing blockchain addresses.

4.1 Hiding Ownership Information using Attribute-Based Encryption

The proposed system hides the ownership and distribution relationships information by recording the encrypted blockchain addresses. Only the manufacturer can browse the ownership information of its products. Thus, if there is a problem with a product, the manufacturer can immediately identify the owner and recall the product to prevent further distribution. The proposed system manages not only single products but also containers like C in Figure 2. Therefore, we use attribute-based encryption in the proposed system. Attribute-based encryption is a cryptographic scheme that allows multiple users to decrypt by specifying a set of attributes [16]. Attribute-based encryption consists of the following algorithms;

- Setup: Generate a public parameter pk and a master secret key msk based on a security parameter.
- KeyGen: Generate a secret decryption key sk_U from the master secret key msk and a set of attributes U.
- Encryption: Output a ciphertext ct by taking a random number s, a message m, a policy W and the public parameter pk as input.
- Decryption: If a set of attributes U satisfies the policy W, output message m by taking the secret decryption key sk_U and the ciphertext ct as input.

Let us illustrate with the example in Figure 2. Assume that manufacturer M_i ($i \in \{1..n\}$) has been assigned attribute u_i and owns the decryption key sk_{u_i}. Ciphertext $Enc(A_D, M_1)$ is generated as $m = A_D, W = \{u_1\}$. Manufacturer M_1 can decrypt the ciphertext and obtain distributor D's address A_D as plaintext because attribute u_1 satisfies the policy W. Ciphertext $Enc(A_R, M_1 \cdots M_n)$ is generated as $m = A_R, W = \{u_1 \vee u_2 \vee \cdots \vee u_n\}$. In the case of a container that contains multiple products, the attributes of the manufacturer of the included products are specified and encrypted, as in $W = \{u_1 \vee u_2 \vee \cdots \vee u_n\}$. Manufacturer M_1 can also decrypt the ciphertext and obtain retailer R's address A_R as plaintext because attribute u_1 satisfies the policy W. Manufacturers $M_2, \cdots$, and M_n can also decrypt ciphertext $Enc(A_R, M_1 \cdots M_n)$ as well. In the example in Figure 2, for simplicity, we use the recipient's address as the message m. In practice, however, we use the exclusive-OR of the owner's address and the recipient's address as the message m. In other words, in the distribution from manufacturer M_1 to distributor D, the proposed system records ciphertext $Enc(A_{M_1} \oplus A_D, M_1)$ instead of ciphertext $Enc(A_D, M_1)$. Moreover, a random number s is used for encryption. We use

s as a secret token. In the proposed system, the owner specifies the recipient with that ciphertext. By using a random number s as a secret token, the owner and the recipient can generate the same ciphertext. Therefore, the recipient can verify that the owner has not specified an invalid ciphertext.

In the proposed system, since only manufacturers decrypt the ciphertexts, we assign attributes only to the manufacturers. Attribute-based encryption can assign any string as an attribute. Thus, a manufacturer's name, etc. can be used as an attribute. However, if there are duplicates, there is a possibility of unexpected decryption. The attribute, therefore, needs to be a string unique to the manufacturer. We apply the manufacturer's company prefix used in EPC as an attribute.

Setup and KeyGen must be performed by a trusted third party. As described in detail in Section 4.4.2, we assume an administrator who manages manufacturers' information. Secret decryption keys are also manufacturers' information. Therefore, we assume that the administrator is in charge of Setup and KeyGen.

4.2 Ownership Authentication using Zero-Knowledge Proof

The proposed system allows the owner to execute processes on a product by authenticating a secret token only known to the legitimate owner. To authenticate this secret token, we use a zero-knowledge proof. As we use a zero-knowledge proof, there is no information that is disclosed except that the process was executed by the legitimate owner.

Although there are several zero-knowledge proof methods, we use zk-SNARKs in the proposed system. Zk-SNARKs is a non-interactive zero-knowledge proof method, which is widely used in blockchains in terms of proof size, execution time and computational complexity. Zk-SNARKs allows us to specify the knowledge to be proved in the form $f(x) = y$. The x is the knowledge that a prover wants to prove, and the prover generates proof information for the result of giving x to the function f. We use this proof information and y to verify that $f(x) = y$ is satisfied. In the proposed system, we use a cryptographic hash function for the function f, and the x is a secret token. Therefore, y is the hash value of the secret token. Zk-SNARKs requires a trusted setup between the prover and the verifier, where the proving and verification keys are generated. The trusted setup needs to be given a function f as input, and the keys depend on the function f. Therefore, the

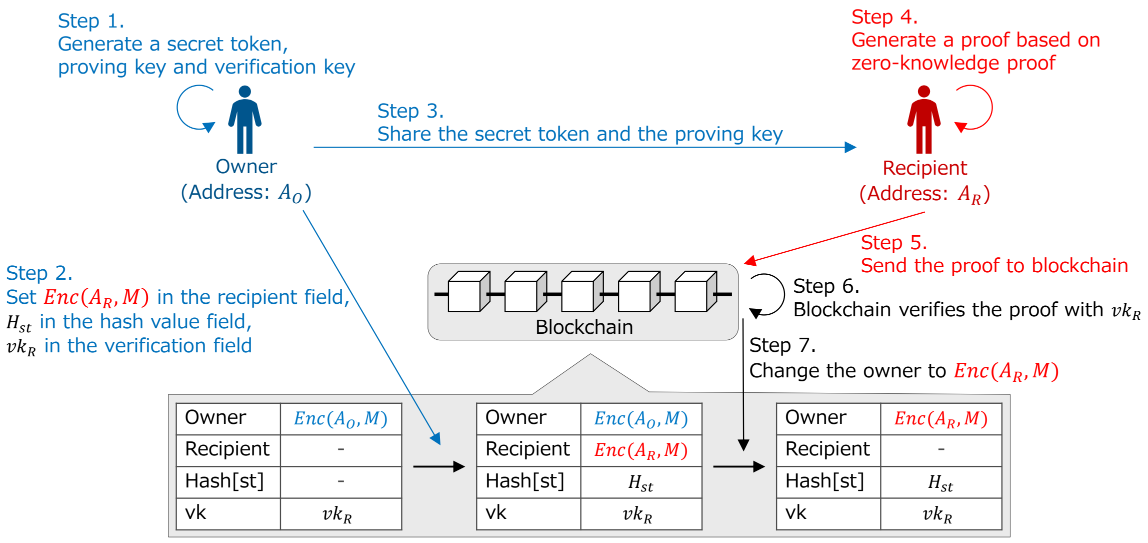

Step 1.
Generate a secret token, proving key and verification key
Owner
(Address: A_O)
Step 3.
Share the secret token and the proving key
Step 4.
Generate a proof based on zero-knowledge proof
Recipient
(Address: A_R)
Step 5.
Send the proof to blockchain
Step 2.
Set $Enc(A_R, M)$ in the recipient field,
H_{st} in the hash value field,
vk_R in the verification field
Blockchain
Step 6.
Blockchain verifies the proof with vk_R
Step 7.
Change the owner to $Enc(A_R, M)$
Owner $Enc(A_O, M)$
Recipient -
Hash[st] -
vk vk_R
Owner $Enc(A_O, M)$
Recipient $Enc(A_R, M)$
Hash[st] H_{st}
vk vk_R
Owner $Enc(A_R, M)$
Recipient -
Hash[st] H_{st}
vk vk_R

Figure 3: Proving ownership in receiving process

keys cannot be diverted to prove other knowledge. The prover uses the proving key to generate a proof, and the verifier uses the verification key to verify the proof.

4.2.1 Ownership Authentication of Single Product

We describe the ownership authentication for a single product. The process that requires this ownership authentication is shipping and receiving processes. In the proposed system, the owner shares a secret token with the recipient to ensure that the shipping/receiving process is executed by the legitimate owner/recipient. The proposed system allows the owner/recipient to execute the shipping/receiving process by authenticating this secret token at runtime.

Figure 3 illustrates the overview of shipping and receiving processes. Note that M represents the manufacturer of the product in distribution.

1. The owner generates a secret token, and proving and verification keys by a trusted setup.

2. The owner records the encrypted recipient's address $Enc(A_R, M)$, the secret token's hash value H_{st} and the verification key vk_R in the blockchain.

3. The owner shares the secret token and the proving key with the recipient by a secure

method, then ships the product.

4. The recipient uses the shared proving key to demonstrate knowledge of the secret token based on a zero-knowledge proof.
5. The recipient sends the proof to the blockchain.
6. The blockchain verifies that the proof is valid using H_{st} and vk_R.
7. The blockchain updates the owner to $Enc(A_R, M)$.

The steps above ensure that the one who executes the receiving process is the legitimate recipient. In practice, when the owner executes the shipping process in step 2, the proposed system also verifies that the one who executes the shipping process is the legitimate owner. The secret token and the proving/verification keys used in the shipping process are the same as those used in the distribution with the previous owner. In zk-SNARKs, a random number is used to generate a proof. Even if the same proving key is used, the generated proof will be different than before. Therefore, we can use the same secret token and proving/verification keys. Note that since proofs that have been used once are public, the system verifies that previous proofs aren't diverted.

4.2.2 Ownership Authentication of Multiple Products

We describe the ownership authentication for multiple products. The process that requires this ownership authentication is the packing process. The system verifies that the one who executes the packing process is the owner of products to be targeted by the packing process. We use secret tokens to authenticate as well.

The following steps is a naive method for proving the ownership in the packing process. Let n be the number of products targeted in the packing process. We explain the packing process of n products as an example of the packing process executed by distributor D in Figure 2.

1. Distributor D sends n proofs to the blockchain at once.
2. The blockchain updates the ownership information for n products and a container when all proofs are successfully verified.

Table 3: The number and size of proofs in Method A/B

	The number of proofs	The size of each proofs
Method A	n	$O(1)$
Method B	1	$O(n)$

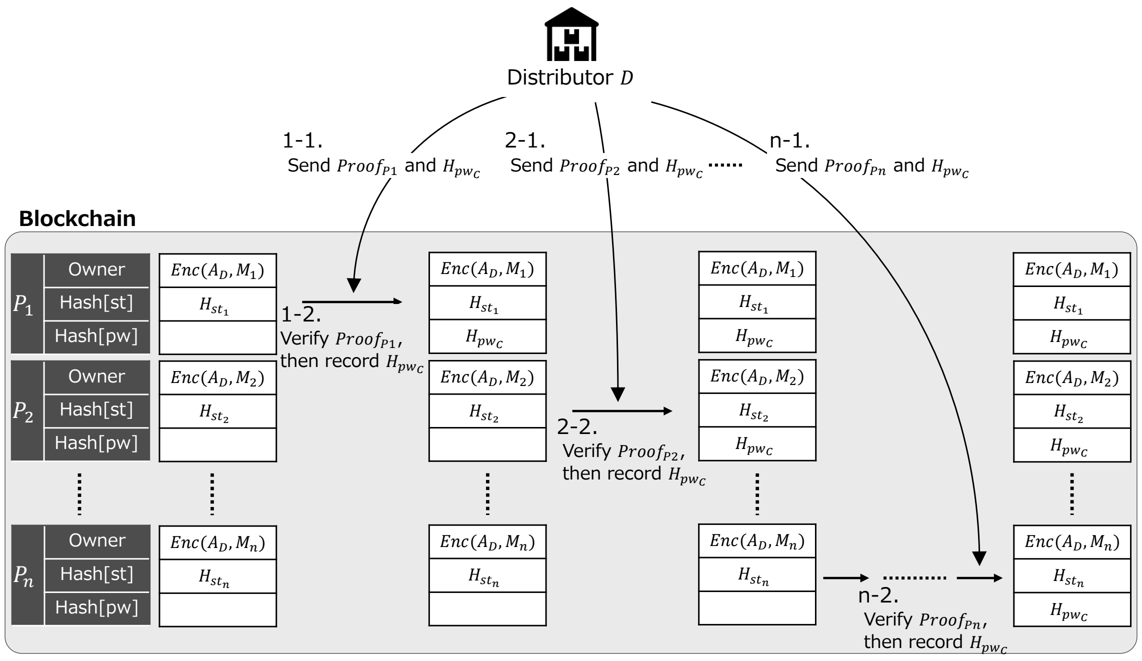
In step 1, each of the n proofs proves the ownership of each product. This method verifies n proofs and updates the ownership information for n products and one container at once. However, we found that the maximum number of n is limited to 22. In the proposed system, we use Ethereum blockchain. In Ethereum, there is an upper limit of transaction fees per block called the block gas limit. The higher the amount of data and computation required for execution, the higher the transaction fee. With this method, the transaction fee reaches the block gas limit when n is 22. It is difficult to represent a real-world supply chain with such a packing process. Therefore, we consider two methods, Method A and Method B, to increase the maximum number of n. Method A reduces the transaction fee required for each execution by splitting the execution that used to be done at once into multiple executions, thereby increasing the maximum number of n. Method B reduces the transaction fee and increases the maximum number of n by aggregating n proofs of ownership into a single aggregated proof.

Table 3 illustrates the number of proofs and proof sizes for each method. Method A requires proofs for the number of n products, thereby the number of proofs is larger than that of Method B. Method B requires only one proof, thereby the proof size is larger than that of Method A.

We discuss a trade-off between Method A and Method B based on the evaluation results in Section 5.

Method A

In Method A, distributor D prepares a password and its hash value. Before executing the packing process, distributor D proves the ownership for each of the n products independently. At this time, the proposed system records the hash value of the password so that it can be verified that this prover and the executor of the packing process match.



Distributor D
1-1. Send $Proof_{P1}$ and H_{pw_C}
2-1. Send $Proof_{P2}$ and H_{pw_C} ……
n-1. Send $Proof_{Pn}$ and H_{pw_C}
Blockchain
Owner
Hash[st]
Hash[pw]
P_1
P_2
P_n
$Enc(A_D, M_1)$
H_{st_1}
$Enc(A_D, M_2)$
H_{st_2}
$Enc(A_D, M_n)$
H_{st_n}
H_{pw_C}
1-2. Verify $Proof_{P1}$, then record H_{pw_C}
2-2. Verify $Proof_{P2}$, then record H_{pw_C}
n-2. Verify $Proof_{Pn}$, then record H_{pw_C}

Figure 4: Proving ownership in Method A

When executing the packing process, distributor D enters the password, and the proposed system compares the hash value of this password with the recorded hash value to verify the match between the executor of the packing process and the prover of ownership.

We use Figure 4 and 5 to illustrate the steps in detail. We assume that distributor D in Figure 2 has received the products from manufacturers $M_1, M_2, \cdots, M_n$. Accordingly, distributor D possesses secret tokens $st_1, st_2, \cdots, st_n$, and the blockchain records its hash value $H_{st_1}, H_{st_2}, \cdots, H_{st_n}$. In Figure 4 and 5, `Owner` represents the field that records the product owner, `Hash[st]` represents the field that records the hash value of a secret token, and `Hash[pw]` represents the field that records the hash value of a password for the packing process.

1. Distributor D prepares a password pw_C and its hash value H_{pw_C}.

2. Distributor D proves ownership of n products using secret token.

   1-1. For product P_1, distributor D generates a ownership proof $Proof_{P_1}$ using secret token st_1, then sends $Proof_{P_1}$ and hash value H_{pw_C} to the blockchain.

   1-2. The blockchain verifies $Proof_{P_1}$ using H_{st_1} and records hash value H_{pw_C} if the

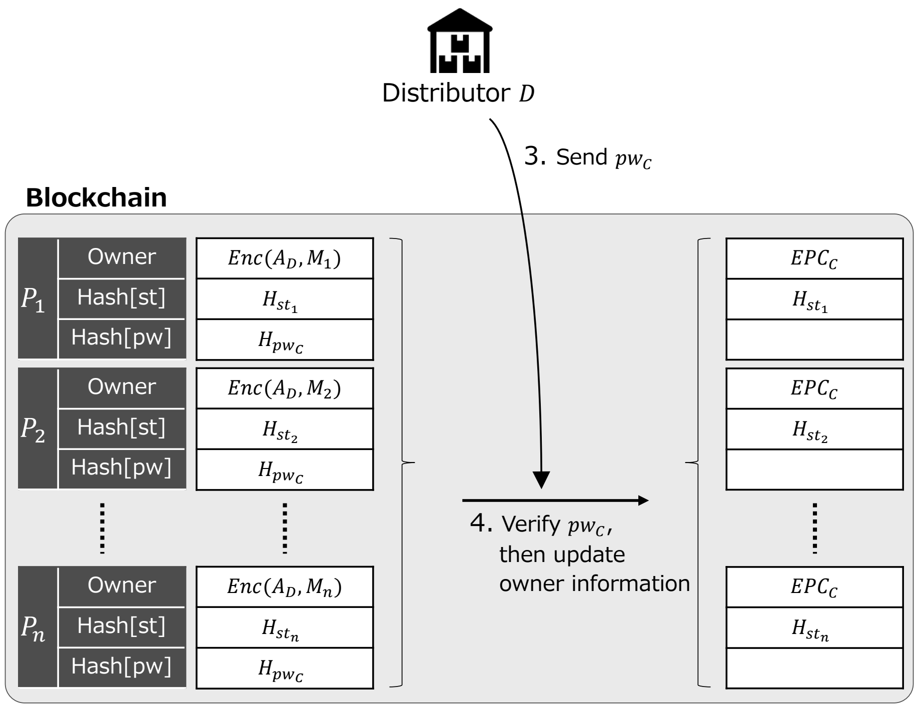


Figure 5: Updating owner information in Method A

proof is valid.

2-1. For product P_2, distributor D generates a ownership proof $Proof_{P_2}$ using secret token st_2, then sends $Proof_{P_2}$ and hash value H_{pw_C} to the blockchain.

2-2. The blockchain verifies $Proof_{P_2}$ using H_{st_2} and records hash value H_{pw_C} if the proof is valid.

⋮

n-1. For product P_n, distributor D generates a ownership proof $Proof_{P_n}$ using secret token st_n, then sends $Proof_{P_n}$ and hash value H_{pw_C} to the blockchain.

n-2. The blockchain verifies $Proof_{P_n}$ using H_{st_n} and records hash value H_{pw_C} if the proof is valid.

3. Distributor D sends pw_C to the blockchain.

4. The blockchain generates a hash value from pw_C, compares the hash value with H_{pw_C}

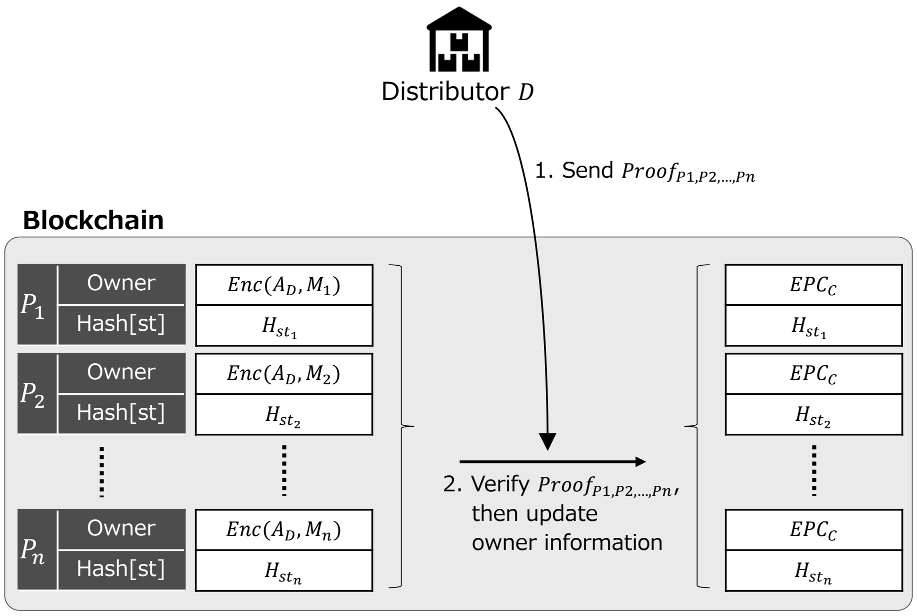


Figure 6: Packing process of Method B

in `Hash[pw]` of $P_1, P_2, \cdots, P_n$, and if they match, the blockchain updates `Owner` of products $P_1, P_2, \cdots, P_n$ with container C's EPC.

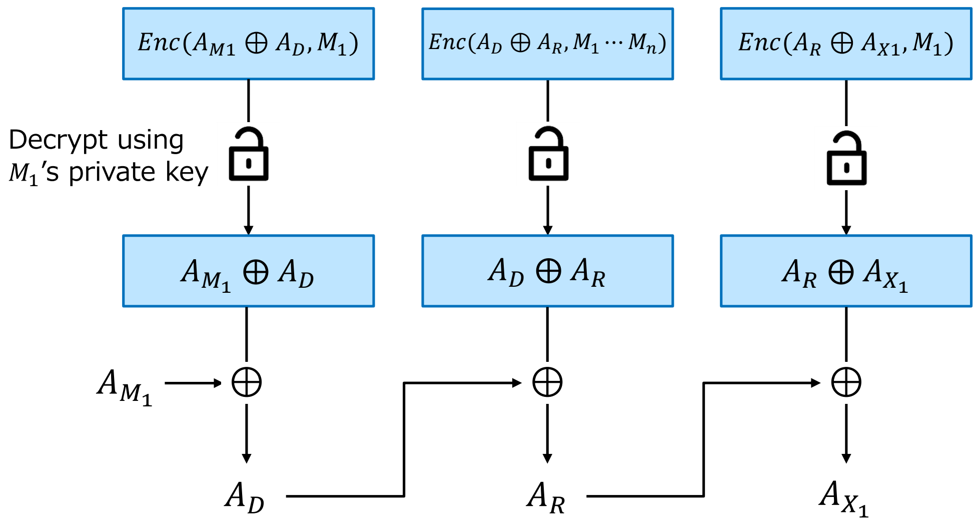
In Method A, when there are n products to be packed, it is necessary to perform the ownership proof n times. Each proof is a proof of knowledge of a single secret token. Therefore, as shown in Table 3, the proposed system requires n proofs, and the size of each proof is $O(1)$.

Method B

In Method B, we use a single aggregated proof, which aggregates the proofs of each secret token. If the system verifies this proof as valid, it updates the ownership information.

We use Figure 6 to illustrate the steps in detail. We also assume that distributor D in Figure 2 has received the products from manufacturers $M_1, M_2, \cdots, M_n$. Accordingly, distributor D possesses secret tokens $st_1, st_2, \cdots, st_n$, and the blockchain records its hash value $H_{st_1}, H_{st_2}, \cdots, H_{st_n}$.

1. Distributor D generates an aggregated proof $Proof_{P_1,P_2,\cdots,P_n}$ using secret tokens $st_1, st_2, \cdots, st_n$, then sends it to the blockchain.



$Enc(A_{M1} \oplus A_D, M_1)$
$Enc(A_D \oplus A_R, M_1 \cdots M_n)$
$Enc(A_R \oplus A_{X1}, M_1)$
Decrypt using M_1's private key
$A_{M_1} \oplus A_D$
$A_D \oplus A_R$
$A_R \oplus A_{X_1}$
A_{M_1}
A_D
A_R
A_{X_1}

Figure 7: Tracking product P_1 by manufacturer M_1.

2. The blockchain verifies $Proof_{P_1,P_2,\cdots,P_n}$ using $H_{st_1}, H_{st_2}, \cdots, H_{st_n}$, and if the proof is valid, the blockchain updates `Owner` of products $P_1, P_2, \cdots, P_n$ with container C's EPC.

Distributor D generates an aggregated proof taking all the secret tokens of n products as input. The aggregated proof shows that the hash values of the inputs match the hash values used to verify this proof. By using the hash values $H_{st_1}, H_{st_2}, \cdots, H_{st_n}$ for the verification of this proof, the system can verify ownership of all n products at once.

In Method B, when there are n products to be packed, a single proof shows that one knows n secret tokens. Therefore, as shown in Table 3, the proposed system only needs one proof, but the size of the proof is $O(n)$.

4.3 Product Tracking

We explain how manufacturers track their products in the distribution example in Figure 2.

Figure 7 shows a diagram of how manufacturer M_1 tracks product P_1. As described in Section 4.1, we use the exclusive-OR of the owner's address and the recipient's address as a message for encryption. Therefore, we can get the following ownership history of product P_1: A_{M_1}, $Enc(A_{M_1} \oplus A_D, M_1)$, EPC_C, $Enc(A_R \oplus A_{X_1}, M_1)$. Moreover, we can get the following ownership history of container C: $Enc(A_D \oplus A_R, M_1 \cdots M_n)$. All

Table 4: Variables for managing manufacturer information

Variable	Description
`companyPrefix`	Manufacturer's company prefix used in EPC
`companyName`	Manufacturer's name

of the ciphertexts in these histories have manufacturer M_1 specified as the entity who can decrypt. Therefore, manufacturer M_1 decrypts these ciphertexts and obtains the following values: $A_{M_1} \oplus A_D$, $A_D \oplus A_R, A_R \oplus A_{X_1}$. Manufacturer M_1 then calculates the exclusive-OR of A_{M_1} and $A_{M_1} \oplus A_D$ to obtain address A_D. Manufacturer M_1 further calculates the exclusive-OR of A_D and $A_D \oplus A_R$ to obtain address A_R. By repeating this, manufacturer M_1 can finally identify the following ownership history of product P_1: manufacturer M_1, distributor D, retailer R, consumer X_1.

4.4 Design Details

We explain the details of the processes in the proposed system. In the following, we explain the information to be recorded in the blockchain in the proposed system. We then describe the registration of the manufacturer, product manufacturing, product shipping, product receipt, product packing and product unpacking.

4.4.1 Information Recorded in Blockchain

The proposed system records manufacturer information, product information, and information necessary to verify ownership.

Information about manufacturers

In order to manage a supply chain, it is necessary to enroll products in the blockchain. The proposed system accepts product enrollment only when the manufacturing process is done by the registered manufacturer. We show the information to record the registered manufacturer in Table 4. This information is uniquely mapped to the manufacturer's blockchain address. A smart contract `ManufacturersManagerContract (MMC)` manages this information, and updates the information when the function provided by this smart

Table 5: Variables for managing product information

Variable	Description
`manufacturer`	Blockchain address of the product manufacturer
`owner`	Encrypted blockchain address of the product owner
`recipient`	Encrypted blockchain address of the product recipient
`status`	Product status
`number`	Block number when the product enrolled
`history`	History of the product owners

Table 6: Variables for managing product/container information

Variable	Description
`containedItems`	EPCs of the products contained in a product/container

contract is executed.

Information about products

Table 5 shows the product information that the proposed system manages. Since the proposed method uses EPC to manage product information, the information in Table 5 is uniquely mapped to EPC. A smart contract `ProductsManagerContract (PMC)` manages this information, and updates the information when the function provided by this smart contract is executed.

`PMC` also manages the information of products to realize the manufacturing process or containers to realize the packing process. In addition to the information in Table 5, the system manages the information in Table 6. For containers, `number` in Table 5 records the block number when the container is started to use. This allows us to get the past information if the container is reused.

Information about zero-knowledge proofs

Table 7 shows the information that the proposed system manages for ownership authentication. This information is uniquely mapped to the EPC. A smart contract `VerifierContract (VC)` manages this information, and updates the information when

Table 7: Variables for managing ownership authentication

Variable	Description
`verifyingKey`	Verification key of zk-SNARKs
`secretTokenHash`	Hash value of a secret token
`verifiedProof`	Verified proof to prevent diversion

the function provided by this smart contract is executed.

`VC` also provides a function `verifyTx()` to verify the proof of the argument. If the proof is verified as valid, `verifyTx()` returns $True$; if it fails, it returns $False$.

4.4.2 Manufacturer Registration

Algorithm 1 `enrollManufacturer()`: Enrolling process for a manufacturer's information

```
Input: The message sender's address ($A_{msg}$), Manufacturer M's address ($A_M$), company prefix
    ($companyPrefix$), company name ($companyName$)
1: if $A_{msg}$ is the address of an Admin then
2:     companyPrefix[$A_M$] $\leftarrow companyPrefix$
3:     companyName[$A_M$] $\leftarrow companyName$
4: end if
```

Algorithm 1 shows the process of registering a manufacturer's information. This information is required when executing the manufacturing process. This function is provided in `MMC`. We assume an administrator who manages manufacturers' information. We assume that the administrator is GS1, who manages company prefixes and EPCs. Only the administrator can execute this process. This function checks if the executor matches the administrator in step 1. If found to be $True$, `MMC` records `companyPrefix` and `companyName` as mapping to the manufacturer's address A_M.

4.4.3 Product Manufacturing

The following are the processes for enrolling a manufactured product. These are provided in `PMC`.

Algorithm 2 `enrollProduct()`: Enrolling process of a product

```
Input: $EPC$, The message sender's address ($A_{msg}$)
1: if the company prefix in $EPC$ is the same with $companyPrefix[A_{msg}]$ in MMC then
2:     manufacturer[$EPC$] $\leftarrow A_{msg}$
3:     owner[$EPC$] $\leftarrow A_{msg}$
4:     status[$EPC$] $\leftarrow Owned$
5:     number[$EPC$] $\leftarrow$ block.number
6:     history[$EPC$] $\leftarrow A_{msg}$
7: end if
```

Algorithm 2 shows the process of enrolling a new product simply. This function, in step 1, checks whether the company prefix in the EPC matches the company prefix of the executor. If found to be $True$, `PMC` records `manufacturer`, `owner`, `status`, `number` and `history` as mapping to the product's EPC. In step 5, the built-in variable `block.number` is used to record the block number when a new product is enrolled.

Algorithm 3 shows the process of registering a new product that consists of multiple products as parts. As in Algorithm 2, this function, in step 1, checks the company prefix in the EPC. In step 3, the function checks that `status` of each element in $EPCs$ is $Owned$ and that `owner` is A_msg. If these are found to be $True$, `PMC` updates `owner` of each element of $EPCs$ to EPC and `status` to $Transformed$, and records EPC, which is the owner of $EPCs$, to `history`. `PMC` records $EPCs$ in `containedItems`, which shows the parts of EPC's product. `PMC` also records `manufacturer`, `owner`, `status`, `number` and `history` as mapping to the new product's EPC. If the condition in step 3 is found to be $False$, `revert()` function is executed. The `revert()` is the built-in function in Ethereum. Once this function is executed, all data will return to the state before `transformProduct()` was executed.

Algorithm 4 shows the process of registering a new product that consists of multiple products as parts, similar to Algorithm 3. These processes differ in `status` specified in step 5. This difference in `status` indicates whether the products used as parts of the transformed/assembled product can be retrieved. The products enrolled with `transformProduct()` cannot be disassembled, while the products registered with `assembleProduct()` can be disassembled to retrieve its parts.

Algorithm 3 `transformProduct()`: Enrolling process of a product by transforming products

```
Input: Array of EPCs to be transformed ($EPCs$), New Product's EPC ($EPC$)
 1: if the company prefix in $EPC$ is the same with companyPrefix[$A_{msg}$] in MMC then
 2:     for $i = 1 \to n$ do
 3:         if status[$EPCs[i]$] = $Owned$ AND
            owner[$EPCs[i]$] = $A_{msg}$ then
 4:             owner[$EPCs[i]$] $\leftarrow EPC$
 5:             status[$EPCs[i]$] $\leftarrow Transformed$
 6:             history[$EPCs[i]$] $\leftarrow EPC$
 7:         else
 8:             revert()
 9:         end if
10:     end for
11:     containedItems[$EPC$] $\leftarrow EPCs$
12:     manufacturer[$EPC$] $\leftarrow A_{msg}$
13:     owner[$EPC$] $\leftarrow A_{msg}$
14:     status[$EPC$] $\leftarrow Owned$
15:     number[$EPC$] $\leftarrow$ block.number
16:     history[$EPC$] $\leftarrow A_{msg}$
17: end if
```

Algorithm 4 `assembleProduct()`: Enrolling process of a product by assembling products

Input: Array of EPCs to be assembled ($EPCs$), New Product's EPC (EPC)

```
 1: if the company prefix in $EPC$ is the same with companyPrefix[$A_{msg}$] in MMC then
 2:     for $i = 1 \rightarrow n$ do
 3:         if status[$EPCs[i]$] $=$ $Owned$ AND
            owner[$EPCs[i]$] $=$ $A_{msg}$ then
 4:             owner[$EPCs[i]$] $\leftarrow$ $EPC$
 5:             status[$EPCs[i]$] $\leftarrow$ $Assembled$
 6:             history[$EPCs[i]$] $\leftarrow$ $EPC$
 7:         else
 8:             revert()
 9:         end if
10:     end for
11:     containedItems[$EPC$] $\leftarrow$ $EPCs$
12:     manufacturer[$EPC$] $\leftarrow$ $A_{msg}$
13:     owner[$EPC$] $\leftarrow$ $A_{msg}$
14:     status[$EPC$] $\leftarrow$ $Owned$
15:     number[$EPC$] $\leftarrow$ block.number
16:     history[$EPC$] $\leftarrow$ $A_{msg}$
17: end if
```

4.4.4 Product Shipment

Algorithm 5 `shipProduct()`: Shipping process

```
Input: $EPC$, a proof of ownership ($Proof$), the encrypted recipient's address ($E_{rec}$), a hash value
    of the secret token ($H_{st}$), a verifying key ($vk$)
1: if status[$EPC$] $= Owned$ and
   $Proof$ is not recorded in VC.verifiedProof[$EPC$] and
   VC.verifyTx($Proof$) $= True$ then
2:   recipient[$EPC$] $\leftarrow E_{rec}$
3:   status[$EPC$] $\leftarrow Shipped$
4:   VC.verifyingKey[$EPC$] $\leftarrow vk$
5:   VC.secretTokenHash[$EPC$] $\leftarrow H_{st}$
6: end if
```

Algorithm 5 shows the process of shipping a product/container by the owner. This is provided in `PMC`. This function checks 3 conditions in step 1. First condition is to check that `status` of EPC is *Owned*. Second condition is to check that $Proof$ is not recorded in `VC`'s `verifiedProof[`EPC`]`, where the proofs verified before are recorded. Third condition is to check that the executor is the owner by executing `verifyTx()` of `VC` with $Proof$ as input. If these are found to be $True$, `PMC` updates `recipient` of EPC to E_{rec} and that `status` to *Shipped*. In addition, `VC` updates `verifyingKey` and `secretTokenHash`.

4.4.5 Product Receipt

Algorithm 6 `receiveProduct()`: Receiving process

```
Input: $EPC$, a proof of ownership ($Proof$)
1: if status[$EPC$] $= Shipped$ and
   VC.verifyTx($Proof$) $= True$ then
2:   owner[$EPC$] $\leftarrow$ recipient[$EPC$]
3:   status[$EPC$] $\leftarrow Owned$
4:   history[$EPC$] $\leftarrow$ owner[$EPC$]
5:   VC.verifiedProof[$EPC$] $\leftarrow Proof$
6: end if
```

Algorithm 6 shows the process of receiving the product/container shipped with `shipProduct()`. This is provided in `PMC`. This function checks 2 conditions in step 1. First condition is to check that `status` of EPC is *Shipped*. Second condition is to check that the executor is the recipient by executing `verifyTx()` of `VC` with $Proof$ as input. If these are found to be $True$, `PMC` updates `owner` of EPC with `recipient` and that `status` to *Owned*. `PMC` also adds a new `owner` to `history`. In addition, `VC` adds $Proof$ to `verifiedProof`.

4.4.6 Product Packing

We proposed two methods of the packing process, Method A and Method B. The following functions are provided in `PMC`. In the proposed system, we can not only pack products but also pack containers.

Method A

In Method A, the executor of the packing process first executes *preparePacking()*, which proves the ownership for each product/container to be packed. The executor then executes `packProductsMethodA()`, which is the packing process of Method A.

Algorithm 7 `preparePacking()`: Pre-process of method A's packing process

```
Input: $EPC$, a proof of ownership ($Proof$), the hash value of the password ($H_{pw}$)
1: if status[$EPC$] = $Owned$ and
   VC.verifyTx($Proof$) = $True$ then
2:   VC.passwordHash[$EPC$] ← $H_{pw}$
3: end if
```

Algorithm 7 shows a pre-process for the packing process of Method A. This function checks 2 conditions in step 1. First condition is to check that `status` of EPC is *Owned*. Second condition is to check that the executor is the recipient of the product by executing `verifyTx()` of `VC` with $Proof$ as input. If these are found to be $True$, `VC` records H_{pw}, which is the hash value of a password.

Algorithm 8 shows the packing process of Method A. In step 1, this function checks that `status` of EPC is $NotInUse$. If found to be $True$, in step 3, this function checks that `status` of each element in $EPCs$ is *Owned*. It also checks that `passwordHash` of `VC` matches the hash value obtained from pw. If these are found to be $True$, `PMC` updates

Algorithm 8 `packProductsMethodA()`: Packing process of method A

```
Input: Array of EPCs to be packed ($EPCs$), the password ($pw$), Container's EPC ($EPC$)
1: if status[$EPC$] = $NotInUse$ then
2:   for $i = 1 \rightarrow n$ do
3:     if status[$EPCs[i]$] = $Owned$ and
       Hash($pw$) = VC.passwordHash[$EPCs[i]$] then
4:       owner[$EPCs[i]$] ← $EPC$
5:       status[$EPCs[i]$] ← $Packed$
6:       history[$EPCs[i]$] ← $EPC$
7:     else
8:       revert()
9:     end if
10:  end for
11: end if
12: containedItems[$EPC$] ← $EPCs$
13: number[$EPC$] ← block.number
14: status[$EPC$] ← $Owned$
```

`owner` of each element in $EPCs$ to EPC and that `status` to *Packed*, and adds EPC to `history`. `PMC` records $EPCs$ in `containedItems`, which shows the products/containers packed in EPC's container. `PMC` also records the block number when the container starts to use. Finally, `PMC` records *Owned* in `status` with EPC.

Method B

In Method B, the executor of the packing process executes only *packProductsMethodB()*.

Algorithm 9 shows the packing process of Method B. This function checks 2 conditions in step 1. First condition is to check that `status` of EPC is $NotInUse$. Second condition is to check that the executor is the owner of all elements of $EPCs$ by executing `verifyTx()` of `VC` with $Proof_{agg}$ as input. If found to be $True$, this function checks `status` is *Owned* in step 3. If found to be $True$, `PMC` updates `owner` of each element in $EPCs$ to EPC and that `status` to *Packed*, and adds EPC to `history`. `PMC` records $EPCs$ in `containedItems`, which shows the products/containers packed in EPC's container. `PMC` also records *Owned* in `status` with EPC.

Algorithm 9 `packProductsMethodB()`: Packing process of method B

```
Input: Array of EPCs to be packed ($EPCs$), an aggregated proof of ownership ($Proof_{agg}$), Container's EPC ($EPC$)
 1: if status[$EPC$] $= NotInUse$ and
    VC.verifyTx($Proof_{agg}$) $= True$ then
 2:   for $i = 1 \to n$ do
 3:     if status[$EPCs[i]$] $= Owned$ then
 4:       owner[$EPCs[i]$] $\leftarrow EPC$
 5:       status[$EPCs[i]$] $\leftarrow Packed$
 6:       history[$EPCs[i]$] $\leftarrow EPC$
 7:     else
 8:       revert()
 9:     end if
10:   end for
11:   containedItems[$EPC$] $\leftarrow EPCs$
12:   status[$EPC$] $\leftarrow Owned$
13: end if
```

4.4.7 Product Disassembling/Unpacking

The following are the processes for dis-aggregation. These are provided in `PMC`.

Algorithm 10 shows the disassembling process for an assembled product enrolled with `assembleProduct()`. The disassembling process makes it possible to redistribute and reuse the parts. This function checks 3 conditions in step 1. First condition is to check that `status` of EPC is *Owned*. Second condition is to check that `status` of the first element of `containedItems` is *Assembled*. This condition makes it possible to check that EPC's product is enrolled with `assembleProduct()` and it consists of the parts. Third condition is to check that the executor is the owner by executing `verifyTx()` of `VC` with $Proof$ as input. If these are found to be $True$, `PMC` updates all the components assembled in EPC's product to *Owned*. It then deletes `containedItems` and sets EPC's `status` to *Destroyed*. Disassembled products cannot be reused. Therefore, `PMC` sets the status to indicate that the product can no longer be used. Finally, `PMC` adds *Disassembled*, which shows the product was disassembled, to `history`.

Algorithm 11 shows the unpacking process for a container. This function is similar

Algorithm 10 *disassembleProducts()*: Disassembling process

```
Input: a proof of ownership ($Proof$), Product's EPC ($EPC$)
1: if status[$EPC$] $= Owned$ and
   status[containedItems[$EPC$][0]] $= Assembled$ and
   VC.verifyTx($Proof$) $= True$ then
2:   for $i = 1 \to n$ do
3:     status[containedItems[$EPC$][$i$]] $\leftarrow Owned$
4:   end for
5:   delete containedItems[$EPC$]
6:   status[$EPC$] $\leftarrow Destroyed$
7:   history[$EPC$] $\leftarrow Disassembled$
8: end if
```

Algorithm 11 *unpackProducts()*: Unpacking process

```
Input: a proof of ownership ($Proof$), Container's EPC ($EPC$)
1: if status[$EPC$] $= Owned$ and
   status[containedItems[$EPC$][0]] $= Packed$ and
   VC.verifyTx($Proof$) $= True$ then
2:   for $i = 1 \to n$ do
3:     status[containedItems[$EPC$][$i$]] $\leftarrow Owned$
4:   end for
5:   delete containedItems[$EPC$]
6:   status[$EPC$] $\leftarrow NotInUse$
7:   history[$EPC$] $\leftarrow Unpacked$
8: end if
```

to Algorithm 10. These processes differ in `status` specified in step 1 and step 6. Second condition in step 1 is to check that `status` of the first element of `containedItems` is *Packed*. This condition makes it possible to check that EPC's container was processed with the packing process. In step 6, `PMC` sets EPC's `status` to $NotInUse$. Containers are not disposable and are often reused. Therefore, `PMC` sets the status to indicate that the container is not currently in use. Finally, `PMC` adds *Unpacked*, which shows the product was unpacked, to `history`.

Table 8: Product/Container and their EPCs in the scenarios

Product/Container	EPC
P_1	`1490897360744570370263043943 3`
P_2	`14909474007403800299327527434`
P_3	`14909974407361896896024615435`
P_4	`14910474807319993492721703436`
C_1	`00000000000000000000000000001`

Table 9: Entities and their addresses in the scenarios

Party	Address
Manufacturer M_1	`0xbCa677ff4eB2DfEC87a103392A356b65377Dc94A`
Manufacturer M_2	`0x48D439Ea70Eae2D06c2DC1F817C41FB18d11b6cD`
Manufacturer M_3	`0xd74ba48c88A0428D1632fFDcB26cF9795cE67e28`
Manufacturer M_4	`0xD7B4a8E0a1452e16aDc36D8fd809723522E229c8`
Party X_1	`0xd2b5Fb8288fCDbFFb0e7373b31dc60F2C60f89Fe`
Party X_2	`0x67e0Ff70C80afdcB568e57ac2bfB6e3b32C00a8b`
Party Y_1	`0x201d10515b77C66670f344c83Bd4aCc73Ac2c931`
Party Y_2	`0x4e1E52A4AB99695e928014cB3C374DF36d238d44`
Party Y_3	`0x0eEfa85FF18d3ab29DAe91294167c89cB0387d55`
Party Y_4	`0x62b03399CfD45e37eC095c5385ead3d4624eA9d0`

5 Evaluation

We implemented the proposed system and verified the operations of the proposed system using the scenarios. We also measured the transaction fees to discuss use cases of the proposed system.

5.1 Implementation

We implemented the proposed system using Ethereum, a public permissionless blockchain. We used version 0.8.4 of Solidity [17] to write the smart contracts of the proposed sys-

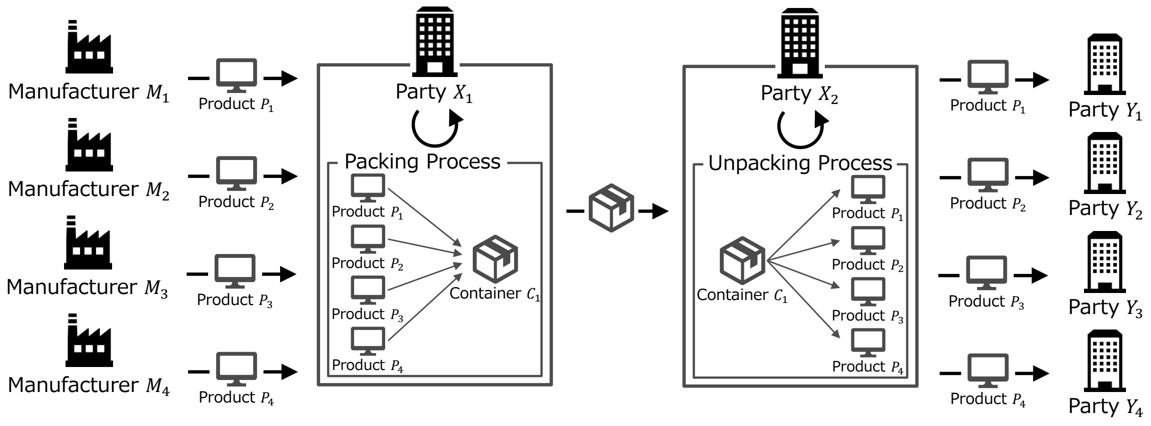


Figure 8: Product distribution in Scenario 1

tem. We used OpenABE [18] for the implementation of attribute-based encryption, and ZoKrates [19], a toolbox of zk-SNARK, for the implementation of zk-SNARKs. We assumed that the parties in the supply chain have secure communication channels.

Table 8 shows the information of products and a container used in this implementation, and Table 9 shows the information on manufacturers and parties.

5.2 Scenarios

We describe the scenarios used for the operation verification. We used the private network provided by Ganache [20].

Let us explain the state before the start of the scenarios. The blockchain address and company prefix of manufacturer $M_i (i \in\in \{1.4\})$ are registered in MMC. Manufacturer M_i holds the EPC of its product P_i. The company prefix of product P_i's EPC corresponds to manufacturer M_i's company prefix. In addition to the manufacturers and parties listed in Table 9, there are also manufacturers M' who attempt to distribute counterfeit products and party Z who are not involved in the supply chain.

5.2.1 Scenario 1: Normal product distribution

In this scenario, we verify that the product distribution can be carried out as designed in the proposed system. Products P_1, P_2, P_3 and P_4 are distributed from Manufacturers M_1, M_2, M_3 and M_4 to parties Y_1, Y_2, Y_3 and Y_4 as shown in Figure 8. The scenario is shown below.

1. Manufacturer $M_i (i \in \{1..4\})$ executes the enrolling process for product P_i.
2. Manufacturer M_i executes the shipping process of product P_i with specifying party X_1 as the recipient.
3. Party X_1 executes the receiving process of product P_i with a proof that he/she is the specified recipient.
4. Party X_1 executes the packing process of container C_1 for products P_1, P_2, P_3 and P_4.
5. Party X_1 executes the shipping process of container C_1 with specifying party X_2 as the recipient.
6. Party X_2 executes the receiving process of container C_1 with a proof that he/she is the specified recipient.
7. Party X_2 executes the unpacking process of container C_1 and retrieves products P_1, P_2, P_3 and P_4.
8. Party X_2 executes the shipping process of product P_1 with specifying party Y_1 as the recipient.
9. Party Y_1 executes the receiving process of product P_1 with a proof that he/she is the specified recipient.

5.2.2 Scenario 2: Confirming the manufacturer

In this scenario, we verify that any party can identify the manufacturer of the product.

After product distribution in Scenario 1, various parties attempt to obtain the manufacturer information of the product P_1. The scenario is shown below.

- Party X_1, which is involved in the supply chain, confirms that the manufacturer of product P_1 is M_1.
- Party Z, which is not involved in the supply chain, confirms that the manufacturer of product P_1 is M_1.

5.2.3 Scenario 3: Hiding the ownership information

In this scenario, we verify that the parties involved in the supply chain cannot know any more information other than the previous owner and the next owner. We also verify that no party, which is not involved in the supply chain, can browse any information of the product owner.

After product distribution in Scenario 1, various parties attempt to obtain the distribution information of product P_1. The scenario is shown below.

- Party X_1 knows the fact that it received product P_1 from manufacturer M_1 and shipped container C_1 to party X_2, but does not know any other distribution relationship.
- Party X_2 knows the fact that it received container C_1 from party X_1 and shipped it to party Y_1, but does not know any other distribution relationship.
- Party Y_1 knows the fact that it received product P_1 from party X_2, but does not know any other distribution relationship.
- Party Z, which is not involved in the supply chain, does not know the distribution relationship among manufacturer M_1, parties X_1, X_2 and Y_1.

5.2.4 Scenario 4: Distribution channel tracking by the manufacturer

In this scenario, we verify that the manufacturer can identify the distribution channel of its product.

After product distribution in Scenario 1, manufacturer M_1 attempts to identify the distribution channel of product P_1. The scenario is shown below.

- Manufacturer M_1 obtains the encrypted ownership history from the blockchain.
- Manufacturer M_1 decrypts the ownership history to obtain the distribution channel represented by the blockchain address.

Table 10: Data recorded in PMC at Step 1 of Scenario 1

Variable	Value
`manufacturer`	`0xbCa677ff4eB2DfEC87a103392A356b65377Dc94A`
`owner`	null
`recipient`	null
`status`	`Owned`

Table 11: Data recorded in PMC at Step 2 of Scenario 1

Variable	Value
`manufacturer`	`0xbCa677ff4eB2DfEC87a103392A356b65377Dc94A`
`owner`	null
`recipient`	`AAABLaETqm/JkYo90QKINYMF89jvNGt6CrIBFaEKQ194NjlmNmJjN6EksqEhAgp1`
	`+PzwFeToZHqIih2UR+3iZzNSFAAMYg9T6zMXkf1uoQZDcHJpbWWhJLKhIQIE0gZq`
	`OfaRyROLuuumZe1ifQPQfuRfuuHvvDe7TeP+7aEKRF94NjlmNmJjN6FEs6FBAhL2`
	`ce0+1QVnap2Je1QIRmQ381lHFM8xPlaGaZbRj9l2F9snkD5pTHsjrtVeuGKzfAKN`
	`vRrTZLrzlFxN43gifIShA19FRKFFHQAAAEBGFjWIR+kCqs+PXQW79D6lyDDUijD3`
	`1tIIes6b1b5Y6uvp0WIJVmX1OF7kMH5H82KhynDrL0BZbb1ip/iJC/IWoQZwb2xp`
	`Y3mhDR0AAAAIeDY5ZjZiYzcAAACDoROqAEaRij3RAog1gwXz2O80a3oKoWyhAkNU`
	`oS8dAAAAKjieiYmSfEdrt9zpAU9RBMzW4qdJ/dEkVPPf+eMxclaGFKFlFfJGLPFF`
	`IqECSVahFR0AAAAQKY0+/weK6AY2FbT/wBilu6EDVGFnoRUdAAAAEL2jL93ojN8d`
	`ixSDGLVEEQM=`
`status`	`Shipped`

5.2.5 Scenario 5: Preventing distribution of counterfeit goods

In this scenario, we verify that a party, which is not a manufacturer, cannot start a product distribution by pretending as if it is the manufacturer.

Manufacturer M' tries to distribute product P_1. The scenario is shown below.

- Manufacturer M' executes the enrolling process of product P_1, but the execution fails.

5.3 Results

We show the results of running the scenarios and the measured transaction fees.

Table 12: Data recorded in PMC at Step 3 of Scenario 1

Variable	Value
manufacturer	0xbCa677ff4eB2DfEC87a103392A356b65377Dc94A
owner	AAABLaETqm/JkYo90QKINYMF89jvNGt6CrIBFaEKQ194NjlmNmJjN6EksqEhAgp1 +PzwFeToZHqIih2UR+3iZzNSFAAMYg9T6zMXkf1uoQZDcHJpbWWhJLKhIQIE0gZq OfaRyROLuuumZe1ifQPQfuRfuuHvvDe7TeP+7aEKRF94NjlmNmJjN6FEs6FBAhL2 ce0+1QVnap2Je1QIRmQ381lHFM8xPlaGaZbRj9l2F9snkD5pTHsjrtVeuGKzfAKN vRrTZLrzlFxN43gifIShA19FRKFFHQAAAEBGFjWIR+kCqs+PXQW79D6lyDDUijD3 1tIIes6b1b5Y6uvp0WIJVmX1OF7kMH5H82KhynDrL0BZbb1ip/iJC/IWoQZwb2xp Y3mhDR0AAAAIeDY5ZjZiYzcAAACDoROqAEaRij3RAog1gwXz2O80a3oKoWyhAkNU oS8dAAAAKjieiYmSfEdrt9zpAU9RBMzW4qdJ/dEkVPPf+eMxclaGFKFlFfJGLPFF IqECSVahFR0AAAAQKY0+/weK6AY2FbT/wBilu6EDVGFnoRUdAAAAEL2jL93ojN8d ixSDGLVEEQM=
recipient	null
status	Owned

Table 13: Data recorded in PMC at the end of Scenario 1

Variable	Value
manufacturer	0xbCa677ff4eB2DfEC87a103392A356b65377Dc94A
owner	AAABLaETqm/Ju+nXa7MJtwGLqo8ioWVdBrIBFaEKQ194NjlmNmJjN6EksqEhAwg1 afC1LXBsqCjUJPNt22Q5vzh9+gpr32Zk5WG1Uzc4oQZDcHJpbWWhJLKhIQIjmKQY RtoH+9tA6YVqWQPhm0sqEereXjTvvx55SNcM8KEKRF94NjlmNmJjN6FEs6FBAwE/ 4F8m3SUZS7kTvhWmPxn6co/RDKOGuTCZjabV4JsrBrV/GxOZOWHmx9bCDJsUwx2y 3JIF7UaL3E7/NSrrTtWhA19FRKFFHQAAAEAJ+Rf3GA3uCLcv3DQhzo4EL6bKswPD BI5MnpX4yW7VD6Vsmj8CYlhUBImTgQKshvAIftlVD2dIq0trAXXzDcVRoQZwb2xp Y3mhDR0AAAAIeDY5ZjZiYzcAAACDoROqAEa76ddrswm3AYuqjyKhZV0GoWyhAkNU oS8dAAAAKkZJjx+pVzFE5NYBEDSzlRGVNJSybns31KxtdvV/TNnJQqJGz6GCd6IZ yqECSVahFR0AAAAQGJPuIp5xi2XO8oh4j7cN0aEDVGFnoRUdAAAAEGeLtG+xm4lq 3Gk3cQ2xv64=
recipient	null
status	Owned

Table 14: Manufacturer information of product P_1 obtained in Scenario 2

Information	Value
Manufacturer's address	`0xbCa677ff4eB2DfEC87a103392A356b65377Dc94A`

5.3.1 Scenario 1: Normal product distribution

As a result of scenario 1, the enrollment of product P_1 by manufacturer M_1 in step 1 was successful. Table 10 shows the information recorded in *PMC* at this time. Table 11 shows the information recorded in *PMC* when manufacturer M_1 executed the shipping process in step 2. Table 12 shows the information recorded in *PMC* when party X_1 executed the receiving process in step 3. Finally, product P_1 was distributed to party Y_1, and we found the information in Table 13 was recorded in *PMC*. Therefore, we verified that product P_1 was properly distributed starting from manufacturer M_1 to party Y_1.

5.3.2 Scenario 2: Confirming the manufacturer

As a result of scenario 2, we obtained the manufacturer's address shown in Table 14, regardless of the supply chain involvement. In this scenario, we used the EPC of product P_1 in Table 14 to obtain the manufacturer information. The obtained address matches manufacturerM_1's address. Therefore, we verified that the parties can identify the manufacturer of product P_1 is M_1.

5.3.3 Scenario 3: Hiding the ownership information

As a result of scenario 3, party X_1 could naturally identify the information of manufacturer M_1 and party X_2 because they are in the distribution relationships. However, since the other information stored in *PMC* was encrypted, party X_1 could not identify the information about distribution relationships, in which he/she was not involved. For example, the ownership information in Table 13 is encrypted, and since party X_1 doesn't have the decryption key, he/she cannot decrypt and identify the distribution relationship between X_2 and Y_1. Similarly, parties X_2 and Y_1 were able to identify information about the distribution relationships they were involved in, but not about the distribution they were not involved in. Since party Z could not decrypt the encrypted ownership information, he/she

Table 15: Ownership history of product P_1 obtained in Scenario 4

Index	Value
1	AAABLaETqm/JkYo90QKINYMF89jvNGt6CrIBFaEKQ194NjlmNmJjN6EksqEhAgp1+PzwFeToZHqIih2UR+3iZzNSFAAMYg9T6zMXkf1uoQZDcHJpbWWhJLKhIQIE0gZq0faRyR0LuuumZe1ifQPQfuRfuuHvvDe7TeP+7aEKRF94NjlmNmJjN6FEs6FBAhL2ce0+1QVnap2Je1QIRmQ381lHFM8xPlaGaZbRj9l2F9snkD5pTHsjrtVeuGKzfAKNvRrTZLrzlFxN43gifIShA19FRKFFHQAAAEBGFjWIR+kCqs+PXQW79D6lyDDUijD31tIIes6b1b5Y6uvp0WIJVmX1OF7kMH5H82KhynDrL0BZbb1ip/iJC/IWoQZwb2xpY3mhDR0AAAAIeDY5ZjZiYzcAAACDoROqAEaRij3RAog1gwXz2O80a3oKoWyhAkNUoS8dAAAAKjieiYmSfEdrt9zpAU9RBMzW4qdJ/dEkVPPf+eMxclaGFKFlFfJGLPFFIqECSVahFR0AAAAQKY0+/weK6AY2FbT/wBilu6EDVGFnoRUdAAAAEL2jL93ojN8dixSDGLVEEQM=
2	00000000000000000000000000001
3	AAABLaETqm/Ju+nXa7MJtwGLqo8ioWVdBrIBFaEKQ194NjlmNmJjN6EksqEhAwg1afC1LXBsqCjUJPNt22Q5vzh9+gpr32Zk5WG1Uzc4oQZDcHJpbWWhJLKhIQIjmKQYRtoH+9tA6YVqWQPhm0sqEereXjTvvx55SNcM8KEKRF94NjlmNmJjN6FEs6FBAwE/4F8m3SUZS7kTvhWmPxn6co/RDK0GuTCZjabV4JsrBrV/Gx0Z0WHmx9bCDJsUwx2y3JIF7UaL3E7/NSrrTtWhA19FRKFFHQAAAEAJ+Rf3GA3uCLcv3DQhzo4EL6bKswPDBI5MnpX4yW7VD6Vsmj8CYlhUBImTgQKshvAIftlVD2dIq0trAXXzDcVRoQZwb2xpY3mhDR0AAAAIeDY5ZjZiYzcAAACDoROqAEa76ddrswm3AYuqjyKhZV0GoWyhAkNUoS8dAAAAKkZJjx+pVzFE5NYBEDSzlRGVNJSybns31KxtdvV/TNnJQqJGz6GCd6IZyqECSVahFR0AAAAQGJPuIp5xi2X08oh4j7cN0aEDVGFnoRUdAAAAEGeLtG+xm4lq3Gk3cQ2xv64=

was unable to identify any distribution relationships.

Table 16: Ownership history of container C_1 obtained in Scenario 4

Index	Value
1	AAAC46ETqm/J5NPqzVjg5SXaMVplcxoAr7ICy6ELQ194MTNkZTQzNTWhJLKhIQMkdbEjGP66GM2Va/muHldThlVLJ2ZjxN2lnhyV9qu+CKELQ194MWE3ZGFmMWOhJLKhIQMS9e5UhBbJe2CqOOparv8mf0r9VrIzkrerYg9lzO2LuKEKQ194NjlmNmJjN6EksqEhAhClWYiCsyqkIijalXXC2+4TQNkiEB5lM0ad5zO+6vZOoQpDX3hkM2VkNzhloSSyoSEDDmyyIAM1kjTFCevW+isNRJGB925aDx66sRSJE5oMQ82hBkNwcmltZaEksqEhAwTI5ji0ZbxISALW47kpKOwjFX7QmEXJyhfkqGovhbU8oQtEX3gxM2RlNDM1NaFEs6FBAgRUN7BsRZdQlpYlogdAdz5xBBoJK/gQ3uG/fc2FW9v4F9G1yHvw9MC7JXr413Ru81AJ21zY9Q3Hha217gCDhJehC0RfeDFhN2RhZjFjoUSzoUEDDNMcNsJ6TAHwZzb73YVJCNA8kW59LaHy8T63CynNQf4HBdxvXbLXNnJf6+OdW4BTRbXfT2P44QKZFv8FOOc3tKEKRF94NjlmNmJjN6FEs6FBAw4/CNHElrbcutouU4QW4VwqJ8/qRFXYhnz4FcpW9UOcEkgxmEjKCBwf++DVsfsXuXoFud3vjaAaOvVWJBl5rD2hCkRfeGQzZWQ3OGWhRLOhQQIJ0Kkv1oH9ZStjSgefJPZBmUi1/NkejDsc3wVdIdp5Ixae1JdJC4961XEj6jElutwCCXFaWg6e1eetDJuX5SjGoQNfRUShRROAAABAWLrFl6Wz8PcdZYi7cZFMDzwst9SWKjV017EpknYMPpBIYRRG3DOFCvTOFI7EtvfTnBn89oqloKZiT5M4krATVqEGcG9saWN5oTMdAAAALng2OWY2YmM3IG9yIHhkM2VkNzhlIG9yIHgxM2RlNDM1NSBvciB4MWE3ZGFmMWMAAACDoROqAEbkO+rNWODlJdoxWmVzGgCvoWyhAkNUoS8dAAAAKovLnxW1rMfXExwx+DjhTOp9l75fuvD+TIFIttg3qrQYsqE8yfekWN8ayaECSVahFROAAAAQb8/qPpPahbq+o+zsEiBmiqEDVGFnoRUdAAAAEDoJLeUmpMU/vv8GLKpdqSA=
2	Unpacked

Table 17: Ownership history after decryption in Scenario 4

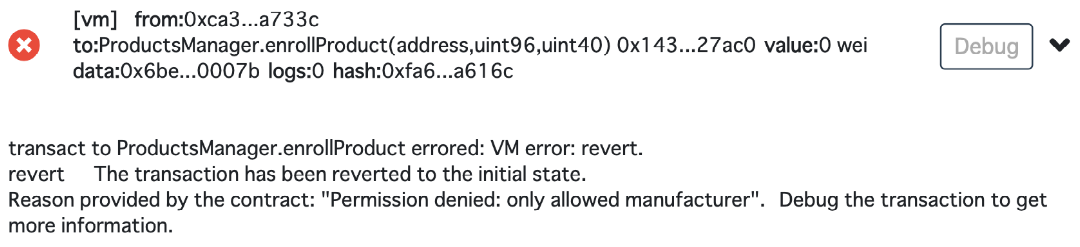
Index	Value
1	`0x6e138c7dc64e0413374634021be90b97f17240b4`
2	`0xb55504f240f62634e66960971a270ec9f4cf8375`
3	`0x47fdef21937d3bad267d1364102fc2fc0802c3ba`

Table 18: Plain text of ownership history in Scenario 4

Index	Value
1	`0xd2b5fb8288fcdbffb0e7373b31dc60f2c60f89fe`
2	`0x67e0ff70C80afdcb568e57ac2bfb6e3b32C00a8b`
3	`0x201d10515b77C66670f344c83bd4acc73ac2c931`

5.3.4 Scenario 4: Distribution channel tracking by the manufacturer

As a result of scenario 4, manufacturer M_1 obtained the ownership history of product P_1 shown in Table 15. Manufacturer M_1 also obtained the ownership history of container C_1 shown in Table 16, since the ownership history of P_1 records that P_1 was packed in container C_1. By using the decryption key held by manufacturer M_1, M_1 obtained the values shown in Table 17. Manufacturer M_1 computed the exclusive-OR as described in Section 4.3, and obtained the values shown in Table 18. These values are the plaintext of product P_1's distribution channel. Therefore, manufacturer M_1 could track the product distribution of product P_1.



```
[vm]  from:0xca3...a733c
to:ProductsManager.enrollProduct(address,uint96,uint40) 0x143...27ac0 value:0 wei
data:0x6be...0007b logs:0 hash:0xfa6...a616c

transact to ProductsManager.enrollProduct errored: VM error: revert.
revert     The transaction has been reverted to the initial state.
Reason provided by the contract: "Permission denied: only allowed manufacturer".  Debug the transaction to get
more information.
```

Figure 9: Result screen of Scenario 5

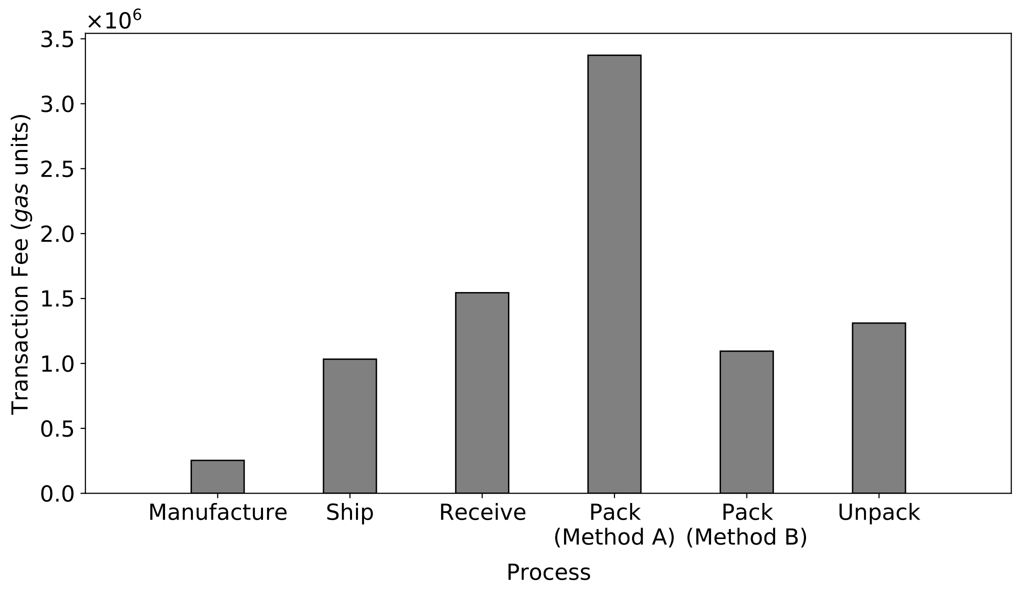


Figure 10: Transaction fees for each process executed in Scenario 1

5.3.5 Scenario 5: Preventing distribution of counterfeit goods

As a result of scenario 5, manufacturer M' failed the enrollment of product P_1 due to the error shown in Figure 9. From the above results, we verified that the proposed system can protect the distribution information, prevent unauthorized product distribution, and enable manufacturers to track the distribution channel.

5.3.6 Transaction fees

We measured the transaction fees for each process that is executed in scenario 1. Figure 10 shows the result for transaction fees. The manufacturing process is the process executed by the manufacturer. In the proposed system, the manufacturer's blockchain address is not

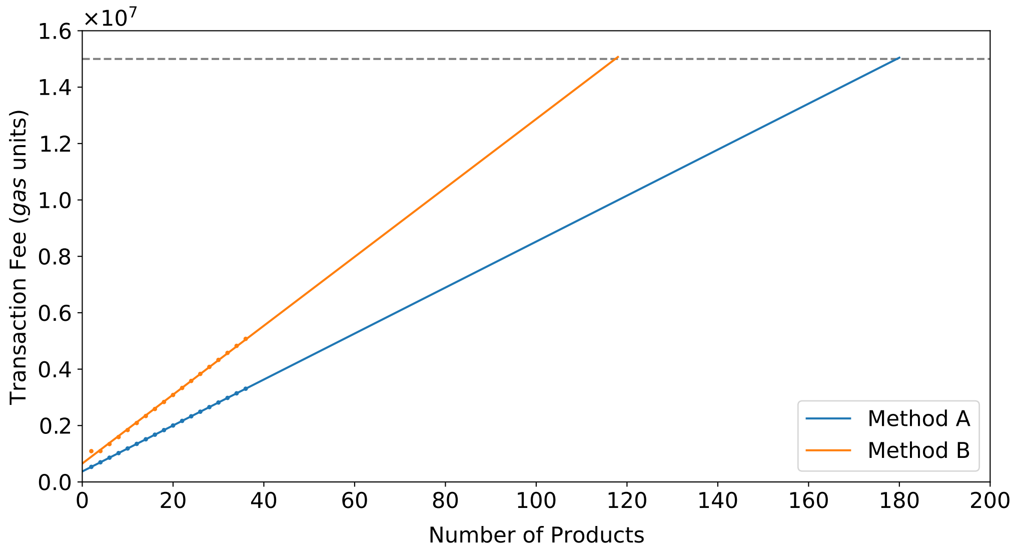


Figure 11: Relationship between transaction fees and the number of products to be packed

encrypted because the distribution information involving the manufacturer is not subject to protection. In other words, its blockchain address is recorded in the manufacturing process. In contrast, the ownership information is recorded as ciphertext in all other processes. The ciphertext is longer than the blockchain address and needs to be recorded as a string. The method of ownership authentication, which is a condition of execution, is also different. The manufacturer's execution can simply use its blockchain address, while other parties need to authenticate their ownership based on zero-knowledge proofs. The proof of zero-knowledge proofs has a larger data size than the blockchain address. Thus, the transaction fee for the manufacturing process is lower than for the other processes. In addition, we have implemented two packing processes, Method A and B. We can find that the packing process of Method A has a higher transaction fee than the packing process of Method B. This is due to the number of executions. In Method A, the number of executions is equal to the number of products to be packed in order to complete the packing process. Method B, on the other hand, requires only one execution by aggregating proofs. As a result, the total transaction fee required for the packing process of Method B is smaller than that of Method A.

We also calculated the limit of the number of products that can be packed in the packing processes. Figure 11 shows the relationship between the number of products and the transaction fee. We measured the transaction fee up to 38 products. We calculated the transaction fees after that are the values predicted by the least squares method from the transaction fees up to that point. For Method A, the upper limit of the number of products to be packed is calculated to be 179. This is a limitation due to Ethereum. Ethereum has a block gas limit, which is the maximum transaction fee for each block. The higher the amount of data and computation required for execution, the higher the transaction fee. Therefore, as the number of products to be packed increases, the required transaction fees also increase. In the current release of Ethereum, the block gas limit is set to be $1.5 \times 10^7 gas$ units on average, so we calculated the upper limit by assuming the block gas limit as $1.5 \times 10^7 gas$ units. For Method B, we found that the upper limit of the number of products to be packed is 38. This is a constraint by ZoKrates that is used for proof generation in zero-knowledge proofs. Method B requires a single proof that aggregates ownership of all products to be packed. The larger the number of products to be packed, the larger the proof size and the larger the computational complexity to generate the proof. We found that if the number of products to be packed is more than 39, the computational limit is reached and the generation of the proof fails. In addition, we calculated the upper limit for the case where the generation of proofs does not fail. We calculated the same method as in Method A. As a result, we found the upper limit of the number of products to be packed is 117, as shown in Figure 11. Method A reduces the transaction fee required for each execution by splitting the execution into several parts. As a result, the upper limit of products to be packed in Method A is larger than that of Method B.

5.4 Discussion

In one distribution (shipping and receiving), the total transaction fee required for one party is up to $2.6 \times 10^6 gas$ units. Converted to legal tender using the gas fee as of January 25, 2022, the date of this evaluation, it is 765.8 USD, which is not cheap. We then implemented the proposed system to Binance Smart Chain (BSC) [21], a public permissionless blockchain that can use the same codes as Ethereum. As a result, the transaction fee required for one party is at most 4.5 USD. Comparing Ethereum and BSC, Ethereum is more decentralized. The consensus algorithm used by Ethereum allows anyone to become a validator by staking a certain amount of ETH. Validators can hold the right to propose blocks or to vote on the validity of another node's block. In contrast, there are only 21 validators in BSC. This means that Ethereum is more decentralized and more secure than BSC. The products where counterfeiting is a problem are comparatively expensive products. For such products, it is likely to use Ethereum, which is more secure, even with higher transaction fees. With the low transaction fees in BSC, the proposed system can be applied to various products. If the distribution is based on the proposed system, only the manufacturer can identify the product owner. This allows the manufacturer to immediately recall products if they have any problems and defects. The transaction fee can be regarded as a fee for receiving warranty in this case. Thus, by selecting a blockchain depending on the target product, the proposed system can be applied to products of various prices.

6 Conclusion

Supply chains are growing significantly, and this has caused problems with traceability. In order to improve supply chain traceability, blockchain-based systems for managing distribution information have been proposed. It is desirable to use a public permissionless blockchain for a supply chain system that is available to new businesses and general consumers. However, anyone can browse the information recorded in a public permissionless blockchain, causing the leakage of confidential information such as transaction relationships between businesses. In this thesis, we proposed a supply chain system to protect the confidential information of products and track product distribution, using a public permissionless blockchain. We realized protection of confidential information by encrypting the blockchain address that represents the ownership information. We used attribute-based encryption and assigned unique attributes to manufacturers. In the distribution of a single product, the single attribute of the manufacturer is used to encrypt the ownership information. In the distribution of multiple products, the union of the manufacturers' attributes is used to encrypt the ownership information. These allow the manufacturers to track the product distribution in various units. We used zero-knowledge proof to hide the ownership information and ensure distribution between legitimate owners and recipients, thereby preventing unauthorized distribution. We implemented the proposed system and found that the transaction fee per distribution is $2.6 \times 10^6 gas$ units. This was equivalent to 765.8 USD for Ethereum and 4.5 USD for Binance Smart Chain (BSC) at the time of writing (January 25, 2022). Ethereum is more decentralized than BSC, which means Ethereum is a more secure blockchain. We discussed that the proposed system can be applied to various products by taking a strategy such as using Ethereum for high-priced products to prevent counterfeits and using BSC for low-priced products to provide quick response when problems occur.

As a future work, we need to protect confidential information at the protocol level. In the proposed system, the blockchain addresses stored in the smart contracts are protected with encryption at the application level. On the other hand, the blockchain protocol records the blockchain address of the smart contract executor. In this thesis, we assume that the product owner and recipient execute the smart contracts, so anyone can identify them from the execution history of the smart contracts. One of the possible solutions is to introduce an intermediate server that executes smart contracts on behalf of the owner and recipient, thereby we can avoid the direct execution by them.

Acknowledgments

This thesis could not be accomplished without a lot of great support from many people. First, I would like to express my sincere gratitude to Professor Masayuki Murata of Osaka University, for his valuable comments, insights and continuous encouragement. Furthermore, I would show my greatest appreciation to Mr. Yoshinobu Shijo. He gave me appropriate guidance and a lot of helpful advices. Without his support, I would not have accomplished this thesis. Moreover, I would like to sincerely appreciate Associate Professor Shin'ichi Arakawa, Associate Professor Yuichi Ohsita, Associate Professor Naoto Yanai, Assistant Professor Daichi Kominami and Assistant Professor Tatsuya Otoshi of Osaka University, giving me continuous and appropriate advices. Finally, I would like to thank all the members of Murata Laboratory for their support.

References

[1] OECD and European Union Intellectual Property Office, *Global Trade in Fakes*, Jun. 2021.

[2] Centers for Disease Control and Prevention, "Multistate outbreaks of Shiga toxin-producing Escherichia coli O26 infections linked to Chipotle Mexican Grill Restaurants (final update)," https://www.cdc.gov/ecoli/2015/o26-11-15/index.html, Feb. 2016, [Online; accessed 3-February-2022].

[3] K. Toyoda, P. Takis 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. 17 465–17 477, Jun. 2017.

[4] H. M. Kim and M. Laskowski, "Toward an ontology-driven blockchain design for supply-chain provenance," *Intelligent Systems in Accounting, Finance and Management*, vol. 25, no. 1, pp. 18–27, Mar. 2018.

[5] H. Huang, X. Zhou, and J. Liu, "Food supply chain traceability scheme based on blockchain and EPC technology," in *Proceedings of Smart Blockchain*, Nov. 2019, pp. 32–42.

[6] R. B. dos Santos, N. M. Torrisi, and R. P. Pantoni, "Third party certification of agri-food supply chain using smart contracts and blockchain tokens," *Sensors*, vol. 21, no. 16, 2021.

[7] M. Westerkamp, F. Victor, and A. Küpper, "Tracing manufacturing processes using blockchain-based token compositions," *Digital Communications and Networks*, vol. 6, no. 2, pp. 167–176, 2020.

[8] V. Acharya, A. E. Yerrapati, and N. Prakash, *Oracle Blockchain Quick Start Guide: A practical approach to implementing blockchain in your enterprise.* Packt Publishing Ltd, Sep. 2019.

[9] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. challenges and opportunities," *Future Generation Computer Systems*, vol. 88, pp. 173–190, Nov. 2018.

[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf, 2008, [Online; accessed 3-February-2022].

[11] V. Buterin, "Ethereum whitepaper," https://ethereum.org/en/whitepaper/, 2013, [Online; accessed 3-February-2022].

[12] "IBM food trust," https://www.ibm.com/blockchain/solutions/food-trust, [Online; accessed 3-February-2022].

[13] S. R. Bryatov and A. A. Borodinov, "Blockchain technology in the pharmaceutical supply chain: Researching a business model based on hyperledger fabric," in *Proceedings of the International Conference on Information Technology and Nanotechnology*, 2019, pp. 21–24.

[14] D. Agrawal, S. Minocha, S. Namasudra, and A. H. Gandomi, "A robust drug recall supply chain management system using hyperledger blockchain ecosystem," *Computers in Biology and Medicine*, vol. 140, p. 105100, Jan. 2022.

[15] M. el Maouchi, O. Ersoy, and Z. Erkin, "Decouples: A decentralized, unlinkable and privacy-preserving traceability system for the supply chain," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, Apr. 2019, pp. 364–373.

[16] S. Rass and D. Slamanig, *Cryptography for Security and Privacy in Cloud Computing*. Artech House, 2013.

[17] "Solidity programming language," https://soliditylang.org/, [Online; accessed 3-February-2022].

[18] "The openabe library - open source cryptographic library with attribute-based encryption implementations in c/c++," https://github.com/zeutro/openabe, [Online; accessed 3-February-2022].

[19] "Github - zokrates/zokrates: A toolbox for zksnarks on ethereum," https://github.com/Zokrates/ZoKrates, [Online; accessed 3-February-2022].

[20] "Github - trufflesuite/ganache-ui: Personal blockchain for ethereum development," https://github.com/trufflesuite/ganache-ui, [Online; accessed 3-February-2022].

[21] Binance.org, "Binance smart chain: A parallel binance chain to enable smart contracts," https://www.binance.org/en/smartChain, 2020, [Online; accessed 3-February-2022].