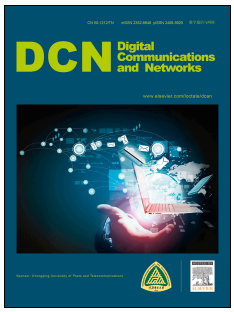
Accepted Manuscript



Tracing manufacturing processes using blockchain-based token compositions

Martin Westerkamp, Friedhelm Victor, Axel Kupper

PII: S2352-8648(18)30244-X

DOI: https://doi.org/10.1016/j.dcan.2019.01.007

Reference: DCAN 156

To appear in: *Digital Communications and Networks*

Received Date: 1 October 2018

Revised Date: 17 December 2018

Accepted Date: 27 January 2019

Please cite this article as: M. Westerkamp, F. Victor, A. Kupper, Tracing manufacturing processes using blockchain-based token compositions, *Digital Communications and Networks* (2019), doi: https://doi.org/10.1016/j.dcan.2019.01.007.

Digital Communications and Networks(DCN) ACCEPTED MANUSCRIPT

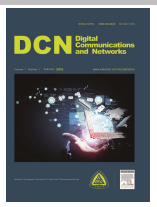
HOSTED BY

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

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



Tracing Manufacturing Processes using Blockchain-based Token Compositions

Author 1*, Author 2, Author 3

Institute, Department, University,
City, Country

Abstract

Supply Chain Traceability is one of the most promising use cases to benefit from blockchain characteristics such as decentralization, immutability and transparency, without requiring to build prior trust relationships among entities. A plethora of blockchain-based supply chain traceability solutions has been proposed recently. However, current systems are limited to tracing simple goods that have not been part of a manufacturing process. We propose a system that allows for the traceability of manufactured goods, including their components. Products are represented using digital non-fungible tokens that are created on a blockchain for each batch of products that has been manufactured. To create a link between a product and the components that are needed to manufacture it, we propose "token recipes" that define the amount of tokenized goods that are required for minting a new token. As input tokens are automatically and transparently consumed when creating a product token, the physical production process of composing a new good out of existing components is projected onto the ledger. This ultimately leads to the complete traceability of goods, including the origin of inputs. Evaluating the performance of the system, we show that a prototypical implementation for the Ethereum Virtual Machine (EVM) scales linearly with the number of inputs and goods tracked.

KEYWORDS:
Supply Chain, Traceability, Blockchain, Distributed Ledger Technology, Smart Contract

1. Introduction

Providing traceability of goods from resource to retailer has become increasingly important in the past decade. Consumers have a larger interest in consuming goods that comply with certain ecological and ethical standards [1]. Global supply chains have become complex to a greater extent, hampering quality management in manufacturers' procurement [2]. Furthermore, regulations, international standardizations and increased consumer awareness imply novel requirements towards supply chain management systems. For instance, the European Parliament postulates the traceability of food, requiring food suppliers and market actors to provide information about provenance of goods [3]. In addition, the ISO 9001:2015 standard instructs organizations to monitor identifiability and traceability of products and services.

To cope with these requirements, supply chain management systems should ideally be operated by multiple business partners to trace a product's origin and its transformation process. However, traditional supply chain management systems are operated isolated from other participants and are not capable of providing comprehensible provenance information [4]. Resulting shortcomings include insufficient trust between parties, isolated data storage and unsatisfactory standardization in communication and data formats [5, 6].

Recently, blockchain technology has been proposed for providing enhanced traceability in supply chains [6, 7, 8, 9, 10]. Major drivers originate from typical blockchain characteristics such as decentral-

*Corresponding author.
E-mail address: author@mail.com (Author1)

ization, verifiability and immutability that could tackle the observed shortcomings.

Current blockchain-based solutions for supply chain traceability promote tracking goods over multiple tiers by utilizing markers such as RFID and QR codes [5]. This linkage mechanism enables proving provenance for anti-counterfeit with regard to high value goods such as diamonds[1], medicine [11] or generally in the post-retail supply chain [10]. But these approaches are limited to non-modifiable goods and do not consider the production processes. Thus, it is neither possible to track a product after it has been processed, nor to trace an end-product's inputs towards its primary resources.

In contrast to existing solutions, we foster a representation mechanism for the convertibility of products. Instead of only projecting physical goods onto the blockchain in the form of tokens, our target is to document their transformation in the production process on the ledger. We therefore propose a set of smart contracts that handles modifiable goods by capturing their creation, transformation and exchange on a distributed ledger. As a result, not only the good's origin is traceable but also its inputs. Hereby, we tackle two requirements for supply chain management that conventional systems cannot deliver comprehensively [12]. Firstly, customers are empowered to review a product's and its ingredients' quality in various dimensions such as environmental and labour standards. Secondly, in the context of quality management, the proposed system enables manufacturers to monitor the supply chain for multiple tiers rather than relying on information provided by suppliers. Our key contributions in this paper are as follows:

1. We design a supply chain traceability system that models manufacturing processes as token recipes.
2. We present a prototypical implementation for the Ethereum Virtual Machine using smart contracts.
3. We evaluate our implementation with respect to smart contract execution costs and scalability with increasing product complexity in terms of inputs.

2. Background and related work

2.1. Conventional Supply Chain Provenance Systems

The state of the art for tracing and tracking products throughout a supply chain is storing records of suppliers and customers in a centralized manner within each participant's supply chain management system [4]. This information is isolated or shared among suppliers and customers to achieve comprehensive insights on supply chain provenance [12].

[1] https://www.everledger.io/

In the context of supply chain management, traceability promotes following a good's path upwards to its origin, while tracking refers to the downward operation, from raw materials to end products [1]. For storing, sharing and managing relevant information, a plethora of information systems is used in practice. Warehouse and transport management systems focus on internal warehouse operations, while enterprise resource planning (ERP) systems include supplier management, reordering and billing [13]. Dedicated supply chain management software applications target forecasting future demands and fulfilling these from available suppliers [14]. As no global view on the supply chain is given, provenance information is retrieved from the next tier, requiring trust in the supplier or a third party. However, enabling multi-tier traceability is essential for decreasing risks caused by quality fluctuations in source products [15].

To ensure the coupling between physical goods and digital representations, identification mechanisms are fundamental. The ISO 28219:2017 standard defines guidelines for creating globally valid identifiers that are enforced by utilizing bar codes or two-dimensional symbols such as QR codes or alternative RFID tags for projecting physical goods onto digital systems [16]. Identifiers either refer to single goods or product batches. Varying the batch size permits influencing the granularity to which products are traceable. To ensure a certain quality degree, Bechini et al. propose defining batches of goods that are linked to a multitude of quality features [12].

While traditional supply chain information systems are capable of uniquely identifying products, traceability is limited. This is mainly due to isolated data that reflects organizations' sourcing and sales. Tracing ingredients over multiple tiers would require shared data that is tamper proof while maintaining high accessibility.

2.2. Smart Contracts and the Ethereum Virtual Machine

Blockchains provide a distributed, shared state all participants agree on using a consensus algorithm. Their tamper proof characteristics facilitate the opportunity of introducing a global view on multi-tier supply chains [17]. To apply business logic in blockchains, smart contracts have been introduced. First proposed by Nick Szabo in 1994 [18], smart contracts are computer programs that enforce rules without requiring a third party. In the Bitcoin blockchain, a basic version of smart contracts is implemented through the means of a scripting system that facilitates use cases like multi-user accounts (multi-signature wallets) and escrow services.

The main technology advancement of the Ethereum blockchain [19] is the introduction of a general purpose and turing complete smart contract system

which is manifested in the Ethereum Virtual Machine (EVM).

In the EVM, program code is executed by miners and other network participants who verify state changes. A smart contract in Ethereum is typically written in a high level programming language like Solidity or Viper and compiled to bytecode that is then deployed on the blockchain. Execution requires sending a transaction to the contract's address, while specifying which function is to be called given a set of parameters. These functions in turn can call other smart contracts if they have been programmed to do so. Computationally intensive routines, however, are not suitable, as the execution has a cost attached to it. For each operation supported by the EVM, a gas cost is defined in the Ethereum yellow paper [20], where the main cost drivers are operations that store or change values on the blockchain. A transaction therefore needs to contain a sufficient amount of gas in order to guarantee successful execution. The actual costs of a transaction depend on how much gas is needed, and a gas price a user is willing to spend for each unit of gas. The transaction costs spent by a user are awarded to the miner that includes the transaction in a new block, as s/he has to verify and execute the transaction.

While the EVM was orginally designed for the Ehereum blockchain, several projects aim to port it to other ledgers. For instance, Counterparty[2] adds a secondary computation layer to run EVM code on top of the bitcoin network. Qtum[3] supplies a bitcoin fork implementing the EVM with the goal of abstracting from Ethereum's account based model in order to support bitcoin-like light clients. Additionally, smart contracts for the EVM can also be run on permissioned blockchains like Quorum[4] or Hyperledger Burrow[5], removing the need to operate in a public environment. To provide a common form of tokens in Ethereum, a standard interface referred to as *Ethereum Request for Comments (ERC) 20* was introduced[6]. Popular Ethereum wallets like Mist[7] and Parity[8] support tokens that implement this interface out of the box. All standard operations such as receiving balances and transferring tokens to other addresses are available in the wallet's GUI.

In contrast to ERC20 tokens which cannot be distinguished, a standard for non-fungible tokens, ERC721, is proposed for handling deeds[9]. The target is to create a standardized interface for creating and trading discriminable tokens reflecting digital or physical goods. Consuming tokens or defining conditions for their generation is not included in the proposal, resulting in the requirement for the implementation of non-standard functionality.

2.3. External data storage

While a blockchain can be used as a data store, it is impractical for large files due to the attached transaction costs that prevent rapid ledger growth [19, 21]. As sharing additional product-specific information along the supply-chain is a requirement in many use cases [22], external data should be accessible in a secure and distributed manner while guaranteeing high availability.

In the context of blockchain technologies, peer to peer storage networks such as IPFS [23] or Swarm [24] have been proposed that permit storing data in a peer-to-peer fashion, but outside of a blockchain. Stored data can be held redundantly by multiple participants and can be accessed through a unique hash of the data. As data is addressed by its hash, the underlying information is tamper-proof and therefore well suited for linking to external data from, for example, smart contracts. However, the data can only be accessed as long as network participants make it available. Mechanisms to incentivize users to keep data available are currently in development in Swarm and Filecoin [25], a project that builds upon IPFS.

2.4. Blockchain-based Supply Chain Traceability

Several blockchain-based solutions have been proposed in literature to overcome current obstacles in guaranteeing a certain product quality [26], compliance with legal obligations [12] or to counter fraud [27].

While there are various approaches tackling supply chain traceability, they are mostly concerned with tracing single, non-modifiable goods[10] [28, 9]. In fact, they target proving a good's authenticity and ownership via multiple hops. For instance, Kim and Laskowski present an ontology that promotes provenance in supply chains using the Ethereum blockchain [9]. The proposed contract is implemented using Solidity and supports several typical supply chain operations such as *produce* and *consume*. Nevertheless, the definition of novel functions and properties in the contract is limited to a rigid type system and the production of new goods out of existing resources is not possible.

A common issue in supply chain traceability is the projection of physical goods onto a digital representation. Tian suggests a token representation that implies benefits such as proving authenticity by attaching RFID chips or QR codes to link physical products with their digital counterparts on the blockchain [28].

[2] https://counterparty.io/
[3] https://qtum.org/
[4] https://github.com/jpmorganchase/quorum
[5] https://www.hyperledger.org/projects/hyperledger-burrow
[6] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
[7] https://github.com/ethereum/mist
[8] https://wiki.parity.io/Parity-Wallet
[9] https://github.com/ethereum/EIPs/pull/841
[10] https://www.ascribe.io/

It inherently allows for proving ownership and permits transferring it to another party. Hereby, a good can be tracked from creation to retail. Toyoda et al. propose a blockchain-based post-retail supply chain system for anti counterfeits in which they utilize this linkage to enable tracing products after being sold in retail [10].

Reviewing the here presented approaches for providing traceability in the supply chain unveiled a major shortcoming for more complex use cases. While they permit creating digital representations of physical goods, which facilitate tracking across multiple entities, this connection is lost in case the product is processed.

3. Concept

We propose a blockchain-based, decentralized supply chain management system based on smart contracts. In order to provide comprehensive provenance information to consumers and producers, we maintain the relationship between resources and products in manufacturing processes. The concept is based on two key ideas: representing physical goods in the form of digital tokens, and recipes that enable their transformation. Additional functionality like certifying goods, transferring, splitting and combining tokens facilitates cross-business traceability. The following subsections describe each of these aspects in detail and introduce the participants' roles based on a specific example visualized in Figure 1.

3.1. Tokenization of goods

For each type of good managed in the supply chain, a smart contract is set up. Within such a smart contract, tokens that represent physical goods can be created. One token corresponds to one batch of goods that could be measured in items, weight, volume or size. Product information can be stored via IPFS or Swarm, by adding the corresponding hash to the token. This data could include information such as product images, expiration dates, disposal instructions, manuals, documentation of defects and many others. General product type information could also be referenced in the token contract itself.

The batch size is flexible, so that large quantities of goods, but also single entities are manageable. The tokens are non-fungible, meaning that each token is unique. This allows distinguishing between batches of the same type of good. To apply this concept to the physical world, the contract owner would create digital tokens that correspond to manufactured products.

3.2. Recipes for good transformation

In order to digitally represent a manufacturing process, several tokens can be transformed into a new token. When creating a new smart contract, the product composition is defined. Comparable to a recipe, the creator specifies a number of input goods and corresponding amounts that are required for the creation of a new product. Following the recipe, when a batch of goods is to be created, the owner of the contract needs to possess the required input goods in sufficient amounts. In fact, the specific batches of input tokens need to be specified, so that they can be consumed by the smart contract automatically. If a batch of units is not entirely depleted, the remaining units are kept so that they can be used for future manufacturing. Only contract owners can generate token batches, as a contract always corresponds to a specific producer's or supplier's product only s/he produces. However, token owners can split, merge, transfer and consume batches.

As the product recipe should be immutable to guarantee the comparability of minted batches, it cannot be changed after the token contract's deployment and applies to all minting procedures. When minting a token, the amount of units within the batch, as well as the used components' information need to be passed, as exemplified in Algorithm 1. Within the minting function, it is first ensured that the declared array sizes are equal and comply with the recipe size (line 2). Afterwards, it is checked for each component if enough units were used according to the recipe (lines 4-5). Furthermore, used contract addresses and those defined in the recipe are compared (line 7). If they do not match, it is assumed that the recipe used a certificate contract. In this case, the certificate is called to ensure the used component is certified (lines 8-9). If it is not certified, the entire function reverts. Otherwise, the component's contract is called and the batch is reduced according to the amount that was declared (line 11). Finally, a batch identifier and a token holding owner information as well as its size is created (lines 12-15).

3.3. Certified goods

Some goods are equally usable in a manufacturing process, because they are equal in type. They could also be equal in terms of compliance with standards indicating similar quality. However, the goods may originate from different sources. With the help of a certificate contract, multiple token contracts can be defined to be equal. This introduces an ontology for defining product inputs during token creation. In this aspect, our approach bears similarity with the architecture proposed by Bechini et al. [12]. The certificates can be used in place of a specific good when defining a recipe as part of a manufacturing process. Once a batch is to be created, the input tokens are checked on whether they conform to the specified certificate.

In addition to creating comparability between products, supplier certificates are essential for ensuring quality in sourcing [29]. As a result, the proposed system can be used for quality management over multiple tiers. In this regard, certificates that have been

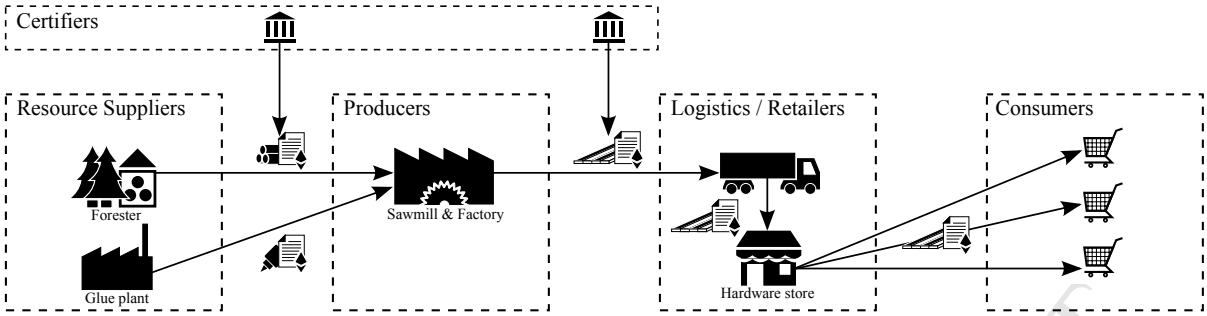


Fig. 1: Projecting the production process onto digital tokens using smart contracts in an example supply chain

issued enforce the authenticity of claimed production processes. As product tokens consisting of other components can only be minted if respective inputs are consumed, this mechanism enforces a link to physical processes. However, as this is not the case for primary resources, certifiers monitor the correctness of applied information.

3.4. Incentives to store external data

We propose additional product information to be stored externally in P2P storages such as IPFS or Swarm. Currently these systems do not offer any inbuilt incentive mechanisms for users to host or distribute data. To mitigate this issue, we utilize characteristics of the proposed supply chain traceability system. Each participant in the supply chain that owns tokens of a given contract has an interest in providing comprehensive data to increase its value. Therefore, not only the contract owner hosts external data in the P2P system, but also suppliers, customers, retailers and so forth. As a result, data availability can be guaranteed by creating intrinsic intensives that are specific to the given use case and provide product-specific information to participants.

3.5. Roles

A set of roles emerge from the actions participants can perform in the system. A user may act in a single or multiple roles. Figure 1 illustrates these roles based on a simplified example taken from the wood processing industry. It contains participating entities as icons, surrounded by dashed lines separating the roles. Edges represent interaction, such as a token transfer or a certification. Smart contract symbols attached to every good indicate that a smart contract exists on a blockchain representing batches of it as digital tokens. We identify the following roles that are also summarized in Table 1, displaying the set of functionalities typically executed by the corresponding entity.

1. *Resource suppliers* create goods without any input. When a forester cuts trees, several inputs, such as the labour or gasoline for machines, are required. However, they never become a physical part of the resulting product. As every supply chain will only have a partial view of the real world, resource suppliers that do not consider all inputs must exist in the model. In our example, the glue plant is also a resource supplier as it has no inputs, but in other scenarios its inputs may be modeled to originate from another entity. A resource supplier may split or merge batches before they are transferred to other entitites. In the example, the forester transfers batches of logs to the sawmill.
2. *Producers* in turn do require input goods for producing outputs. To represent this process on the blockchain, the processing industry entity first defines all the goods that are required to create a certain product. The input goods are acquired physically, while this process is digitally captured through token transfers. Consequently, the producer owns the input tokens which are required for the creation of a new token. In the example, the sawmill is a producer that acquires batches of logs as well as glue. To ensure that sufficient input tokens are present, it can merge batches. The sawmill is then able to produce a batch of edge-glued wood, which it could split and transfer to a logistics or retail company.
3. *Logistics* and *Retail* firms acquire tokens, but do not alter a good itself. For example, a batch of 100 units of edge-glued wood could be split, transferred to a logistics company and distributed to multiple hardware stores. If goods have been obtained by a wholesaler, merged batches could be transferred to other retailers. Individual customers could also keep ownership without consuming the item, in order to resell it to someone else. In this scenario, the customer is not strictly modelled as a consumer.
4. *Consumers* ultimately receive and consume products. This process results in the representing token batch to be deleted, meaning it can no longer be used as part of the supply chain. However, its provenance, can still be verified. If the retail customer is not modelled as part of the supply chain, a hardware store may act as a consumer: it removes the token from the supply chain when the item has been sold.
5. *Certifiers* issue certificates of equality for multi-

Algorithm 1: mint(u, I_c, I_b, I_u)

recipe : R_c− array of contract addresses of required inputs
R_u− array of required units
input : u− units to mint in batch
I_c− array of input contract addresses
I_b− array of input batches identifiers
I_u− array of input units
output: identifier of the newly added token

begin
  if $|I_u| = |I_b| = |I_c| = |R_c|$ **then** continue
  else revert
  for $i \leftarrow 0$ **to** $|R_c| - 1$ **do**
    if $u * I_{u_i} \geq R_{u_i}$ **then** continue
    else revert
    if $I_{c_i} = R_{c_i}$ **then** continue
    else
      if $I_{c_i} \in Instance(R_{c_i})$ *where* R_{c_i} *is a certificate contract* **then** continue
      else revert
    `/* Calling external contract to reduce batch content */`
    $(I_{b_i} \in Instance(I_{c_i})) \leftarrow consume(I_{u_i})$
  $tokenId \leftarrow keccak256(u, I_c, I_b, I_u, msg.sender, time)$
  $token.owner \leftarrow msg.sender$
  $token.amount \leftarrow u$
  $tokens[tokenId] \leftarrow token$

ple goods. They are responsible for guaranteeing quality, following certain product standards or labor safety requirements. In practical terms, an official standardization organization could act as such a certifier. In our example, it verifies that logs coming from the forester meet certain quality guidelines. It doesn't own, create or handle any tokens itself. It is also possible that the sawmill creates its own certificate for equally usable goods. It can then utilize all products that have been certified to qualify to a certain level of similarity.

3.6. Traceability

The physical production process is fully reflected on the blockchain. It works across multiple entities, as they cannot proceed without the correct inputs. Every step of the production process is accountable, and therefore traceability is not only provided for a single good, but also for its ingredients. Depending on the interval in which new blocks are added to the blockchain, the block creation timestamp informs about the time and date a given product has been manufactured. As products are handled in batches, input information is available by batch. For example, a batch of edge glued wood could be traceable to a batch of 100 logs. In order to accomplish traceability for single goods, it should be declared as its own batch. However, fine-grained traceability comes with the cost of higher transaction counts.

Roles	Actions			
	Create	Consume	S/M/T	Certify
Res. Supplier	✓		✓	
Producer	✓	✓	✓	
Logistics			✓	
Consumer		✓		
Certifier				✓

Table 1: Role definition and actions performed within the system (S/M/T = Split / Merge / Transfer)

4. Implementation

To achieve advanced traceability in supply chains, we propose a set of solidity contracts which are deployable on any EVM compatible blockchain. To achieve widespread compatibility, the ERC 721 interface is implemented. As a result, standard Ethereum wallets supporting the interface can be used for managing and trading tokens. Since minting of tokens is not provided by default, such actions have to be taken by using a tailored user interface. To extend the feature set, we inherit from the well tested token contract implementation that is supplied by OpenZeppelin[11], a library for secure smart contract development. In our modified implementation we suggest the representation of product batches as unique data structures which hold certain characteristics, such as the goods which have been consumed during the production process. Each batch of products corresponds to one token that holds unique features. Resource suppliers and manufacturers hold their own set of token contracts, one for each kind of product. To facilitate the deployment of novel token contracts, they can utilize a factory contract. While a token contract's ownership always remains with the initial deployer, the batches may be traded and change owners.

To clarify our approach, we follow the example provided in our concept, but select a subset of processes for improved readability. The simplified example contains three parties who operate along a supply chain in six steps. The first entities involved are resource suppliers, namely a forester and glue plant who produce and sell wood and glue correspondingly, as presented

[11] https://github.com/OpenZeppelin/openzeppelin-solidity

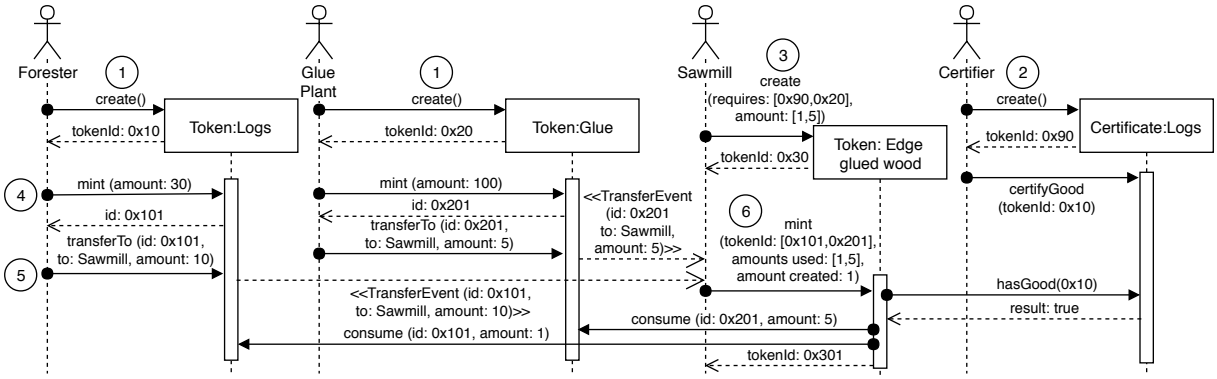


Fig. 2: Example Use Case: After a Certifier approves a token that is used for defining required inputs in novel tokens

in Figure 2. The parties reflect production and trading processes along the supply chain following these subsequent steps:

1. Token contracts are created that hold all produced batches of wood and glue that were produced by that specific forester and glue plant. As these goods do not involve any direct inputs, no additional products are required to add a batch of wood or glue in the contract.
2. A certifier approves the forester's log tokens if the organization's measures apply. This step is essential for defining inputs for goods which need to comply with certain standards. For instance, when a sawmill creates a board token contract, it may want to guarantee a high level of wood quality for all boards produced. Instead of specifying a particular wood contract, certificates can be used.
3. The sawmill defines two required inputs, any logs that are certified by the certifier and the specific glue plant's glue. Through this mechanism, the sawmill is not bound to a single supplier but is enabled to either define a certificate to which consumed products need to comply or to directly specify supplier's good.
4. To add a batch of logs, the forester does not require any inputs, as s/he acts in the role of a resource producer. The created batch may be split thereafter, resulting in two or more new batches. Split operations are not visualized in Figure 2 to prevent introducing further complexity.
5. The created batch of ten units is sent to the sawmill which is notified through an event triggered in the *transfer* contract function. The glue plant performs corresponding steps.
6. To add a batch of edge glued wood, the sawmill needs to define the wood and glue used during production. The `addbatch` function first checks if the defined wood is certified by querying the certifier's certificate contract. If the result is positive, wood is consumed, i.e. the wood token's `consume` function is called, reducing the overall amount of wood the forester's wood contract holds. The consumption of tokens is only possible for owned token batches, which in our example is the case as the forester transfered the ownership in Step 5. As the glue token was defined directly without utilizing a certificate before, the token is reduced instantly.

As a result, the forester holds twenty units of wood, the glue plant owns fifty-nine units of glue while the sawmill holds nine units of wood, no glue and one unit of edge glued wood.

4.1. Light weight implementation

Implementing the ERC 721 interface holds various benefits such as compatibility to wallets supporting the standard and the possibility to extend well tested contracts such as those supplied by OpenZeppelin. However, not all functionality specified is required for the given use-case. For instance, the `ownerOf` function permits retrieving the owner of a specific token, or in this case product batch. In the specific contract implementation, this results in the requirement for creating a mapping from batch to owner, while otherwise only the opposite direction would be necessary to keep track of owned tokens. In scenarios that require large amounts of batches, these additional storage operations produce increased gas costs for transactions such as minting and transferring. Furthermore, the need for additional function code increases deployment costs for contract owners.

To mitigate the involved overhead, a light weight contract is implemented as an alternative to the more general ERC 721 compliant version. Here, only those functions deemed necessary for the given use-case are implemented, producing lower overhead and therefore decreased gas costs. The downside of this approach is the lost compatibility provided by the standard interface. Section 5.1 provides an evaluation of the involved gas costs in both approaches. In the following, we refer to the light weight implementation as it is well tailored to supply chain scenarios.

4.2. Design Decisions

Certifiers deploy their own contracts that hold certificates for multiple goods which possess similar characteristics. Alternatively, these organizations could issue signed certificates which are added to corresponding products' smart contracts. This approach enables observers to receive all certificates a good holds without querying a third contract. However, the certifier's signature has to be verified, adding overhead. When storing certified products in a dedicated certificate contract, the certificate has to be known before querying for a specific product, while in the former approach this is not the case. Nevertheless, certificate contracts provide an important property for providing an ontology of products' similarity. Hereby, certificates can be declared as inputs for tokens rather than defining a specific good which would enforce a single supplier for sourcing.

For generating unique batch identifiers, resources, sender address and time are hashed using a SHA-3 hash function. While hashes bear the disadvantage of possible collisions, using a counter would require another storage operation every time a new batch is created. As the possibility of a collision within a single token contract with a byte length of 12 is in the measures of approximately $1 : 7.9 * 10^{18}$, we assume it to be small enough for relying on this mechanism to decrease operational gas costs.

4.3. Smart contract optimizations

In the EVM, gas costs are assigned to each operation. Despite deploying new contracts, storing variables in the contract storage is the most expensive instruction [20]. To decrease operational costs, it is therefore desirable to minimize such costly procedures. As a first measure, we use events rather than storage in case the data is not accessed within the contract (cf. Section 4.4). Secondly, the word size in the EVM is 32 bytes long so that storing smaller types results in costs reflecting the full word size. The solidity optimizer[12] merges smaller variables to mitigate the involved expenses under certain conditions. We analyze the impact of this approach in Section 5.1.2.

When storing required inputs for creating tokens, we demand the inputs' contract address, batch identifiers and the amounts needed to create one product unit. Contract addresses in Ethereum are 40 hexadecimal digits long and therefore require 20 bytes for storage. While usually the remaining 12 bytes are padded with zeros, we use 12 byte long batch identifiers and concatenate them with the corresponding contract added, resulting in a single 32 byte write operation, as illustrated in Figure 4. The size of 24 hexadecimal digits provides sufficient space for storing unique identifiers and is derived from SHA-3 hash in the contract. The concatenation and respective split operations are implemented using in-line assembly to provide a gas efficient calculation.

The use of smart contracts should be decoupled from user interaction. Adding a new product is therefore conducted through a factory contract which deploys a corresponding token contract on the blockchain. Hereby, the maintainability is increased, as front-end applications do not need to handle the contract code and following migrations are deployable through the factory contract.

4.4. Events for analytics

As the main target of our approach is to increase transparency in supply chains, information about minted tokens, their components and ownership has to be accessible. Intuitively, batch inputs would be stored within their corresponding structs and pulled via getter methods. As provenance information is not required by contracts but only queried for analysis purposes, the token contract emits events rather than storing all information in order to decrease gas costs. Declaring topics when emitting events enables filtering and searching for attributes but are more expensive than using a raw data format. Even though events are not stored, they are reproducible, verifiable and easily observable due to the use of bloom filters for receiving relevant blocks in the blockchain [20].

We utilize the emitted events for providing traceability of products by recursively querying the inputs that have been declared for the token's creation. As the contract enforces the consumption of these inputs, the real world production process is reflected. Respectively, receiving all inputs in a recursive manner results in a tree of production inputs and unveils all resources used.

4.5. Mobile and web application

In order to increase accessability, a mobile and web client to interact with the system in a real world setting is implemented. While the former targets resource suppliers, producers, retailers and logistics, the latter focuses on consumers' requirements.

As illustrated in Figure 3, the web application facilitates the discovery of existing goods and the creation of new types of goods. It also supports batch creation based on existing goods and can generate QR-codes for individual product batches. In practical terms, QR codes encoding specific batch identifiers can be printed and added to physical product batches, that in turn can be scanned with a smartphone running the mobile client. Furthermore, individual batches can be inspected: a tree visualization reveals which other product batches have been used to manufacture the selected good, enabling the possibility to trace an individual product's origin as well as its components.

After scanning a QR code using the mobile client, information about the quantity within the individual

[12] http://solidity.readthedocs.io/en/develop/miscellaneous.html#internals-the-optimizer

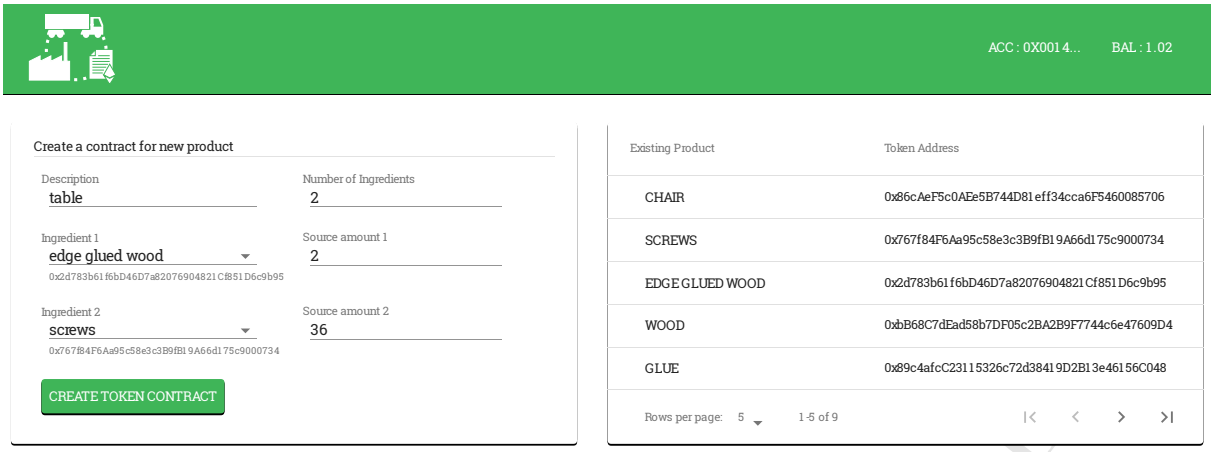


Fig. 3: Creating a new product type in the web application (a wooden table represented by a token contract). As the application automatically discovers existing products, possible inputs can be chosen from a drop down list.

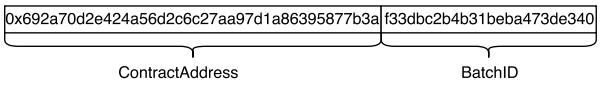


Fig. 4: Contract addresses and batch identifiers are saved in a single 32 byte array to minimize storage operations

batch as well as its ingredients are presented. To achieve more elaborate information about resources used in the product, users can iteratively follow a tree structure representing the components with raw materials as leafs in the tree.

For communicating with the presented smart contract, Ethereum's JavaScript library Web3[13] is used. It relies on an injected Web3 instance that is connected to an Ethereum node providing a JSON RPC interface. For public networks, such instances are provided by Infura[14]. In a consortium centered on supply-chain traceability, each partner could supply such managed instances. Thus, the individual client can, but doesn't necessarily have to run its own blockchain instance.

5. Evaluation

The prototypical implementation of a supply chain traceability system based on smart contracts raises various questions regarding deployment and maintenance costs as well as scalability. As such drawbacks are frequent issues when utilizing blockchain technologies [8], we focus on these topics in the following sections for evaluating the proposed system's applicability.

Information distribution in decentralized networks is limited by its weakest link in terms of bandwidth, storage and processing capabilities. In the blockchain context, block time and size are adjusted to satisfy the majority of nodes participating in the network. While Bitcoin pursues to produce a single block of one megabyte every ten minutes [30], Ethereum permits mining blocks every 15 seconds. In order to limit ledger growth and processing steps performed within a single block, an upper bound is determined by Ethereum's gas limit (cf. Section 2.2). In recent years, the public Ethereum blockchain could prove its robustness in an unpermissioned environment under the given configuration. Therefore, we evaluate the proposed concept's scalability on the same limitations in the form of gas as the public Ethereum network.

5.1. Gas costs

For analyzing the involved costs, we assume gas consumption as defined in Ethereum's yellow paper [20]. We refrain from converting measured gas costs to Ethereum's native currency or conventional currencies due to the lack of expressiveness. Such conversions would not only be subject to fluctuating exchange rates but also changing gas costs which depend on various factors such as current blocks' gas use and waiting time until a transaction is mined (cf. 2.2). In addition, the proposed approach is independent from the underlying blockchain and may be run for example in a permissioned manner.

We differentiate between deployment and batch creation costs. Since participants create a contract for each type of product they offer, deployment costs are only relevant for novel products. The actual deployment costs depend on the amount of inputs declared, as they have to be stored in the contract. In the following subsections we focus on the approaches presented in the implementation - namely ERC721, a light weight version of ERC721 and using a library contract. Additionally, we consider the case of optimized contract storage during deployment (uint32 arrays), and storing batch creation events on chain (storage-based), as well as the impact of multi-tiered supply-chains.

[13] https://github.com/ethereum/web3.js/
[14] https://infura.io/

5.1.1. ERC721 and lightweight approach

Following the OpenZeppelin implementation of ERC721, we can observe the highest deployment costs. This is due to functionality like bi-directional storage, and functionality not needed for the use case as described in Section 4.1. As batch creation also makes use of events, the implementation performs best, very similar to the lightweight approach.

When removing the unneeded functionality to obtain a lightweight implementation, we lose compatibility with the standard. However, as Figure 5a) shows, we can reduce the deployment costs.

5.1.2. Storage optimization and events

To ensure the right amount of goods have been consumed for creating a corresponding token, these requirements need to be stored within the contract. By default, the approaches ERC721, Lightweight and Library use uint256 values for storing input amounts. This results in a linear growth in gas costs with an increasing number of inputs. Optimizing storage use with uint32 arrays – as described in Section 4.3 – we can reduce deployment costs as seen in Figure 5a): we observe a lower slope but a step every eight inputs. This is due to the fact that the optimizer concatenates up to eight variables in a single storage location before allocating a new one. However, when storing only few variables, the overhead for merging variables in a single location exceeds its benefits. This mechanism is only capable of concatenating types with the size of 8 bytes or multiple of it [34].

For logging batch creation, we can resort to using events instead of storage. Comprehensive traceability information can be obtained by retrieving emitted events from a blockchain node using the web3 API. As this data is not required for smart contract operations, using events is sufficient so that transformation information is not stored within the contract, leading to decreased operational costs. Figure 5b) depicts a comparison of multiple approaches with respect to the amount of input goods. We observe a significantly steeper increase in gas costs when storing inputs in comparison to emitting events. While gas costs grow linearly with both approaches, we observe a factor of $39,340gas$ for each added input in the classic storage approach but only $19,698gas$ for using events, so just about half of the former. The base costs for adding a batch with no input goods account for $92,634gas$.

5.1.3. Library and Proxy contracts

Attached costs for deploying and executing code entails novel requirements to software engineering. As contract code needs to be stored for every contract instance created on the blockchain, the code size should be decreased to minimize gas costs. To do so, we split the token contract into a proxy contract with minimal size and a library contract that includes all necessary logic. When creating a new token contract instance, only the proxy has to be redeployed which decreases costs due to the outsourced complexity of execution logic into the library that is only deployed once for all instances. The state remains within the proxy, ensuring the separation between general contract logic and product-specific information. Figure 5a) indicates the gap between deployment costs in the lightweight approach and using a library which is constant at $415,854gas$ for any input size. While using a library results in significant gas savings when deploying a token contract, it involves increased costs for execution. As the proxy contract forwards methods to library functions including its state, more execution steps are required, increasing gas costs. This is especially the case for goods with large amounts of inputs, as pictured in Figure 5b). We observe that the utilization of a library is only beneficial if the token contract is not intensively used after deployment. With base costs of $88,202gas$ for adding a new token batch in the classic approach and $92,634gas$ when using a library and factors of $19,698$ and $27,427gas$ for each input respectively, we set up the following equation,

$$x = \frac{415,854gas}{4,432gas + 7,729gas * |S|}$$

where x is the equilibrium between using a library contract and classic deployment and S is the set of inputs. As a result, when having no inputs, the library is only cheaper in terms of gas for up to 93 minting calls while for ten inputs its only cheaper for up to 5 calls. In addition, token owners transfer split, merge and consume batches which takes more gas using a library due to the involved overhead in the proxy contract.

5.1.4. Impact of tiers

To picture more complex scenarios, we create product tokens with two inputs on multiple tiers. Consequently, the resulting relationships can be represented as a binary tree with its height relating to the level of tiers tracked in a supply chain. We create batches with multiple tiers and compare the gas costs with a single batch that includes all inputs directly instead of using multiple tiers. The resulting gas costs confirm our assumption that gas costs depend on the amount of vertices and egdes in the sourcing tree. As a result, the overall gas costs do not depend on the amount of tiers considered in a supply chain but on the amount of sources, allowing precise cost estimations for maintaining a traceability system.

5.2. Scalability

The application's scalability depends on a multitude of variables. Most importantly, the decision of utilizing a public or permissioned ledger depends on the desired scope, accessibility and privacy considerations. In case a public ledger is preferred, the scalability highly depends on the involved gas costs (cf.

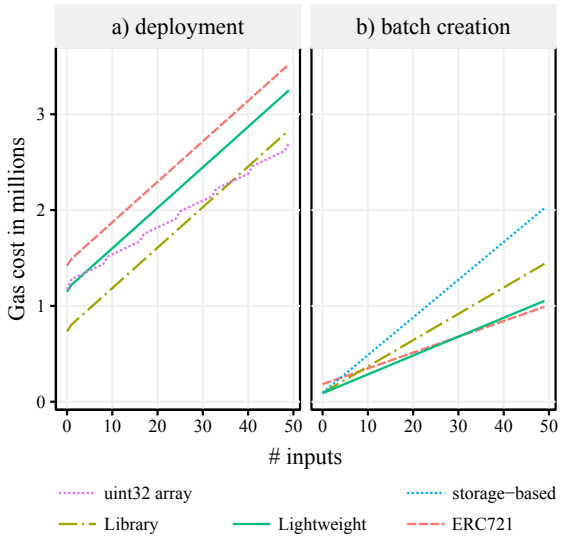


Fig. 5: Gas costs for contract deployment and batch creation by implementation and number of inputs

Section 5.1). The scalability of a permissioned ledger varies with its implementation of block size limitations, consensus algorithm and mining time, determining attributes such as throughput and ledger size. Tests using the Ethereum client geth[15] unveiled a plethora of aspects that influence ledger growth. For instance, the file system's block size affects the ledger size as files created by the client may fill only fractions of it, leading to increased disk space consumption.

Varying batch sizes influence the systems scalability in terms of throughput, ledger size and potential gas costs. Consequently, small batch sizes, as required for tracing single goods, negatively affect the system's performance.

The gas limit determines the amount of operations that can be performed in a single block in Ethereum. For estimating the scalability of the proposed supply chain traceability concept, we assume a gas limit of eight million gas, as applied in the public mainchain at the time of writing. While deploying token contract instances is an essential element of the introduced system, we expect such operations to occur infrequently due to the minor measure of newly introduced product types. Therefore, the main factors restraining the system's scalability originate from batch creation and token transfer operations. While gas costs related to batch creation depend on the amount of input tokens, the gas required for transferring a token is constant at $32,426 gas$. Thus, when considering transfer operations only, a theoretical maximum of 246 transactions can be performed per block or 1,416,960 transactions per day at an average block time of 15 seconds.

As scalability is a general challenge in today's blockchain systems [31], supply chain participants could implement industry-specific permissioned ledgers to mitigate related shortcomings. Due to intersections of industries and bidirectional dependencies however, interoperability between such specialized chains is required. Relay networks like Polkadot [32] and Cosmos [33] facilitate accessibility and portability of assets between independent blockchain instances. For instance, following the example presented in Section 3.5, a forester and sawmill may deploy their respective token contracts on a blockchain that is targeted towards wood processing, while the glue plant utilizes a ledger assigned to the chemistry industry. In such a case, the glue token has to be transferred to the wood processing ledger in order to be applied in the sawmill's token recipe. As a result, a plethora of incorporating blockchains permit employing industry-specific configurations to facilitate a scalable application landscape.

6. Discussion and Outlook

6.1. Incentives

Employing supply chain traceability on a blockchain implies transparency to an extent that raises questions regarding participants' willingness to disclose otherwise confidential information. The degree to which information is published depends on the underlying blockchain. Using a public ledger entails unveiling information to a large audience, while utilizing a permissioned ledger restricts the number of participants but may still involve revealing information to competitors. Therefore, the proposed solution is only applicable for use cases in which the desired transparency requirements exceed the threat of losing competitive advantage by disclosing internal information. This may be the case if customers have a high interest in resource provenance or an increased quality level in an environment with limited trust.

The main incentive for participating in a traceability system is to increase customer's trust in the delivered product. For example, customers buying food are willing to buy larger quantities and pay higher price if they are enabled to reconstruct a product's provenance [35]. With increasingly complex supply chains, guaranteeing high quality has become a difficult endeavor, generating the need for high transparency.

6.2. Outlook

The proposed solution should be understood as a foundation for blockchain-based supply chain management systems that can be built upon. In its current stage it does not yet cover functionalities that may be required in a real world setting. We anticipate the following features to be of interest:

- *Payment for goods*: In our solution, payments are not considered. It is questionable whether it would be desirable for a business to open its

[15] https://github.com/ethereum/go-ethereum

books on how much has been paid for the acquisition of inputs for a manufacturing process. If purchases were public however, inefficiencies of the market would be transparent and could be exploited.

- *Shrinkage*: If a good is damaged or lost, the system currently has no suitable mechanism to capture such an event, other than letting the affected business simply consume the respective tokens in order to remove them from the supply chain.

- *Ownership vs. possession*: It may be desirable to let a logistics firm possess goods that are owned by another party. This raises additional questions if such a logistics firm suffers from shrinkage.

- *Packaging*: There are scenarios in which goods may need to be packaged together and extracted at a later stage. With the current solution, a packaging process can be implemented similarly to a manufacturing process, but the extraction of the original goods is not possible, inhibiting traceability.

7. Conclusion

We have proposed a blockchain-based supply chain management system that enables tracing and tracking goods including their transformation in the production process using smart contracts. The system provides comprehensive provenance information by projecting product compositions onto the blockchain in the form of tokens. Hereby, shortcomings of current traceability systems regarding isolated data storage and lacking transformation information are tackled. Defining compositions for creating products and enforcing them using smart contract enables the documentation of consumed resources in the production process. Through this mechanism, products are not only traceable from production to retail but starting from resource exploitation. As a result, transparency is generated along the supply chain, providing comprehensible production information.

[1] F. Dabbene, P. Gay, C. Tortia, Traceability issues in food supply chain management: A review, Biosystems Engineering 120 (2014) 65 – 80.

[2] J. Gualandris, R. D. Klassen, S. Vachon, M. Kalchschmidt, Sustainable evaluation and verification in supply chains: Aligning and leveraging accountability to stakeholders, Journal of Operations Management 38 (2015) 1 – 13.

[3] European Parliament , Regulation (ec) no 178/2002 of the european parliament and of the council of 28 january 2002 laying down the general principles and requirements of food law, establishing the european food safety authority and laying down procedures in matters of food safety, http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32002R0178\&from=EN (2002).

[4] S. Appelhanz, V.-S. Osburg, W. Toporowski, M. Schumann, Traceability system for capturing, processing and providing consumer-relevant information about wood products: system solution and its economic feasibility, Journal of Cleaner Production 110 (2016) 132 – 148, special Volume: Improved resource efficiency and cascading utilisation of renewable materials.

[5] S. A. Abeyratne, R. P. Monfared, Blockchain ready manufacturing supply chain using distributed ledger, International Journal of Research in Engineering and Technology 05.

[6] K. Korpela, J. Hallikas, T. Dahlberg, Digital Supply Chain Transformation toward Blockchain Integration, in: Proceedings of the 50th Hawaii International Conference on System Sciences, 2017.

[7] F. Glaser, Pervasive Decentralisation of Digital Infrastructures: A Framework for Blockchain enabled System and Use Case Analysis, in: Proceedings of the 50th Hawaii International Conference on System Sciences, 2017.

[8] K. Wüst, A. Gervais, Do you need a Blockchain?, Cryptology ePrint Archive, Report 2017/375 (2017).

[9] H. M. Kim, M. Laskowski, Toward an ontology-driven blockchain design for supply-chain provenance, Intelligent Systems in Accounting, Finance and Management 25 (1) (2018) 18–27.

[10] K. Toyoda, P. T. Mathiopoulos, I. Sasase, T. Ohtsuki, A Novel Blockchain-Based Product Ownership Management System (POMS) for Anti-Counterfeits in The Post Supply Chain, IEEE Access PP (99).

[11] N. Hackius, M. Petersen, Blockchain in Logistics and Supply Chain: Trick or Treat?, in: Proceedings of the Hamburg International Conference of Logistics (HICL), 2017, pp. 3–18.

[12] A. Bechini, M. G. Cimino, F. Marcelloni, A. Tomasi, Patterns and technologies for enabling supply chain traceability through collaborative e-business, Information and Software Technology 50 (4) (2008) 342 – 359.

[13] A. Gunasekaran, E. Ngai, Information systems in supply chain integration and management, European Journal of Operational Research 159 (2) (2004) 269 – 295.

[14] P. Helo, B. Szekely, Logistics information systems: An analysis of software solutions for supply chain coordination, Industrial Management & Data Systems 105 (1) (2005) 5–18.

[15] Y. K. Tse, K. H. Tan, Managing product quality risk in a multi-tier global supply chain, International Journal of Production Research 49 (1) (2011) 139–158.

[16] C. Costa, F. Antonucci, F. Pallottino, J. Aguzzi, D. Sarriá, P. Menesatti, A Review on Agri-food Supply Chain Traceability by Means of RFID Technology, Food and Bioprocess Technology 6 (2) (2013) 353–366.

[17] Y. Yuan, F. Y. Wang, Towards Blockchain-based Intelligent Transportation Systems, in: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), 2016, pp. 2663–2668.

[18] N. Szabo, Smart contracts.

[19] V. Buterin, A next-generation smart contract and decentralized application platform, Ethereum Project White Paper.

[20] G. Wood, Ethereum: A Secure Decentralised Generalised Transaction Ledger, Ethereum Project Yellow Paper.

[21] M. Ali, J. C. Nelson, R. Shea, M. J. Freedman, Blockstack: A Global Naming and Storage System Secured by Blockchains, in: USENIX Annual Technical Conference, 2016, pp. 181–194.

[22] H. L. Lee, S. Whang, Information sharing in a supply chain, International Journal of Manufacturing Technology and Management 1 (1) (2000) 79–93.

[23] J. Benet, IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3) (2014).

[24] Swarm guide – Swarm 0.3 documentation, https://swarm-guide.readthedocs.io/en/latest/ Accessed: 01.10.2018.

[25] Protocol Labs, Filecoin: A Decentralized Storage Network, Tech. rep., https://filecoin.io/filecoin.pdf. Accessed: 20.8.2018.

[26] C. Van Dorp, Tracking and tracing business cases: Incidents, accidents and opportunities, Proceedings of EFITA Conference (2003) 601–606.

[27] F. Tian, A Supply Chain Traceability System for Food Safety Based on HACCP, Blockchain & Internet of Things, in: 2017 International Conference on Service Systems and Service Management, 2017. doi:10.1109/ICSSSM.2017.7996119.

[28] F. Tian, An Agri-food Supply Chain Traceability System for

China Based on RFID & Blockchain Technology, in: 2016 13th International Conference on Service Systems and Service Management (ICSSSM), 2016.

[29] I. J. Chen, A. Paulraj, Understanding supply chain management: critical research and a theoretical framework, International Journal of Production Research 42 (1) (2004) 131–163.

[30] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, http://www.bitcoin.org/bitcoin.pdf. Accessed: 23.9.2018 (2008).

[31] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, R. Wattenhofer, On scaling decentralized blockchains, in: J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, K. Rohloff (Eds.), Financial Cryptography and Data Security, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 106–125.

[32] G. Wood, Polkadot: Vision for a Heterogeneous Multichain Framework, https://polkadot.network/PolkaDotPaper.pdf Accessed: 27.11.2018 (2016).

[33] J. Kwon, E. Buchman, Cosmos: A Network of Distributed Ledgers, https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md Accessed: 27.11.2018 (2018).

[34] L. García-Bañuelos, A. Ponomarev, M. Dumas, I. Weber, Optimized Execution of Business Processes on Blockchain, in: J. Carmona, G. Engels, A. Kumar (Eds.), Business Process Management, Springer International Publishing, 2017, pp. 130–146.

[35] Y. C. Choe, J. Park, M. Chung, J. Moon, Effect of the food traceability system for building trust: Price premium and buying behavior, Information Systems Frontiers 11 (2) (2009) 167–179.