Verifying the Integrity of Hyperlinked Information using Linked Data and Smart Contracts

Christoph Braun and Tobias Käfer

Institute AIFB, Karlsruhe Institute of Technology (KIT), Germany
uvdsl@student.kit.edu, tobias.kaefer@kit.edu

Abstract. We present an approach to verify off-chained information using Linked Data, Smart Contracts, and RDF graph hashes stored on a Distributed Ledger. We use the notion of a Linked Pedigree, i. e. a decentralised dataset for storing hyperlinked information, as modelling foundation. We evaluate our approach by comparing different ways to build the Smart Contract. We develop a cost model and show, based on our implementation, that for managing multiple Linked Pedigree instances, a single larger Smart Contract is superior to multiple smaller Smart Contracts for supply chains shorter than 50 participants.

1 Introduction

Chained value-creation networks are commonplace in many industries. Consider e. g. supply chain networks in logistics or production systems, where goods and services are handed over decentrally between different independent parties to deliver goods and services to the customer. In such networks, transparency is gaining importance. Customers demand verifiable[1] information on where their food comes from (track & trace), or recall campaigns need to be organised fast and specifically. Recently, distributed ledger-based solutions have gained attention, e. g. TradeLens by IBM and Maersk for global trade networks[2]. But sharing information on a distributed ledger may not always be desirable: As in a distributed ledger, every participant stores a copy of the whole ledger, data sovereignty and privacy become an issue. Moreover, storing data on the distributed ledger is expensive, which calls for so-called "off-chaining" of data [3], i. e. storing data outside of the distributed ledger while keeping the distributed ledger in the loop by storing hashes on the ledger. For off-chaining, to not complicate matters, a uniform access mechanism would be desired. Linked Data is a light-weight standard-based way to publish data in a decentralised fashion, where

[1] To verify: "Make sure or demonstrate that (something) is true, accurate, or justified." (Oxford Living Dictionary) We assume truth of the information on the distributed ledger to be established. Then, we allow for verification that the information has not been changed.

[2] "IBM teams with Maersk on new blockchain shipping solution", https://tcrn.ch/2vRLFLT

access control can be easily implemented. Hence, we ask: Can we combine the verification capabilities of the distributed ledger with Linked Data management?

Transparently provided information is important, e. g. in the food sector, where society demands more transparency regarding details on products and their transportation [1]. More general, in retail, the transparency in production and transport of consumer goods and retail products is a important factor of customer decisions [10, 14]. Regulation authorities discuss such product transparency and documentation to be required in the future [4]. But that information needs not just to be public. Customer trust needs to be ensured, where structural assurances [5] such as the mathematical foundations of distributed ledgers can serve as basis. Publicly shared information has high economic potential in the logistics domain, e. g. by addressing the bullwhip effect, but is hindered by the need for privacy of businesses [8, 11]. Hence, a more cautious approach to share data, like disclosing data only to a selected number of persons, may unlock some of the benefits. But even if organisations are willing to share information, interoperability of the information systems is an issue [8, 11, 17]. Hence, the flexible data model of RDF and the standardised light-weight protocol HTTP can reduce friction. If RDF is not available yet in an organisation, lifting of existing data to semantic models has already been proposed in [7].

Previous works in the intersection of Semantic Web and Distributed Ledger, e. g. at the Linked Data and Distributed Ledgers workshop series (LD-DL)[3] have not considered off-chaining of data. Previous works in off-chaining of data are often built using distributed hash tables [3], where the problem of data sovereignty arises just like with storing data on the chain.

Our approach consists in the following parts (this unique combination and 2, 4, 5, and 6 are the contributions of this paper):

1. We use Linked Data, i. e. RDF accessible using HTTP to store data off-chain in a decentralised fashion. Access control for data privacy can be layered on top, e. g. using HTTP authentication, or more recent approaches such as Web Access Control[4] or WebID+TLS[5].
2. We present a vocabulary that extends the Linked Pedigree ontology [15] to describe a product's handover history and the Ethereum Ontology[6] to describe an Ethereum distributed ledger.
3. We use the RDF graph hashing approach of [6] to connect the off-chained data with the distributed ledger.
4. We present a link-traversal based querying approach for verifying data on a Linked Pedigree off-chain.
5. We present a Smart Contract, i. e. code that can be executed on the Distributed Ledger, for verifying data using the Distributed Ledger.
6. We present a protocol to apply all of the above.

[3] Browse with `http://events.linkeddata.org/ldow-lddl/` as entry point.
[4] `https://www.w3.org/wiki/WebAccessControl`
[5] `https://www.w3.org/2005/Incubator/webid/spec/tls/`
[6] `http://ethon.consensys.net/`

The paper is structured as follows: First, we survey related work (Section 2). Next, we present an example (Section 3), which also introduces the protocol. Subsequently, we present the foundational definitions, on which we base our approach (Section 4). Then, we describe the components of our approach (Section 5), that is the vocabulary, the smart contract, and the graph traversal. We next evaluate our approach (Section 6) by developing a cost model, which we instantiate using an implementation. We then discuss our findings (Section 7). Last, we conclude (Section 8).

2 Related Work

In the intersection of supply chain and distributed ledger, there are two major initiatives started in collaboration with IBM. Both initiatives are based on the distributed ledger Hyperledger: TradeLens for global freight companies, and FoodTrust for agricultural goods. Both approaches have similar characteristics: All information (e. g. document filings, supply chain events, authority approval status, ...) is stored on the distributed ledger. As all nodes that are part of the distributed ledger have a full copy of the ledger, this hints at scalability issues. Both solutions support access rights to this data on the ledger. TradeLens is citing data interoperability as a challenge. While they incrementally move to UN's CEFACT vocabulary[7], our approach allows for using semantic technologies to achieve data interoperability using mappings between schemas. Similarly, `provenance.org`, an online service for track and trace of goods using a distributed ledger, stores all data on the ledger.

In the intersection of semantic technologies and distributed ledgers, different ontologies have been proposed to describe a distributed ledger: There is, e. g. GraphChain [16], BLONDiE[8], and EthOn[9]. Our approach uses parts of EthOn. Besides defining an ontology, the GraphChain [16] approach also allows to distribute RDF data onto a distributed ledger. Our approach however requires data to be provided as Linked Data, irrespective of the back-end.

In the intersection of semantic technologies and supply chain, the Linked Pedigree approach has been developed [15]. Linked Pedigrees are RDF graphs to describe trails of ownership of goods provided via HTTP. Moreover, the paper contains a protocol for using the thus described data in a supply chain. Our approach adds verification using distributed ledger technologies and hashing to Linked Pedigrees.

3 Example

We next describe an example to illustrate our approach. Imagine the following three steps in a simple supply chain:

[7] `https://blog.tradelens.comascomm/news/why-interoperability-matters/`
[8] `https://github.com/hedugaro/Blondie`
[9] `http://ethon.consensys.net/`

Item Creation: A fisherman creates an item, i. e. some fish.
Item Handover: The fish is handed over between supply chain partners, e. g. from the fisherman to a trucker to a local supermarket to the consumer.
Item Verification: At the store, the consumer verifies information about the fish as a decision-making support for the purchase. Verification could also be performed during each handover.

For the illustration, we look at the information transferred during these three steps: Within the first two steps, i. e. item creation and item handover, the item's physical history is described and published as Linked Data. The third step of item verification solely corresponds to the verification of that published information, and does not involve checking on the physical item itself. For brevity of the example, we leave out verification during the handover steps. The overall protocol is depicted in Figure 1. The top left group starting with "create item" in bold relates to the item creation. The next group starting with "transport item" relates to the handover. With "store item", the step for verifying the data starts, ending in the actual purchase.

3.1 Item Creation

The fisherman creates an supply chain item by catching the fish. They record information on the item and the catching process, e. g. fishing ground and time, builds an RDF graph from the information, and publishes the graph via HTTP. Thus, the initial part of the Linked Pedigree on the fish is formed. From this point, the creation procedure is the same as for any item handover in the supply chain.

3.2 Item Handover

When the fish is handed over, e. g. from the fisherman to the trucker that carries the fish to the market, an RDF graph with information on the hand-over is created and stored in the Linked Data store of choice of the party that owns the fish before the hand-over. The information is linked to the RDF graph describing the previous Linked Pedigree part, which contains an event that concerns this fish. Thus, we form a hyperlinked graph of the fish's product trail. Additional information may be included *ad libitum* in each step, e. g. information on the item's creation. For later verification purposes, a hash of the information is put into the Distributed Ledger using a Smart Contract.

3.3 Item Verification

Before actually buying the fish, the consumer may want to ascertain if the fish's information has not (maliciously) been tampered with, e. g. a retrospective adjustment to the cooling information was made. To this end, the consumer looks up the fish's information, which the supermarket provides in the form of a URI of a Linked Pedigree part. This Linked Pedigree part contains a reference to the

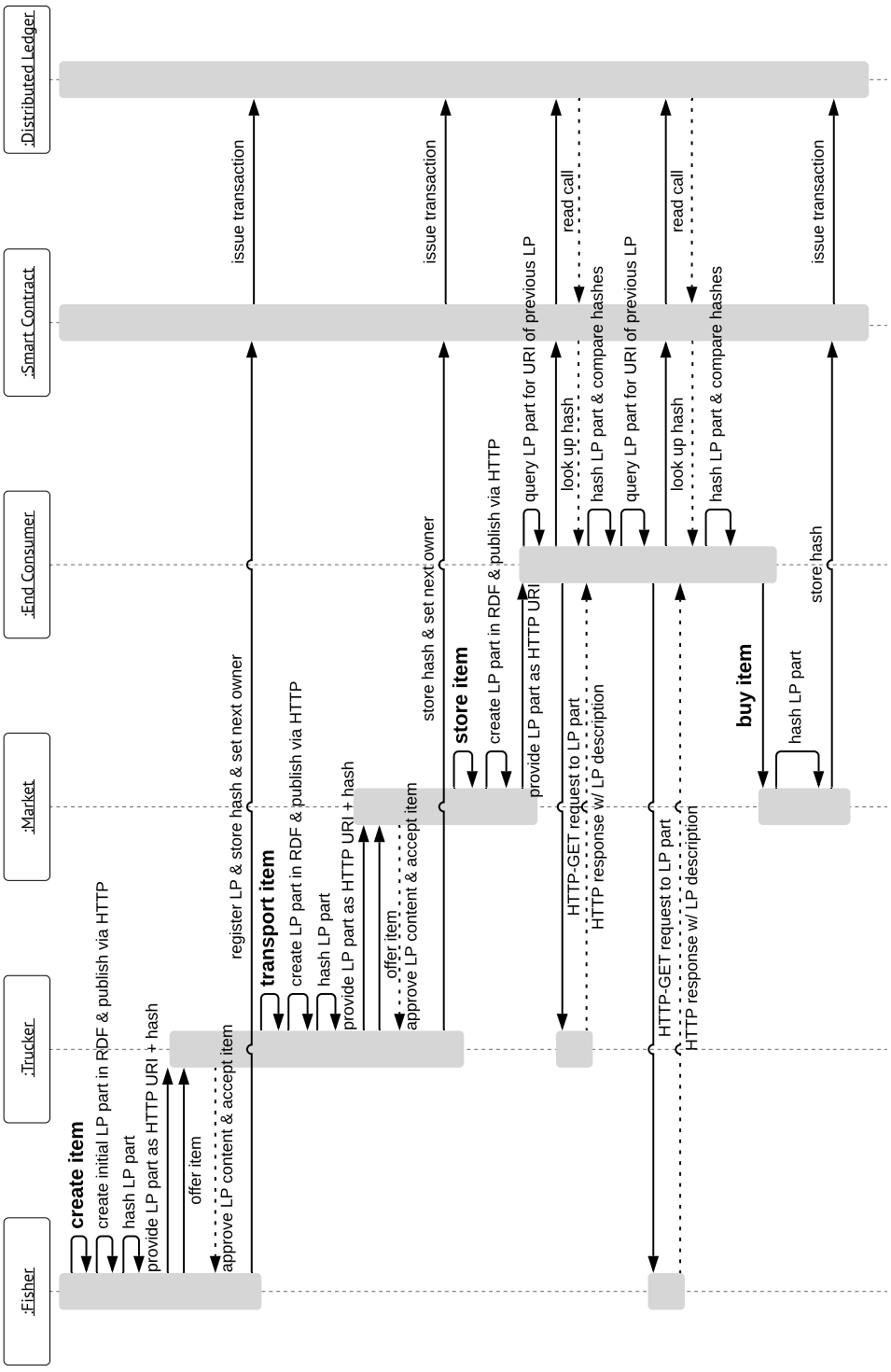

Fig. 1. Protocol for managing and verifying data along a supply chain.

previous part, which the end-consumer now dereferences. Consulting a Smart Contract, the customer can determine whether the retrieved information has not been changed since it was first published. By following the links in a Linked Pedigree to the respective previous Linked Pedigree and by dereferencing the corresponding identifiers, the customer can go back in the information trail on the fish right until the very beginning, i. e. the catchment. In each step, the customer can consult the Smart Contract to verify the integrity of the information provided.

This verification can be performed analogously at any point in the supply chain by any participant, starting at different points in the traversal.

4 Preliminaries

We base our approach on Linked Data, i. e. we make use of URIs, and provide hyperlinked RDF graph via HTTP. We build Linked Pedigrees in the form of RDF graphs. We store RDF graph hashes in a Distributed Ledger based on Ethereum using a Smart Contract.

4.1 Linked Data, URIs, RDF, and HTTP

We following the Linked Data principles[10]: We use Uniform Resource Identifiers[11] (URIs) as names for things. We use graphs expressed according to the Resource Description Framework[12] (RDF) to describe things. An RDF graph is defined as a set of triples. With $\mathcal{U}$ as the set of all URIs, $\mathcal{B}$ as the set of all blank nodes, and $\mathcal{L}$ as the set of all literals, a triple t can be defined as $t \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. In our examples, we use CURIEs[13] that allow to abbreviate URIs using prefixes[14] . We use the Hypertext Transfer Protocol[15] (HTTP) to dereference URIs and assume dereferencable URIs.

4.2 Linked Pedigree

A Linked Pedigree [15] is a trail of ownership of a product published as Linked Data described using terms from the OntoPedigree ontology. Each Linked Pedigree consists of different parts, i. e. instances of the class `p:Pedigree`, which reflect the different owners. The parts are assumed to be linked using the `p:`

[10] `https://www.w3.org/DesignIssues/LinkedData`
[11] `https://www.ietf.org/rfc/rfc3986.txt`
[12] `https://www.w3.org/TR/rdf11-concepts/`
[13] `https://www.w3.org/TR/curie/`
[14] We point to `prefix.cc` for resolving the prefixes. Moreover, we use `p:` as short for `http://purl.org/pedigree#`, `e:` as short for `http://ethon.consensys.net/`, and `x:`, as short for `http://people.aifb.kit.edu/co1683/2019/ld-chain/vocab#` for our extensions.
[15] `https://www.ietf.org/rfc/rfc7230.txt`

`hasReceivedPedigree` property. As each owner of a product may choose a storage provider of their liking, the Linked Pedigree can be regarded as a decentralised dataset. Each part of a Linked Pedigree bears a status, `p:Initial`, `p:Intermediate`, or `p:Final`. We show the terms of the OntoPedigree ontology that we use in this paper as part of Figure 2.

4.3 Hashing RDF graphs

To hash RDF graphs, we apply the approach of Hogan [6]. The approach allows for determining stable hashes of RDF graphs in the presence of isomorphism-preserving transformations of the graph, i. e. triple re-ordering and blank node renaming.

4.4 Distributed Ledger Technologies

Distributed Ledger Technologies is the umbrella term for distributed ledger concepts like blockchain or transaction-based directed acyclic graphs [9]. A distributed ledger is a distributed database in a decentralised network, where changes to the database, i. e. transactions, have to be approved by network nodes via a consensus algorithm [12]. This allows for secure processing of transactions between parties that do not trust each other. Furthermore, when new data is appended to the distributed ledger, timestamps and hash-based references to previous data are included. This meta data leads to a high degree of data integrity and imposes a high effort on retrospective modification of data [12]. In addition, as the database is replicated in full, every network participant can query their instance of the distributed ledger. Hence, all data and all associated changes are transparent to the entire network.

Ethereum Blockchain For our work, we choose Ethereum, a well-established blockchain implementation, that allows for the deployment of decentralised applications via Smart Contracts [2]. Ethereum allows for building private proof-of-work blockchains. Proof-of-work is a consensus algorithm based on expensive compute operations, which need to be executed for the approval of blocks of transactions. Participating in the consensus creation, i. e. approving blocks of transactions following a specified algorithm, here proof-of-work, is also referred to as “mining”.

Closely connected to the mining process in an Ethereum network is Ethereum’s internal cryptocurrency called “Ether”. Ether is used to pay transaction fees. Whenever a transaction is issued, the miner who approves the transaction is to be compensated for lending his computing power to the network. This network utilisation is measured in “gas”, ether’s internal utility value. Therefore, costs are typically given in gas. However, in private blockchain networks the amount of computing power necessary for proof-of-work based consensus can be set to a reasonably low level, such that transaction fees as well as energy cost for computation of the proof-of-work algorithm are kept within limit.

Ethereum Smart Contracts Ethereum also allows for the deployment of Smart Contracts. Smart Contracts allow for defining application logic that can be executed directly on the distributed ledger. Applications built as Smart Contracts are thus sometimes called "decentralised applications". A Smart Contract can be regarded as application logic that executes automatically during mining when the conditions of the contract are met [18]. From a programming perspective, a Smart Contract is a piece of code that is stored on a distributed ledger and executed in a decentralised manner, i. e. local execution, then synchronising and consenting on the resulting database change, if any, with the network. Ethereum's Smart Contracts are typically written in Solidity, a Turing-complete object-oriented high level programming language [2] with a syntax similar to JavaScript.

A notable feature of Solidity are function modifiers[16]. Prior to the execution of the actual function, function modifiers check if specified conditions are satisfied, e. g. if the party who calls the function call is really allowed to execute the contract's function. This way, function modifiers can act as a safety feature, e. g. for limiting access rights on specific functionality to a specified set of accounts, i. e. agents.

5 Technical View on Key Components

In the following, we will present our approach from a technical perspective. We first elaborate the model of a Linked Pedigree and its Ontology to model an item's creation and handovers among supply chain partners. Then, we outline the implemented Smart Contract's functionality that enables for the item verification process. Finally, we present our Linked Graph Traversal algorithm, thereby explaining the procedure of item verification and Linked Pedigree retrieval in detail.

5.1 Vocabulary

In each Linked Pedigree part that is not an initial Linked Pedigree part, the property `p:hasReceivedPedigree` specifies the respective previous Linked Pedigree part by its URI. When additional information is desired to be verifiable as well, additional triples can be added *ad libitum*. For verifying the information on the Linked Pedigree using the Distributed Ledger, we have to add information on where to verify the information. To this end, we built an ontology by taking selected parts from the OntoPedigree ontology, added terms from the EthOn ontology, and invented new terms. A depiction of our overall data modelling can be found in Figure 2.

[16] `https://solidity.readthedocs.io/en/v0.5.8/contracts.html#function-modifiers`

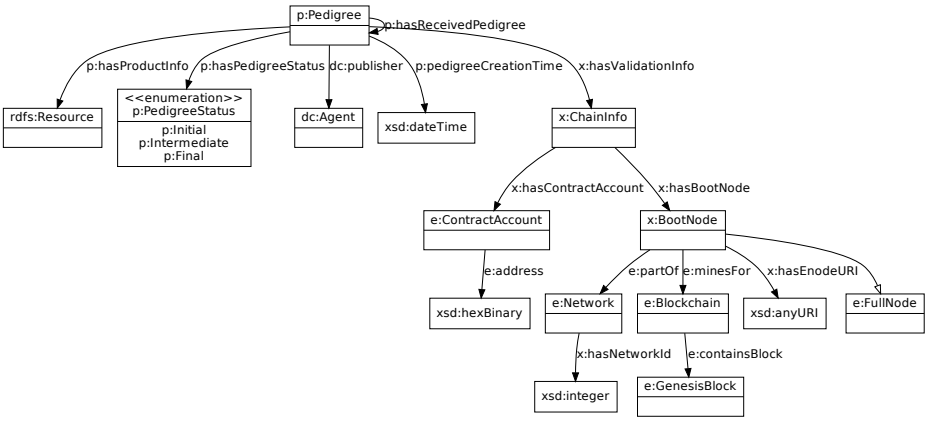

Fig. 2. The vocabulary we use for our approach. We use an UML class diagram to illustrate modelling in RDFS using the following correspondence: UML's class, association, and inheritance map to `rdfs:Class`, `rdfs:domain` and `rdfs:range` of an `rdf:Property`, and `rdfs:subClassOf` relationships respectively.

5.2 Smart Contract

Our Smart Contract offers three functions: First, RDF graph hashes of Linked Pedigree parts can be stored together with their URI on the distributed ledger. Further, these hashes can be looked up from the distributed ledger using their associated URI. Finally, the URI of a single Linked Pedigree part can be looked up using its direct successors' URI.

Storing hashes An agent, requesting the Smart Contract to store a hash of a Linked Pedigree part, must provide the following arguments:

- The hash itself
- The URI of the Linked Pedigree part (required to enable for look ups of the hash by its Linked Pedigree part URI).
- The URI of the previous Linked Pedigree part (needed in order to append the current part's URI to the correct Linked Pedigree)
- The wallet of the next owner (required for rights management, specifically we thus can restrict writing information on this Linked Pedigree to the next owner)

In Figure 1, the calls that "issue [a] transaction" are storing hashes. Once stored, the Smart Contract does not allow for hashes being altered or removed.

A request for storage of a hash results in a transaction on the distributed ledger issued by the Smart Contract. Therefore, the requesting agent has to pay a transaction fee in order to compensate for the required network utilisation.

Retrieving hashes To enable for a verification process by hash comparison, a hash can be retrieved for the RDF graph of a Linked Pedigree part by calling the Smart Contract using the part's URI. Such look-ups are characterised using "read call" in Figure 1. Since this look up can be carried out without a transaction to the network, no transaction fee applies here.

Retrieving URIs An agent may not be able or authorised to dereference the URI of a Linked Pedigree part. The Smart Contract offers a fall-back function for looking up the corresponding previous part's URI. This way, unavailable Linked Pedigree parts can be skipped, thereby keeping the traversable chain of URI references intact. Again, since any agent ought to be able to look up URIs, retrieval of URIs via the Smart Contract is unrestricted. We omitted such calls to the Smart Contract from Figure 1. As for retrieving a hash, the Smart Contract for URI retrieval does not need to invoke transaction, again, no transaction fee applies.

5.3 Link Traversal and Data Verification

To retrieve and verify a specific Linked Pedigree, an agent starts with the URI from the Linked Pedigree they know to be the last in the chain. They can then obtain the RDF graph that describes this Linked Pedigree part using an HTTP GET request. From this RDF graph, the agent calculates a hash value using the blabel approach from [6]. We use the implementation available online[17]. At the same time, the agent retrieves the stored hash for this URI from the Distributed Ledger using the Smart Contract. The agent can then verify the information by comparing the hash they generate to the hash retrieved from the Smart Contract.

To go further in the history of the item, the agent performs Link Traversal-based querying intertwined with verifying as just described: For a part p, the agent queries the RDF graph about the p for triples with p as subject and `p:hasReceivedPedigree` as predicate. Then, the agent finds the URI of the previous Linked Pedigree part in object position. With this URI, the agent performs dereferencing, verifying, and querying as described, until the initial Linked Pedigree part, i. e. the part with `p:Initial` status, is reached.

The traversal algorithm may terminate exceptionally, e. g. when Linked Pedigree parts are unavailable due to outages, or insufficient rights for the agent. However, for each Linked Pedigree part URI the previous Linked Pedigree part's URI can be looked up using the Smart Contract. This allows for skipping of unavailable parts.

By traversing backwards on this chain of URIs, the item's whole Linked Pedigree is retrieved, see the HTTP-GET requests from the End Consumer to Trucker and Fisher in Figure 1. If all hash pairs match, the whole Linked Pedigree can be regarded as verified. Additionally provided links can be looked up for more information, e. g. on the item itself, its production or its transportation, if corresponding access rights are granted.

[17] `https://blabel.github.io/`

6 Evaluation

To evaluate our approach, we contrast two ways of achieving the presented functionality using Smart Contracts: Looking at our supply chain example, we ask if the deployment of a Smart Contract for the whole supply chain network, or a more fine-grained approach of multiple Smart Contracts is more beneficial. For clarity of the presentation, we name the approach with a Smart Contract that manages hashes and URIs for *multiple* items in the network, a "Multi-Item-Contract" (MIC). The alternative, a "Single-Item-Contract" (SIC), is a Smart Contract that is used for validating one item exclusively.

For the evaluation, we first build a cost model to compare two approaches regarding operating cost and storage overhead. We then instantiate the cost model experimentally.

6.1 Cost Model

Let $\mathcal{C}$ denote a set of Smart Contracts that is deployed on the blockchain. Let $\mathcal{I} \subset \mathcal{U}$ denote the set of URIs that identify single item instances, for each of which a Linked Pedigree exists. Let $\mathcal{P} \subset \mathcal{U}$ denote the set of URIs that identify single Linked Pedigree parts.

We define a function $g : \mathcal{P} \to \mathcal{P}$ that maps a Linked Pedigree part $p_k \in \mathcal{P}$ to another $p_j \in \mathcal{P}$, where $k \neq j$. Thus, function g appends the Linked Pedigree part p_k to a previous Linked Pedigree p_j. This results in chains of Linked Pedigree parts that we formally describe as n-tuples. A single chain, i. e. n-tuple, forms an item's Linked Pedigree. Be Λ the set of all Linked Pedigrees is $\Lambda \subset \mathcal{P}^n$. An item i's Linked Pedigree $\lambda_i \in \Lambda$ is then an n-tuple of the form:

$$\lambda_i = (p_0,\ p_1,\ \dots\ ,\ p_n) \in \mathcal{P}^n$$

Each Linked Pedigree has an initial element $p_0 \in \mathcal{P}$, where

$$(g(p_j) = p_0,\ \exists p_j \in \mathcal{P})\ \wedge\ (g(p_0) = p_x,\ \neg\exists p_x \in \mathcal{P})$$

and consists further of n-1 elements $p_k \in \mathcal{P}$, where

$$g(p_k) = p_{k-1},\ \forall m \in \{1,\ 2,\ \dots\ ,\ n\}$$

Further, we define h as the bijective mapping between an item's URI and its Linked Pedigree $h : \Lambda \to \mathcal{I}$. Last, we define the funtion $e : \mathcal{I} \to \mathcal{C}$, which maps an item $i \in \mathcal{I}$ to a Smart Contract $c_j \in \mathcal{C}$, since each item is validated by a Smart Contract.

We thus defined three dimensions of our approach, which we use in the evaluation: the set of deployed Smart Contracts $\mathcal{C}$, the set of items $\mathcal{I}$ (Linked Pedigree equivalent), and the n-tuples of Linked Pedigree parts (each forming a Linked Pedigree):

$$\{\mathcal{C}, \mathcal{I}, \mathcal{P}^n\}.$$

6.2 Applying the Cost Model

Applying the model $\{\mathcal{C}, \mathcal{I}, \mathcal{P}^n\}$ to the question at hand, whether a MIC or a SIC approach is preferable, we make the following assumptions:

The number of deployed Smart Contracts $|\mathcal{C}|$ is variable. Further, the number of supply chain items is growing over time due to ongoing business:

$$\lim_{t \to \infty} |\mathcal{I}| \to \infty.$$

The same holds for the total number of Linked Pedigree parts, since

$$|\mathcal{P}^n| = dim(\mathcal{P}^n) \times |\mathcal{I}|,$$

each item having a Linked Pedigree with n elements. For our evaluation we will assume a constant average size of a Linked Pedigree $dim(\mathcal{P}^n) = n$.

In the following, we compare two approaches regarding operating cost and storage overhead: The first one is a MIC approach with a constant $|\mathcal{C}| = 1$. The alternative is a SIC approach with an over time growing $|\mathcal{C}| = |\mathcal{I}|$.

Operating Cost. When deploying the Smart Contract or issuing a transaction, the computing power lend from the network's miners needs to compensated by a transaction fee. Therefore, the usage of Smart Contracts is associated with operating cost.

To compare the operating cost of a MIC approach and a SIC approach, let d_a denote the average deployment cost for approach a. Let r_a denote an item's average registration cost of for approach a, which is simply the cost of storing the initial Linked Pedigree part. Let s_a denote the average cost of storing a hash of an intermediate Linked Pedigree part for approach a. Then for an approach a, the cost function

$$p_a(\mathcal{C}, \mathcal{I}) = d_a \times |\mathcal{C}| + (r_a + s_a \times n) \times |\mathcal{I}|, \ a \in \{MIC, SIC\}$$

applies. For the MIC approach, we have $|\mathcal{C}| = 1$ resulting in the MIC cost function

$$p_{MIC}(\mathcal{I}) = d_{MIC} + (r_{MIC} + s_{MIC} \times (n-1)) \times |\mathcal{I}|.$$

For the SIC approach, we have $|\mathcal{C}| = |\mathcal{I}|$ since there is a deployment of a Smart Contract per item. This results in the SIC cost function

$$p_{SIC}(\mathcal{I}) = (d_{SIC} + r_{SIC} + s_{SIC} \times (n-1)) \times |\mathcal{I}|.$$

By comparing the two cost functions, we can see that a growing $|\mathcal{I}|$ leads to higher operating cost for a SIC approach due to deployment cost typically being far greater than function execution cost. So, an increasing number of supply chain items $|\mathcal{I}|$ favours a MIC approach.

Further, equating both cost functions $p_{MIC}(\mathcal{I}) = p_{SIC}(\mathcal{I})$ leads regarding the number of supply chain items to

$$|\mathcal{I}|^*(n) = \frac{d_{MIC}}{d_{SIC} + (r_{SIC} - r_{MIC}) + (n-1)(s_{SIC} - s_{MIC})}$$

with $|\mathcal{I}|^*(n)$ being the number of supply chain items, where both approaches are at equal operating cost, dependent on the number of parts in a Linked Pedigree. $|\mathcal{I}|^*(n)$ shows that the more parts form a Linked Pedigree instance, i. e. the greater n, the less desirable is a MIC deployment due to the slightly lower storing cost of a SIC.

For our implementation, the following estimated[18] gas cost apply:

$$d_{MIC} = 1,065,000 \; ; \quad r_{MIC} = 95,000 \; ; \quad s_{MIC} = 185,000$$

$$d_{SIC} = \quad 750,000 \; ; \quad r_{SIC} = 80,000 \; ; \quad s_{MIC} = 170,000$$

These estimated gas costs lead to

$$|\mathcal{I}|^*(n) = \frac{1,065,000}{750,000 + 15,000 \times n}$$

Here, $n = 50$ is the vertical asymptote of $|\mathcal{I}|^*(n)$, meaning that for a Linked Pedigree length of less than 50 parts per single Linked Pedigree a MIC-based approach outperforms a SIC-based one.

Storage Overhead. When deploying a Smart Contract, the Smart Contract's code is stored on the distributed ledger. Every network participant with a full node[19] stores therefore a copy of that Smart Contract's code.

To formally compare the storage overhead of a MIC approach and a SIC approach, let s_a denote the storage space needed per Smart Contract for approach a. Let u denote the storage space needed per URI (item plus Linked Pedigree part), which is independent of the approach taken. Let further h denote the storage space needed per hash, which is also independent of the approach taken. Then for an approach a, the following storage overhead function applies for one network participant:

$$o_a(\mathcal{C}, \mathcal{I}, \mathcal{P}^n) = s_a \times |\mathcal{C}| + u \times (|\mathcal{I}| + |\mathcal{P}^n|) + h \times |\mathcal{P}^n|, \; a \in \{MIC, SIC\}.$$

When omitting approach independent variables, one network participant's storage space overhead function for deployed Smart Contracts remains

$$\hat{o}_a(\mathcal{C}) = s_a \times |\mathcal{C}|, \; a \in \{MIC, SIC\}.$$

For our (granted simple) implementation, the Smart Contracts lead to the following (in bytes):

$$s_{MIC} = 3,300 \; ; \quad s_{SIC} = 2,300$$

Obviously, for our implementation a MIC approach is superior to a SIC approach already for only two supply chain items.

[18] Estimates for URIs of 100 characters length.

[19] As opposed to a light node, which only stores a flat copy, i. e. hash values, of the distributed ledger.

7 Discussion

Applying the cost model on operating cost and storage overhead, we show that using network wide MIC is economically preferable as opposed to using a SIC if we consider networks with average supply chain length below 50. Above 50, the cost of an additional SIC, including deployment and Linked Pedigree storing, is smaller than the overall cost of storing an additional Linked Pedigree in a MIC.

There may be special use cases, where deploying multiple Smart Contract instances may be desirable in general, e. g. when the Smart Contracts are required to interact with each other or when functionality does not fit all participants' needs. With that, a SIC-based seems to be more flexible than a MIC-based. However, also in a MIC-based deployment, updates can be performed, yet they then need to appeal to the entire user base.

It is then in any case the obligation of the business partners to agree on which Smart Contract instance to use. Especially in large supply chains, this might cause significant overhead cost. Therefore, proposing a standard Smart Contract, that is already deployed and ready for usage, might facilitate business making in the network.

8 Summary and Conclusion

We presented an approach to verify the integrity of hyperlinked information using Linked Data and Smart Contracts, where Linked Data is used to store data off the chain. We showed a protocol for the verification in the presence of the transfer of physical goods, outlined the technical aspects of our approach and evaluated our approach using a cost model we developed. The implementation of our approach can be found online[20].

We see wide application possibilities of our approach in decentrally organised logistics networks with many participants of small size who desire on-premise data storage and acces control. As our presented approach allows for verifying information published as Linked Data, we contribute to the often neglected layer *trust* of the semantic web stack [13].

Acknowledgements

We acknowledge the support of Björn Leuthe and Christian Merker in an early draft of the work presented.

Bibliography

1. Arnot, C: Transparency Is No Longer Optional: How Food Companies Can Restore Trust, (2015). `https://www.forbes.com/sites/gmoanswers/2015/11/30/transparency-no-longer-optional/` (visited on 01/30/2019). Online

[20] `https://github.com/uvdsl/LinkedData-Logistics`

2. Buterin, V: Ethereum: A Next Generation Smart Contract & Decentralized Application Platform, https://github.com/ethereum/wiki/wiki/White-Paper. (2013)
3. Eberhardt, J and Tai, S: On or Off the Blockchain? Insights on Off-Chaining Computation and Data. In: Proceedings of the 6th European Conference on Service-Oriented and Cloud Computing (ESOCC) (2017)
4. Fortuna, G: Food safety: Midnight deal for revised General Food Law, (2019). https://www.euractiv.com/section/agriculture-food/news/food-safety-midnight-deal-for-revised-general-food-law (visited on 03/05/2019). Online
5. Gefen, D, Karahanna, E, and Straub, DW: Trust and TAM in Online Shopping: An Integrated Model. MIS Quarterly 27(1) (2003)
6. Hogan, A: Skolemising Blank Nodes while Preserving Isomorphism. In: Proceedings of the 24th International Conference on World Wide Web (WWW) (2015)
7. Huang, CC and Lin, SH: Sharing knowledge in a supply chain using the semantic web. Expert Systems with Applications 37(4) (2010)
8. Jharkharia, S and Shankar, R: IT-enablement of supply chains: understanding the barriers. Journal of Enterprise Information Management 18(1) (2005)
9. Kannengießer, N, Lins, S, Dehling, T, and Sunyaev, A: What Does Not Fit Can be Made to Fit! Trade-Offs in Distributed Ledger Technology Designs. In: Proceedings of the 52nd Hawaiian International Conference on System Sciences (HICSS) (2019)
10. Label Insight: *How Consumer Demand forTransparency is Shaping the Food Industry.* https://www.labelinsight.com/hubfs/Studies%20and%20Reports/2016-LI-Food-Revolution-Study.pdf (2016).
11. Lotfi, Z, Mukhtar, M, Sahran, S, and Zadeh, AT: Information Sharing in Supply Chain Management. Procedia Technology 11 (2013)
12. Nakamoto, S: Bitcoin: A Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf. (2008)
13. O'Hara, K, Alani, H, Kalfoglou, Y, and Shadbolt, N: Trust Strategies for the Semantic Web. In: Proceedings of the ISWC*04 Workshop on Trust, Security, and Reputation on the Semantic Web, Hiroshima, Japan, November 7, 2004 (2004)
14. PricewaterhouseCoopers AG Wirtschaftsprüfungsgesellschaft: *Bevölkerungsbefragung: Rückverfolgbarkeit als Kaufargument?* https://www.pwc.de/de/handel-und-konsumguter/assets/bevoelkerungsbefragung-rueckverfolgbarkeit-als-kaufargument.pdf (2016).
15. Solanki, M and Brewster, C: Consuming Linked data in Supply Chains: Enabling data visibility via Linked Pedigrees. In: Proceedings of the 4th International Workshop on Consuming Linked (COLD) at the 12th International Semantic Web Conference (ISWC) (2013)
16. Sopek, M, Gradzki, P, Kosowski, W, Kuzinski, D, Trójczak, R, and Trypuz, R: GraphChain: A Distributed Database with Explicit Semantics and Chained RDF Graphs. In: Proceedings of the 3rd Workshop on Linked Data & Distributed Ledgers (LD-DL) at the Web Conference (29th WWW) (2018)
17. Steinfield, C: Inter-Organizational Information Systems. In: Computing Handbook, Third Edition, Volume 2: Information Systems and Information Technology. Ed. by H Topi and A Tucker. Chapman and Hall/CRC, New York(2014)
18. Szabo, N: Smart Contracts, (1994). http://szabo.best.vwh.net/smart.contracts.html. Offline, but available in the Web Archive