SOFIE - Secure Open Federation for Internet Everywhere

779984

DELIVERABLE D2.3

Federation Framework, 1st version

Project title	SOFIE – Secure Open Federation for Internet Everywhere
Contract Number	H2020-IOT-2017-3 – 779984
Duration	1.1.2018 – 31.12.2020
Date of preparation	31.10.2018
Author(s)	Santeri Paavolainen (AALTO), Mikael Jaatinen (LMF), Vasilios Siris (AUEB-RC), Spyros Voulgaris (AUEB-RC), Nikolas Fotiou (AUEB-RC), Tommi Elo (AALTO), George Xylomenos (AUEB-RC), Yannis Oikonomidis (Synelixis), Sotiris Karachontzitis (Synelixis), Pekka nikander (AALTO)
Responsible person	Santeri Paavolainen (AALTO), santeri.paavolainen@aalto.fi
Target Dissemination Level	Public
Status of the Document	Completed
Version	1.0
Project web-site	https://www.sofie-iot.eu/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779984.

Table of Contents

1. Introduction

This document is work-in-progress and will be updated during the SOFIE project.

1.1 Overview

This document specifies interfaces and protocols that implement the SOFIE Architecture requirements, and “business platform” pattern in general, and several specific business platform models used in the SOFIE project. Please note that the SOFIE Architecture sets out generic requirements and does not itself specify any particular set of technologies to be used. This document also tries to follow the architectural requirements on backward compatibility, so that future versions of the framework specification are as much backward compatible as feasible.

This document does not prescribe any particular programming language, nor any runtime or deployment environment. There does exist, however, a SOFIE Reference Platform that implements all the required components specified in this document, and does this within the context of a specific language and deployment environment.

“Business platform” is not a technical term — instead it refers to a platform where business takes place. Business platforms[1] can vary from the everyday and mundane, such as a grocery store (which features a physical space open at specified times of day, open access with the proprietor reserving the right to expulse badly behaving individuals, with the exit criteria of having paid for the selected merchandise), to elaborate, such as multiparty auctions (think of flower auctions in the Netherlands), or market matchmaking platforms (Apple Store, Google Play).

Within the SOFIE project, several *generic* business platform *models* have been identified. These are not specific business platforms, but generic implementations that can be configured for different situations, and that would often require custom business logic to be included.

1.2 Structure

A detailed list of protocols, interfaces and approaches for the SOFIE framework is outlined in Section 2, below. See the SOFIE Architecture document for the general requirements that apply to the framework and its implementations. Section 3 discusses integration patterns for business platforms and business platform templates. Specific business platform templates are described in numbered sections thereafter. Section 4 describes an offer marketplace, Section 5 provenance tracking for trackable assets, and Section 6 IoT resource access.

[1] Not to be mixed with “platform business”, which is a specific business strategy, which may or may not result in the existence of a business platform. Not all business strategies are business platforms.

2. Components

The subsections below lists different use cases and protocols that MUST be used, when relevant, for the implementation of SOFIE components and software. If there is no suitable use case for a specific situation, the implementation may choose any protocol, but with the caveat that in a future SOFIE Framework specification version that particular area may end up to be covered by the specification.

2.1 Protocols

2.1.1 General

The "general" protocol requirements apply to most situations, except for cases specifically listed in further sections.

Table 1: Situations and protocols.

Situation	Protocol	Notes
General remote resource access protocol	HTTPS	
Resource access pattern	REST	"REST" is not a protocol per se, but an approach for defining resource access patterns.
Data interchange format	JSON	
Data linkage	Linked Data, JSON-LD[2]	When feasible, data linkage information should be provided
REST API descriptions	OpenAPI 3.0[3]	
IoT device integrations	W3C Web of Things[4]	This includes linkage to IoT devices, including gateways, description of interfaces and actions supported by IoT devices etc.
Delegated application authorization	OAuth2	
Delegated party-to-party authorization	User-Managed Access 2 (UMA2)	
Federated authentication	OpenID Connect (OIDC)	
Data governance information	?	

[2] https://json-ld.org/ and https://www.w3.org/2018/json-ld-wg/
[3] https://www.openapis.org/
[4] https://www.w3.org/WoT/

Data provenance information	?	
Data ontology categorization	?	
Payments	W3C Web Payments[5]	For generic situations, methods applicable to specific situations (on-ledger payments, for example) are not covered by this

2.1.2 IoT devices

Specifically, protocols which are suitable for resource-constrained devices. In general, one should use general protocols such as HTTPS as they allow maximal interoperability. However, in some situations a resource-constrained device may not be able to operate a full HTTP/TLS stack, being restricted to a lower-level protocol. Also note that if a resource-constrained device does not operate an Internet Protocol stack at all, and is connected through a non-IP protocol to a gateway, the specification really applies to the gateway, and not the device itself.

Table 2: *Situations and protocols for IoT devices.*

Situation	Protocol	Notes
Overall	W3C Web of Things	The W3C WoT patterns in general must be followed.
Communication protocol (constrained)	CoAP or MQTT	
Session security	DTLS (or TLS)	Pre-shared and other keying (detailed later)

2.1.3 Distributed ledgers and blockchains

At this stage in blockchain development, it is impossible to mandate the use of a particular blockchain, although the use of Ethereum, Hyperledger Fabric and Guardtime KSI Blockchain can be suggested. Thus, the protocol descriptions below cover only some very general cases, or a specific mature protocol on a particular ledger technology. In some situations we specify multiple protocols or implementations in which case **any** of them is suitable (although this may lead to a situation where general interoperability would require clients to implement multiple protocols, in the case of blockchains this is not a general problem, since specific services are usually closely coupled to a particular set).

Table 3: *Situations and protocols for blockchain-related protocols. Note that many of these are specific to a particular blockchain implementation or network.*

Situation	Protocol	Notes
Interface discovery	ERC-165 (Ethereum)	
Payments	Native format (blockchains with	

[5] https://www.w3.org/blog/wpwg/

	cryptocurrencies) Interledger Protocol[6]	
Distributed identity	Sovrin[7] uPort[8] Veres One[9]	
Interledger operations	Hashed time-locked contracts	

6 https://interledger.org/
7 https://sovrin.org/
8 https://www.uport.me/
9 https://veres.one/

3. Business platforms

3.1 Overview

This section contains an overview of what business platforms (BPs) are listed and how their descriptions are structured. The individual BPs are described in the following sections. Note that in some cases the description can be very brief, in which case a link to more comprehensive document is provided (for example, in a case that the business platform has an implementation template residing in a public repository).

The following business platforms, implementations (within the SOFIE project), and their current status are listed below:

#	Business Platform	Status	Description
1	Offer marketplace	Under development	Generic model of request-offer (or proposal-bid) batch transaction model
2	Provenance chain for trackable assets	Under design	Establishment and recording into ledger provenance information for trackable assets
3	IoT resource access	Under design	Access mediation to generic IoT resources

3.2 Integration

The term “business platform” refers to a comprehensive whole upon which a business typically operates. Such a platform typically comprises of multiple different components. A business may operate on a platform that consists of a batch-auction marketplace, sourcing management, billing and invoicing systems, etc. Another business may use some of these components, but arranged differently for their business platform. However, for our purposes, a “business platform” is something that can (at least theoretically) be used singularly to implement an actual business platform (assuming that owning and operating that particular platform would be the sole purpose of the business platform). This is a question of context and scope.

It is assumed that the business platforms described below will form only a single component of a comprehensive and deployed business platform.

4. Offer marketplace

4.1 Overview

The offer marketplace is a service that operates a batched, limited-time request-offer marketplace. The marketplace has the following defining characteristics:

- **Request-driven**: There is a request for offers, with the request defining different conditions that the offers must conform to.
- **Time-limited**: Each offer to a request must be submitted within a limited time window.
- **Batched**: The requestor picks out winning offers only at the end of the round (end of time window).
- **Non-repudiability**: The marketplace offers non-repudiation guarantees for both requests and offers. This can be fully or partially implemented using a distributed ledger.

This offer marketplace can have different configurable or customizable aspects that give each specific offer marketplace their specific form, such as:

- Can offers be made by anyone, is some form of identification required (and what), or is there some form of pre-approval process for parties making offers?
- Can a single entity make multiple offers (exclusive to each other), and can an offer be revoked within the time window?
- Are the details of requests public, or available only to identified or pre-approved parties?
- Are the details of offers public, or hidden to other parties than the requestor?
- Are the details of offers available to the requestor until the end of the round? (Single or double envelope?)
- Are the details of the selection open to everybody, or pseudonymous, or can selected offers be identified by the offer-maker without divulging information to non-selected parties?
- Are requests and offers signed? (This gets into the whole question of signed certificates, or public keys, or public key hashes etc. etc.)
- Can requests be cancelled, or are they required to run to completion?
- Additional details on the interaction between non-DLT and DLT components, while part of the implementation, also affect the transparency and trustability of the system, such as:
  - Are the details of the request stored in the DLT, or only the minimal (non-repudiable) reference to it in an external system?
  - Are offers made directly to the DLT (with full information), is the offer in the DLT a non-repudiable reference to one retained by the offer-maker, or are the offer details first stored in an external service with non-repudiable proof of receipt?
  - Are requests and offers stored in the DLT at all, in any form, even if finalization results are stored there? If so, this would require signing and countersigning of requests and offers to guarantee non-repudiation.
  - If offer details are held by offer-makers, is there some form of external commitment required (e.g. escrow) to ensure the details are published in time?
  - Is the offer selection made on the DLT (by use of a smart contract, for example), or only partially, or fully outside of the DLT?

Furthermore the actual request and offer formats, apart from generic aspects, are dependent on the actual business performed on the marketplace. Note also that not all implementations following the general offer marketplace pattern necessarily support all of the options listed above. Thus, while the generic aspect of the offer marketplace BP defines the minimum required to operate any kind of marketplace following the specifications listed above, a specific instantiation of the marketplace can include further and more refined information on the request and offers such as logical or geographic constraints, price tiering etc. (See also the section on protocols below for more information on how the protocols vary depending on the marketplace type.)

4.2 Process flows

The way an offer marketplace works is a sequence of steps, some of them embedded within others. The primary ones are:

1. Marketplace lifecycle management, e.g. commissioning and decommissioning -- creation of the marketplace smart contract, and its (potential) destruction. (While a single non-DLT service component may operate on multiple contracts, for trustability reasons a single “marketplace” should be defined only as the lifetime of its smart contract.)
2. Information queries, such as retrieving metadata of the marketplace itself, retrieving information on requests, on offers (if allowed), and request’s final status (selection).
3. Request lifecycle management, e.g. creation and finalisation (selection process).
4. Making offers.

4.2.1 Marketplace lifecycle management

At least on Ethereum, smart contracts cannot be modified after their creation. While it is possible to use contract proxies, their use can make the system less trustworthy as any contract validation has limited usefulness[10]. The overall lifecycle management events are described in the figure below.

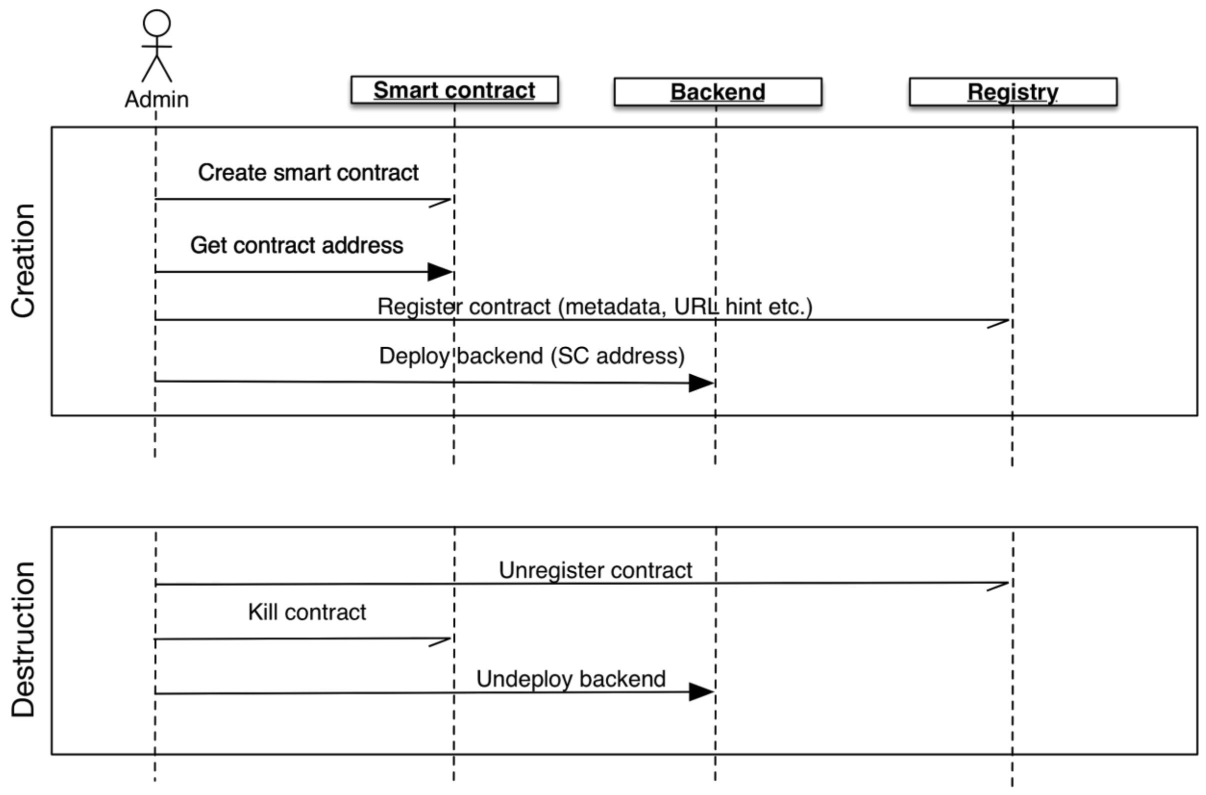


Figure 1: Offer marketplace lifecycle events.

[10] A contract proxy is one that delegates calls to it to another contract. The target contract is defined as a variable, and can be changed during the contract proxy’s lifetime. Thus, auditing the contract proxy and *a* target contract does not give any assurances if and when the target contract is changed later.

Since deployment of the backend requires knowledge of the smart contract address, conceptually this has to occur later in the process (although, since the backend is mutable, this does not really matter in practice). On the other hand, some information required for the smart contract (backend URL discovery address etc.) need to be stored, but not necessarily immutably in either the contract and/or in well-known Ethereum registry services (global registrar, hash registrar and URL hint registrar) after the contract creation.

During decommissioning, the reverse needs to happen — services unregistered, the contract killed and the deployment removed. (Although, the contract is not required to have a kill switch, so it might be permanent. In this case the contract would be unregistered and left otherwise unused.)

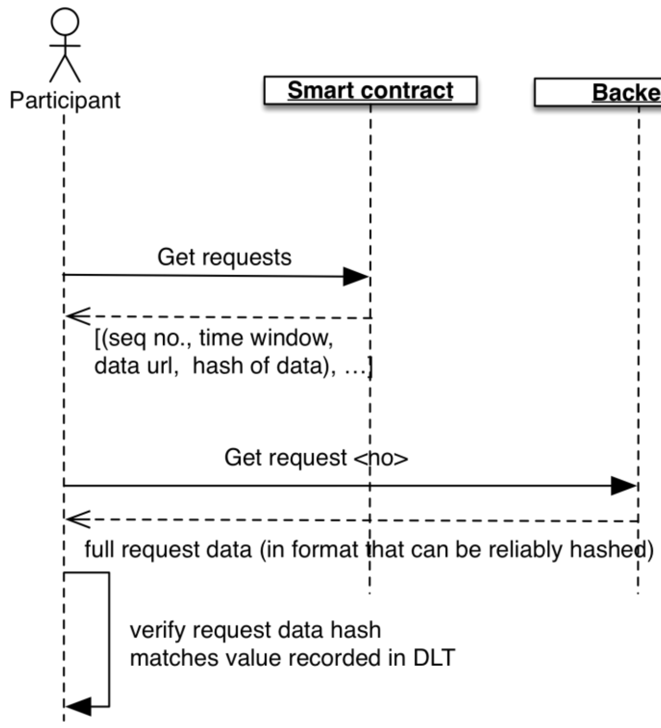
The commissioning process may include other steps such as smart contract auditing, publication of the smart contract address etc. and other assurance steps that occur mostly on the organizational or communication level between the marketplace owner and (potential) marketplace participants.

4.2.2 Information queries

Information queries can be sent to either the smart contract, or the backend. Some of these may require that the caller is authenticated or identified. Any discovery, authentication or identification needs to follow the SOFIE Framework specification. In general, the query interfaces can be categorized as follows:

1. Discovery and description interfaces. These can offer different information based on the authentication/identification of the requester (e.g. public vs. private interfaces).
    a. The smart contract address must be discoverable through the metadata of the backend and vice versa.
    b. The fact that this is an implementation of this particular marketplace needs to be identifiable in the metadata.
2. Metadata about the marketplace operation. This would identify externally visible configuration parameters and quantification of the trust model (e.g. is selection done in smart contract, or in the backend; are multiple offers allowed; can offers be revoked; are requests and offers presented via handles or directly, and so on — see above on the configurable aspects).
3. Retrieve status of current and past requests.
4. Retrieve information about open offers. This information may not be available at all, or be provided only to authorized entities, and may be limited to include only information on an entity's offers.

In some cases the entity retrieving the information may want to confirm non-repudiability aspects by retrieving information initially from the DLT, and using this information to retrieve a copy or full version of the referred information from the backend as shown in the figure below.



Participant
Smart contract
Backend
Get requests
[(seq no., time window, data url, hash of data), …]
Get request <no>
full request data (in format that can be reliably hashed)
verify request data hash matches value recorded in DLT

Figure 2: Fetch and validate full request data.

The reverse also applies — if the market participant first retrieves data from the backend, they can verify that it has been publicly announced in the DLT by fetching data from the smart contract and cross-validating it. Also note that if the request data is small in size, it may be stored fully in the smart contract. In that case the participant retrieves all information directly from the DLT (although the reverse does not apply — they would need to cross-validate data from the DLT if they first retrieved it from the backend).

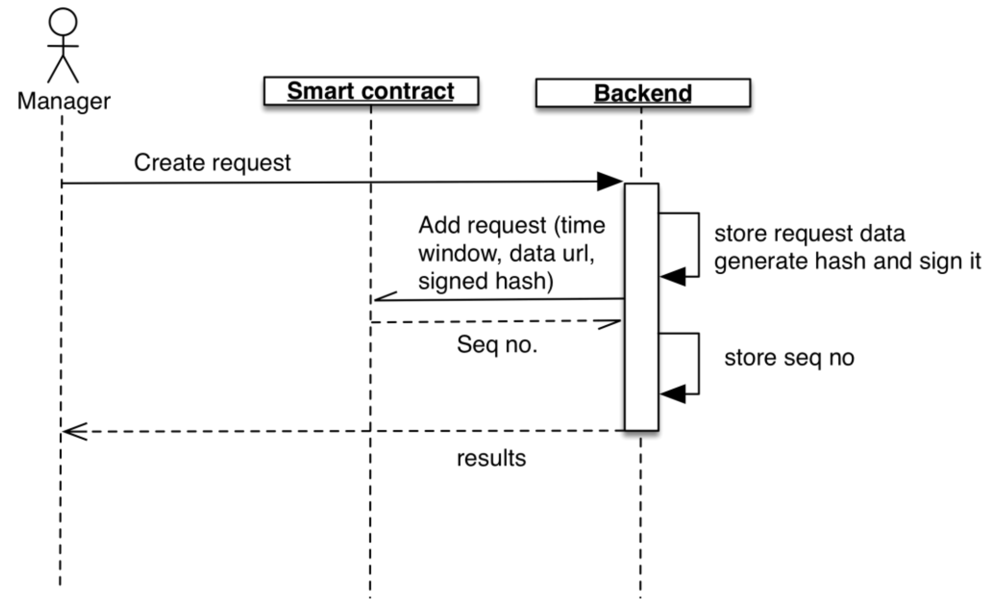
4.2.3 Request lifecycle management

The creation of requests, and finalizing a request at the end of its time window are among the most complicated operations on the marketplace. Naturally, only a person with sufficient authorization (e.g. access to the address that the smart contract accepts as a valid operator) can initiate request lifecycle operations.

Since we assume that the non-DLT backend always has a copy (not necessarily non-repudiable) of the request information, it is kept in the loop even in situations where all of the request information is stored in the DLT. Also, the interactions described below assume the most heavy-duty security approach is taken (with signatures etc.) — some of these may be omitted when operating fully on a DLT, for example. Similarly we assume the need to have the backend integrated maximally such as having the selection performed at the backend.

The creation of a request starts with the manager making a request to the backend to create a new request. The manager supplies any necessary information for the request (time window, other parameters). Since we assume the manager is within the trust boundary of the offer marketplace, they can safely use the backend directly (and assume it works correctly)[11].

[11] While this diagram describes the manager interacting directly with the backend, in reality the “manager” here is really another service offering the user interface to the manager (potentially a page application, or a whole other management system tied to the company enterprise systems.)

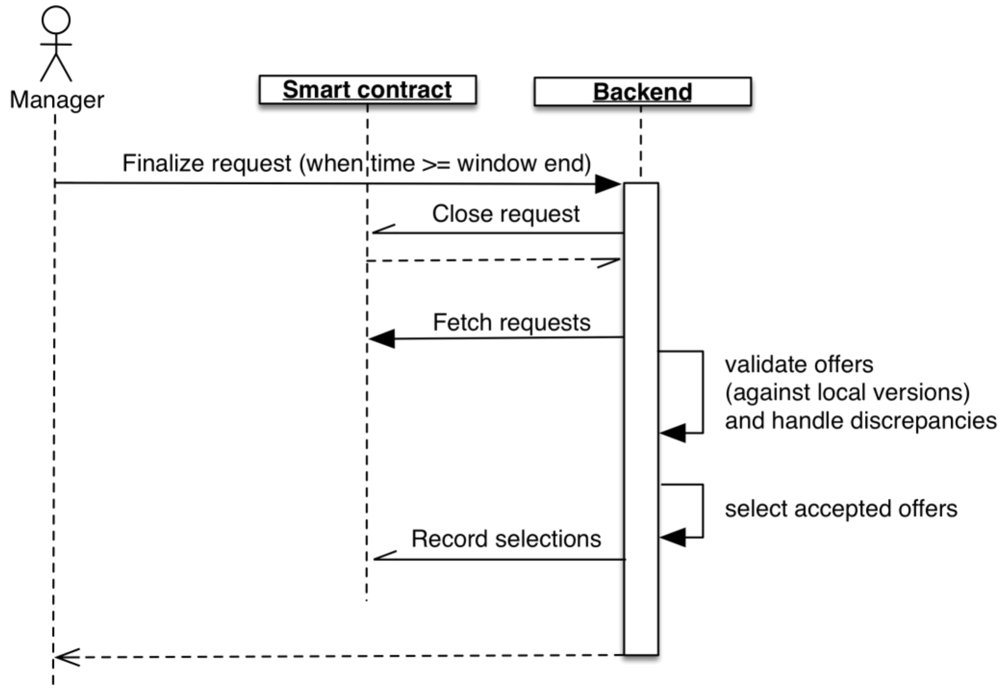


Manager
Smart contract
Backend
Create request
Add request (time window, data url, signed hash)
store request data generate hash and sign it
Seq no.
store seq no
results

Figure 3: Request creation flow from manager to backend to smart contract.

During the time window that the request is available it is possible for the manager to use the information querying interfaces to monitor the state of offers made so far. If requests can be cancelled, then a similar sequence of actions occurs (if the smart contract is configured to allow cancellation).

The next step is the finalization of the request, e.g. the selection of a winning offer or offers. The process of selection is highly specific to the marketplace, and may require steps taken to unblind the offers. Here we omit steps required for blinding and unblinding of offers (see below for separate discussion on that), and instead focus on the case where the offer selection is performed by the backend. In the figure below, the manager is making the request to finalize the request, although in reality this would be more likely to occur automatically via a scheduling system (on behalf of the manager, essentially).



Manager
Smart contract
Backend
Finalize request (when time >= window end)
Close request
Fetch requests
validate offers (against local versions) and handle discrepancies
select accepted offers
Record selections

Figure 4: Finalization of a request, validating state of offers and making a selection.

Here an important consideration is that in the case of Ethereum, the smart contract cannot reliably enforce the time window nor that all participating parties would have seen all transactions that may have been submitted, thus requiring some form of barrier synchronization between the two (it can be achieved also by waiting for blocks with later timestamp). If the selection was done entirely in the smart contract, then instead of "record selections" an operation such as "make and record selections" would be invoked.

4.2.4 Making offers

Similar to initiating a request, the creation of an offer can be a multi-step process where the offer-maker first sends out the offer to the backend (in case the offer data is too large to be stored on the DLT, or contains information that is not to be disclosed publicly). This may also require the request to be authenticated or otherwise identified.

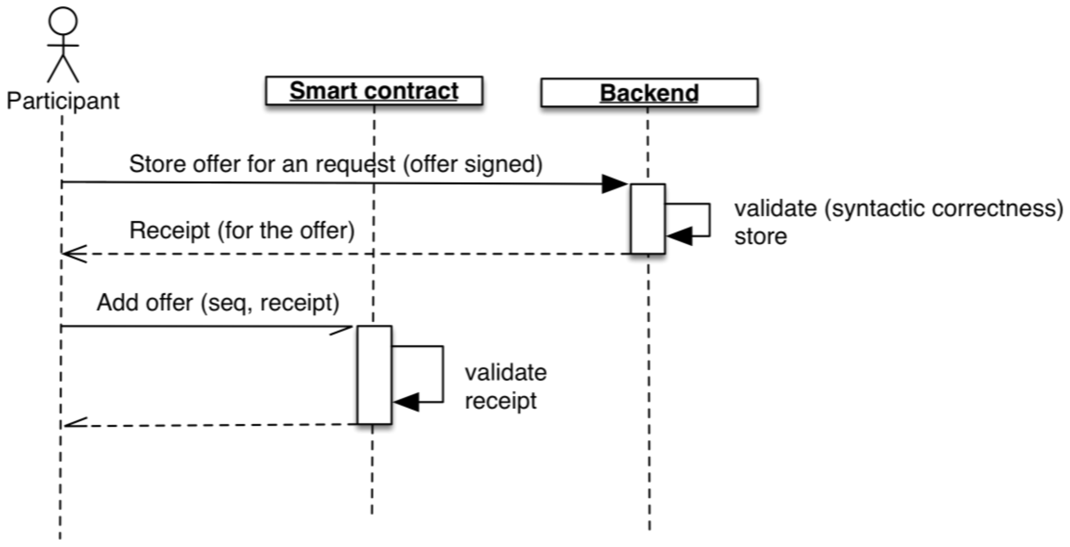


Figure 5: Making an offer against a request.

The details of the cryptographic protocols need to be considered of course further, the above is only a rough outline of the operations during an offer. The goal is to ensure that all steps are non-repudiable, e.g. if an offer has been made, the participant can show that it was successfully received (receipt) and likewise the marketplace can confirm the data that was correctly submitted and associated with a specific offer on the DLT.

While not necessarily recommended, if the offer needs to be kept confidential from the DLT users in general, but is small enough that it is economical to be sent directly to the smart contract, it would be possible for the offer-maker to digitally encrypt the offer with the marketplace's public key. Since the corresponding decryption key cannot be stored in the smart contract this would necessitate the use of the backend for request finalization.

4.2.5 Offer blinding

Some markets may require either single or double blinding[12]:

- In single blinding, all offers are kept hidden from the marketplace until the round is closed, and then all offers are "opened" (decrypted or revealed).

[12] Please note that the concern of offers being public on the DLT is a separate issue — offers can be either stored in the backend with only a reference (receipt) being part of the actual offer on DLT, or sometimes the offer itself can be encrypted with the marketplace's public key. In either case the offer is still visible to the marketplace itself (at backend).

- In double blinding, all offers are kept hidden, but also the identities of the selected offers only become known after they have been selected, e.g. during the selection process the identity is not available to the entity making the selection.

In all situations there is a need for time-locked encryption. In practice this occurs via some mechanism where the offer makers will either give out the decryption key (if offer is encrypted), or the original offer (if offer is in the form of a hash). In all of these cases it is important to consider the case of a *no-show* participant — there should be either an enforcement mechanism or other disincentive for participants from failing to unblind their offer at request finalization time. The protocol essentially operates within the request finalization process, where the request is first closed and then the marketplace waits for participants making offers to provide either the decryption key or the hash preimage. The situation of not providing a valid decryption key, a valid preimage or withholding them must be considered.

In double-blinded offers the identity of the offer-maker is not revealed until the offers have been unblinded and evaluated. Since this is not likely to be required (at least in the short term), the details of this process and its (various) associated challenges are not discussed further here.

4.2.6 Escrow

If marketplace participants are pre-approved (with a potentially externally enforceable contract in place) then things usually can just proceed without further assurances. However, if there is less enforcement capability available then some other mechanism to penalize badly behaving marketplace players needs to be in place. In these cases an escrow payment might be a suitable approach — offers require an escrow of cryptocurrency to be placed, which is automatically released for non-selected offers, and for selected offers kept in time-lock escrow (so that the request maker can open a dispute).

4.3 Protocols

There are some specific interactions that contain at least a sets of information that can listed, and also those parts that are extensible and may contain marketplace-specific information. These are outlined below (different implementations may choose different encoding and naming conventions, so these are conceptual).

Note! These are very much work in progress and will co-evolve with framework and reference platform implementations.

4.3.1 Service description

At backend:

- Service type: sofie-bp-offer-marketplace-1, with subtype identifier for the particular marketplace specification (+ versioning!)
- Smart contract address: DLT type, and address, e.g. ethereum and SC address
- Round length: fixed, dynamic or variable (dynamic = anything goes, variable = definite minimum and maximum, but unknown statically)
- Pre-approval required: yes / no
- Pre-approval information URL: location of a web page, or API endpoint for pre-approval registration
- … (TBD)
- … API endpoint descriptions ...

At smart contract:

- Service type: sofie-bp-offer-marketplace-1, with subtype identifier for the particular marketplace specification

- Backend: URL of the backend (for API discovery)
- Round length: …

(essentially all static parameters should be 1:1 between backend and SC description fields)

4.3.2 Marketplace request

The marketplace request format may be either fully contained, or partial, where partial refers to the actual contract at an external URL. Some of the fields are required to be present even in the partial request format; the list below marks fields that required with an asterisk (*):

- Identifier*: Unique identifier for the request (an integer counter is a possibility)
- Information URL*: URL that will provide the request, including the full version if only a partial request is provided at SC
- Full request signature / hash*: Signature or hash of the full request, sans hashed fields (see specification elsewhere XXX)
- This request signature / hash*: Like previous, but only applies to this request — by necessity if this request is full, this and previous are identical, otherwise for partial this signs only the partial request
- Round open interval: time period during which bids are accepted
- Decision interval: the time interval when a decision is to be made (earliest to latest)
- (TBD: can marketplaces mix open / closed offers — if yes, then need to specify how offers are made here too)

Note that the request can contain any other fields that are specific to the marketplace. Some possible scenarios are, for some theoretical examples:

- Selling (or buying) flowers: Type of the flower, and quantity, potentially a minimum or maximum price required for offers
- Car rental (request for cars): Geographical area of pick-up and drop-off, type of car, rental period, potential extra requirements (child seat, hitch, …)

4.3.3 Marketplace offer

Similar to the marketplace request, the marketplace offer needs specific fields to be present even in partial offer (in DLT), with the full request having to be at minimum in the backend.

- Request identifier*: The request identifier for which this is an offer
- Offer identifier*: per-participant identifier, so that <request id, participant, offer id> are unique
- Participant*: Participant identity identifier — depending on the requirements on the marketplace, this may be an DLT public key hash, pre-approval identifier etc.

Rest of the information present on an offer are dependent on the particular marketplace — considering the examples from requests we could have:

- Selling flowers: Price
- Car rental: Exact pick-up and drop-off addresses (geolocations), detailed information on the car on offer, price for the rental

4.3.4 Decision result

The decision needs to be communicated. At minimum this needs to contain the following information:

- Request identifier*: The request that this decision is for
- Decision time*: When the decision was made, this may include also the DLT state (block number etc.) that was the last one that included valid offers

- Selected offer(s): Participant and offer identifier of the offer (this may be available only for the selected offer or offers) — note that a selection can cover multiple offers

The specific marketplace may also include other information, with for example considering again the cases above (this information might be only available to authenticated participant with offers that were chosen):

- Selling flowers: Token for lot pickup, pickup details
- Car rental: Identity of the person picking up and dropping off the car, proof of rental payment in escrow

5. Provenance chain for trackable assets

5.1 Overview

The problem this BP is modeling is that there exist assets that are tracked, which may be repackaged or refined during their transport from the original raw material producer (farm, for example), and whose provenance needs to be tracked for the purposes of 1) providing consumers with guarantees on the quality and proper handling of any goods they purchase, 2) enabling more efficient operations for companies along the logistics chain through increased transparency, integrity and availability of the digitalised services.

While the description below does not specifically require any particular mechanism of implementation, for descriptive purposes we'll show a logistics chain that is formed around a consortium blockchain where different entities in the system in turn use blockchains for their claim generation.

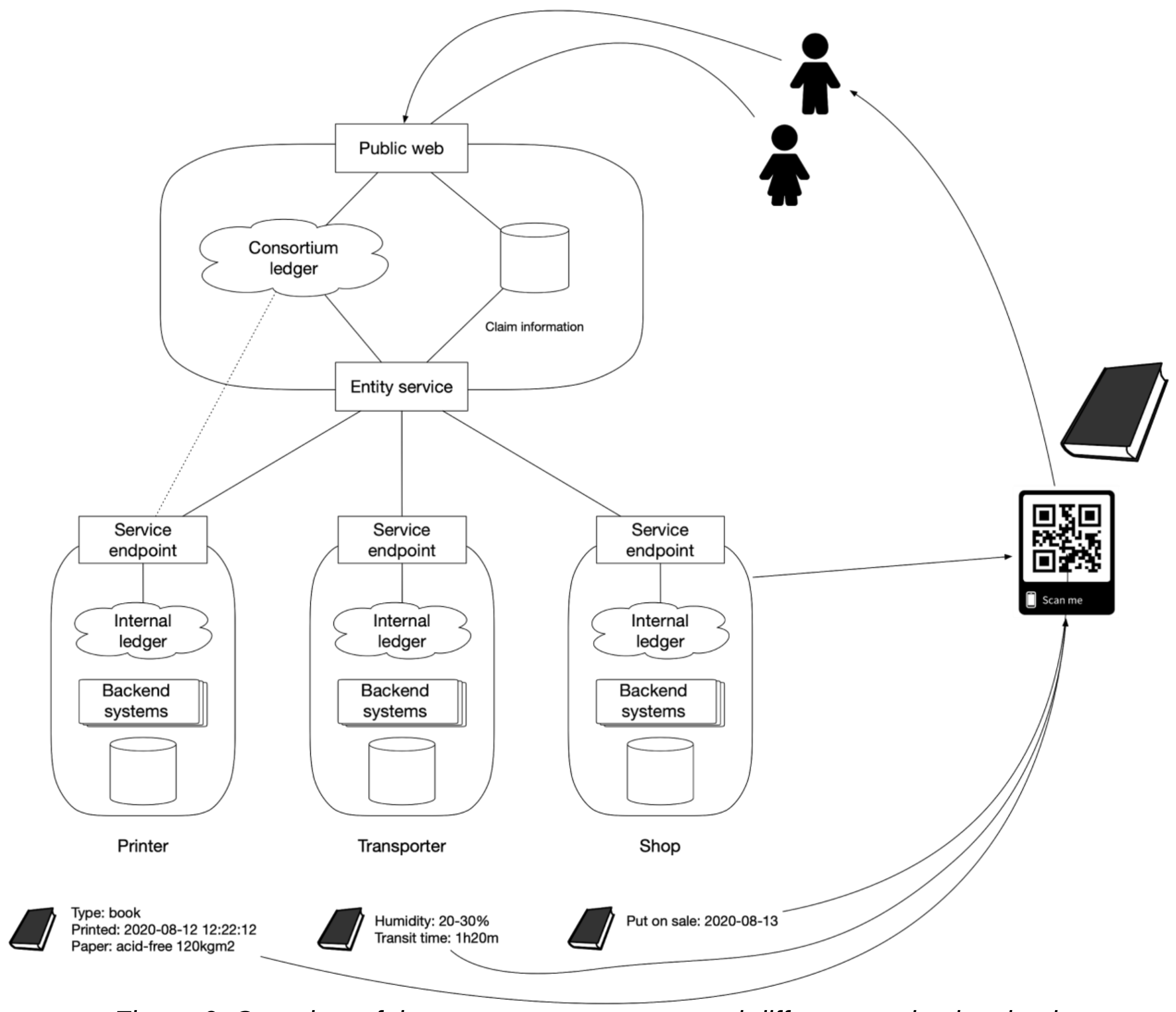


Figure 6: Overview of the system components and different parties involved.

This example shows three different companies handling books from printing to the consumer. The printer would store information about the book in its systems, then incorporate this information as claims on the asset (book shipment) when it is defined and transferred to the transporter. The transporter updates information and provides claims on its state during the transport, and finally, the store can include additional information relevant to the provenance of the product for consumers. Finally, the consumer can scan a QR code at the shop and query detailed information about the product's path to the store. The system and all interactions in it is designed to guarantee provenance of the information that is provided, e.g. that it has not been tampered with or post-hoc edited.

5.2 Terminology

The system has the goal of **tracking the provenance of assets** that are being transported within the system. **Assets** are the units that are handed off between different entities in the system, which may be transformed into new assets while in the system. Thus, an asset is a mutable entity. The entities in the system which handle assets are **nodes** in the asset flow. Assets are transported and stored in **containers**. An asset may change its container during its lifetime many times. Containers may also be re-used, while assets are unique. When an entity is in possession of a container, they may define **claims** regarding the state of the container and/or the asset. Claims are tagged by their **purpose** — a purpose may for example be that the information associated with the claim is meant for consumers, or that it describes the asset's general properties etc. Depending on the purpose of the claim, its visibility may change during the lifecycle of the asset. Claims are made about specific **information** being present at the claimant, thus claims themselves are not the information, but statements of fact, such that the claimant can provide the information associated with the claim if necessary. Overall the combination of the claims and the information associated with them create the **provenance chain** for the asset.

Sometimes an asset may be also a container: packaged goods can have RFID tags. If a packaged good was transported in a larger container, then for transport purposes it would reside in the outermost container. However, if it was individually repackaged, then it could be tracked by its own RFID tag (an asset would be its own container). In most cases, logistic partners and stores would keep track of each package separately, but for transiting purposes this level of detail might not end up in the system (e.g. only a larger container unit would be used).

The traceability of provenance stems from the requirement that any transformation of an asset (repackaging, refining) has to link to another asset (in control of the transforming entity). This linkage provide a chain of provenance that goes back to the original producer of the underlying goods (raw material production).

5.3 Assets and entities

5.3.1 Asset lifecycle

Assets are created either through registration (at a source), or through transformation (refining, repackaging). The default state for an asset is *in transit*, meaning that it is ready to flow or flowing through the system. An in transit asset may be either released, in which case it is considered to be "out of the system". A typical case for this could be spoilage, causing the asset to be entirely thrown out (not just reduced in amount). Another possibility is for an asset to be finalized, which means it has reached a sink. At this step the asset provenance information is made available to consumers. Finally, in transit assets may be transformed. The transformation process may take some time — a large shipment may be gradually repackaged into smaller shipments, until it has been "consumed". When an asset is considered "exhausted" (no physical division or further sale of the asset is possible, for example, the asset unit of 10 kg of grapes have all been either sold, or tossed away as spoilage), then it is "released". This means the asset is "out of the system" from the viewpoint of control and transport, and no further changes are possible to occur for the asset *within the system* (it can of course be used by the consumer in any way they wish). Note that transformation and finalization are sticky attributes, thus release after transformation and finalization reflect different states than direct release from transit. (Finalized, but released assets might be queryable by a consumer even when the asset has been out of sale for weeks — the customer may be checking the provenance of something they bought earlier.)

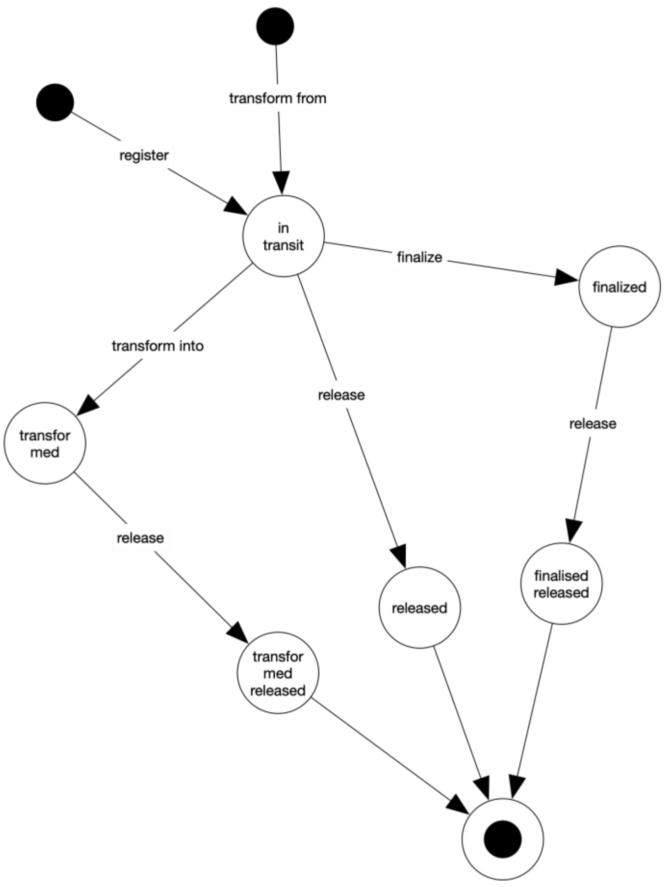


Figure 7: Lifecycle of assets.

The final step to a terminal state is a bit questionable in this kind of system, and might be something that occurs "never." ("Never" would in practice mean the extent of any potential legal liability, so while definitely finite, it could be still tens of years, a practical eternity from computing point of view.)

Provenance lifecycle is tied to the asset lifecycle, and containers are separately managed, and exist only transiently within the system (while an asset is assigned to them).

5.3.2 Nodes and the flow of assets and containers

Assets are created, transited, transformed and finally released from the system to consumer use. These assets may be transported via multiple different entities and end up in different stores, even when originating from the same producer. The diagram below shows a simple asset flow network, where the farmers are sources of assets that then are transported (flow) via the network to stores (sinks).

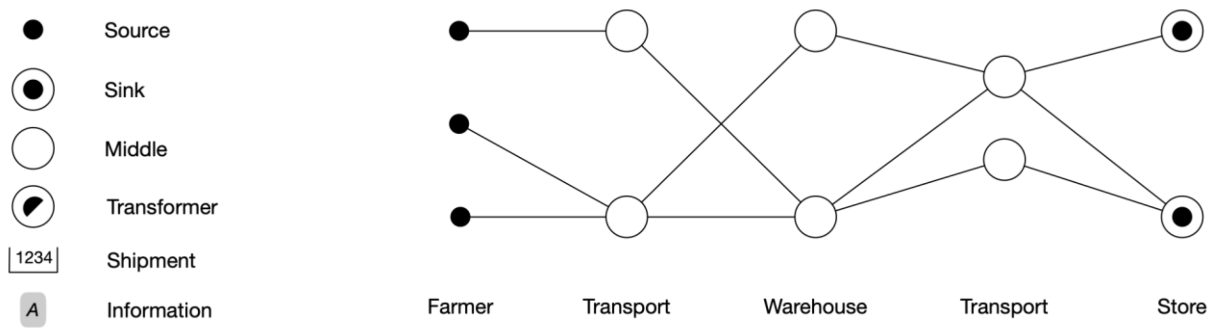


Figure 8: Simple flow of unprocessed produce from farmers to stores.

Sources, sinks, middle and transformer nodes define how asset lifecycle states can change. The table below lists which node types (roles) are allowed to initiate a state change.

Table 4: Types of nodes being able to initiate asset lifecycle changes.

		From state			
		Non-existent	**In transit**	**Finalized**	**Released**[13]
To state	**Non-existent**	-	-	-	-
	In transit	Source Transformer[14]	-	-	-
	Finalized	-	Sink	-	
	Released	-	Any	Sink	-

A transformer node may create assets in the system in a limited way (transforming) to cover two use cases: refining and repackaging. See below for a figure of an example of how transformation (refining) would work for a coffee shop's supply chain. The initial inputs to the system are coffee beans and raw milk. The roastery transforms the raw beans into roasted beans, and the dairy produces cream and milk from the raw milk.

[13] "Released" column includes all variants of released state.
[14] Only under certain conditions, see below.

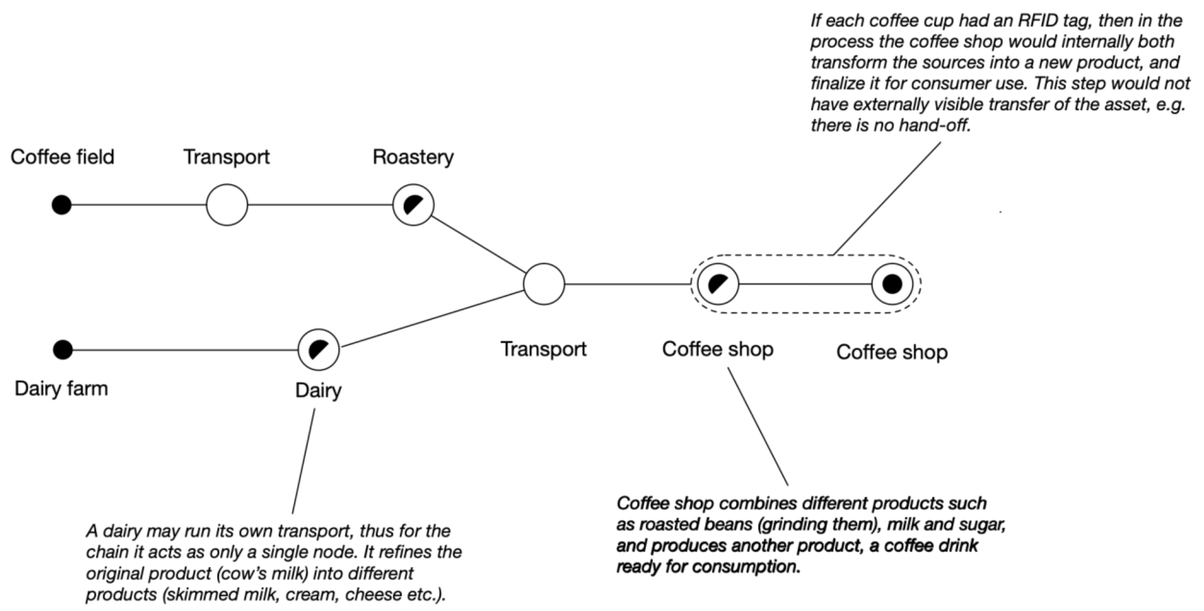

If each coffee cup had an RFID tag, then in the process the coffee shop would internally both transform the sources into a new product, and finalize it for consumer use. This step would not have externally visible transfer of the asset, e.g. there is no hand-off.
Coffee field
Transport
Roastery
Transport
Coffee shop
Coffee shop
Dairy farm
Dairy
A dairy may run its own transport, thus for the chain it acts as only a single node. It refines the original product (cow's milk) into different products (skimmed milk, cream, cheese etc.).
Coffee shop combines different products such as roasted beans (grinding them), milk and sugar, and produces another product, a coffee drink ready for consumption.

Figure 9: Example of an asset flow that transforms its goods (refining in this case).

These refined products are shipped to the coffee shop, which performs its own transformation and repackaging step. If the original raw coffee bean shipment was 1 ton per unit (large sacks), the roasted beans may be shipped to the coffee shop in 20kg sacks. The beans are ground and brewed at the coffee shop. A single cup does not consume all the 20kg from a single sack. The coffee shop has both transformed (roasted, brewed, foamed milk and combined all together) and repackaged (used only part of the shipped asset of roasted beans and milk from a carton). Here the transformed and repackaged asset has **not** been transferred between entities — the last steps are internal to the coffee shop.

Finally, one must keep in mind that **assets** are different from the **containers** they travel in. Containers may be re-used, whereas each asset in the system has an unique identifier. The figure below shows this process where the warehouse repackages the original asset ABCD packed at farmer into container 1234 into two smaller containers 986 and 987 which are given asset identifiers DCEB and BCDE. What is important is that the new assets are linked back to the original asset (shown with dotted arrows).

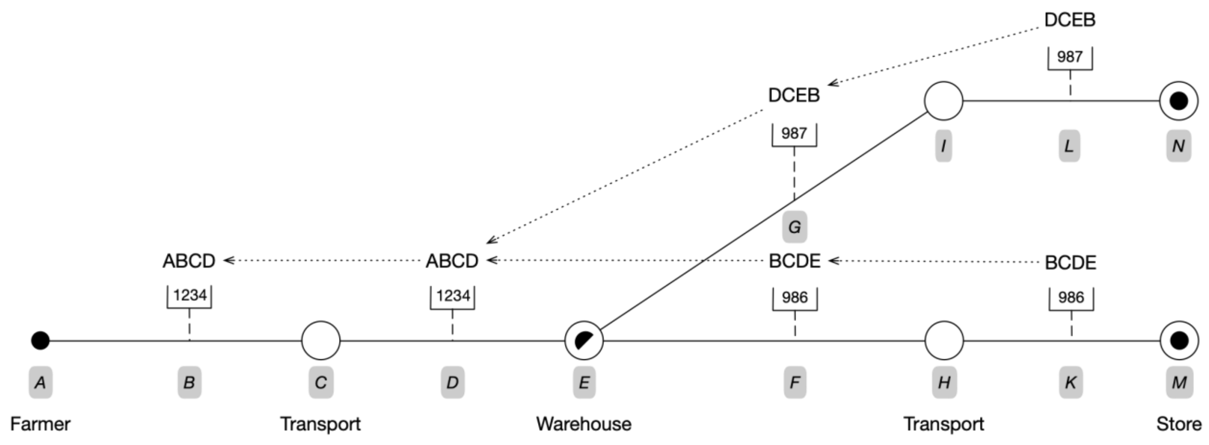


Figure 10: The containers assets are transported in may change arbitrarily, in this case due to repackaging of original container into smaller containers sent to different stores.

Thus, while transformers may create new assets, these **must** be linked to existing assets (that they are currently in possession of). In the above example, once DCEB and BCDE are finalized, the customer may query their information. If DCEB was queried, then any information from N, L, I, G, E, D, C, B, and A would be available for the consumer. Any data specific to the other repackaged asset (F, H, K, and M) would not be available to a consumer querying asset DCEB, however.

5.3.3 System entities

The figure below tries to represent the different main entities in the system (see below for those **not** in the figure, black and gray differentiate between different use cases described in examples).

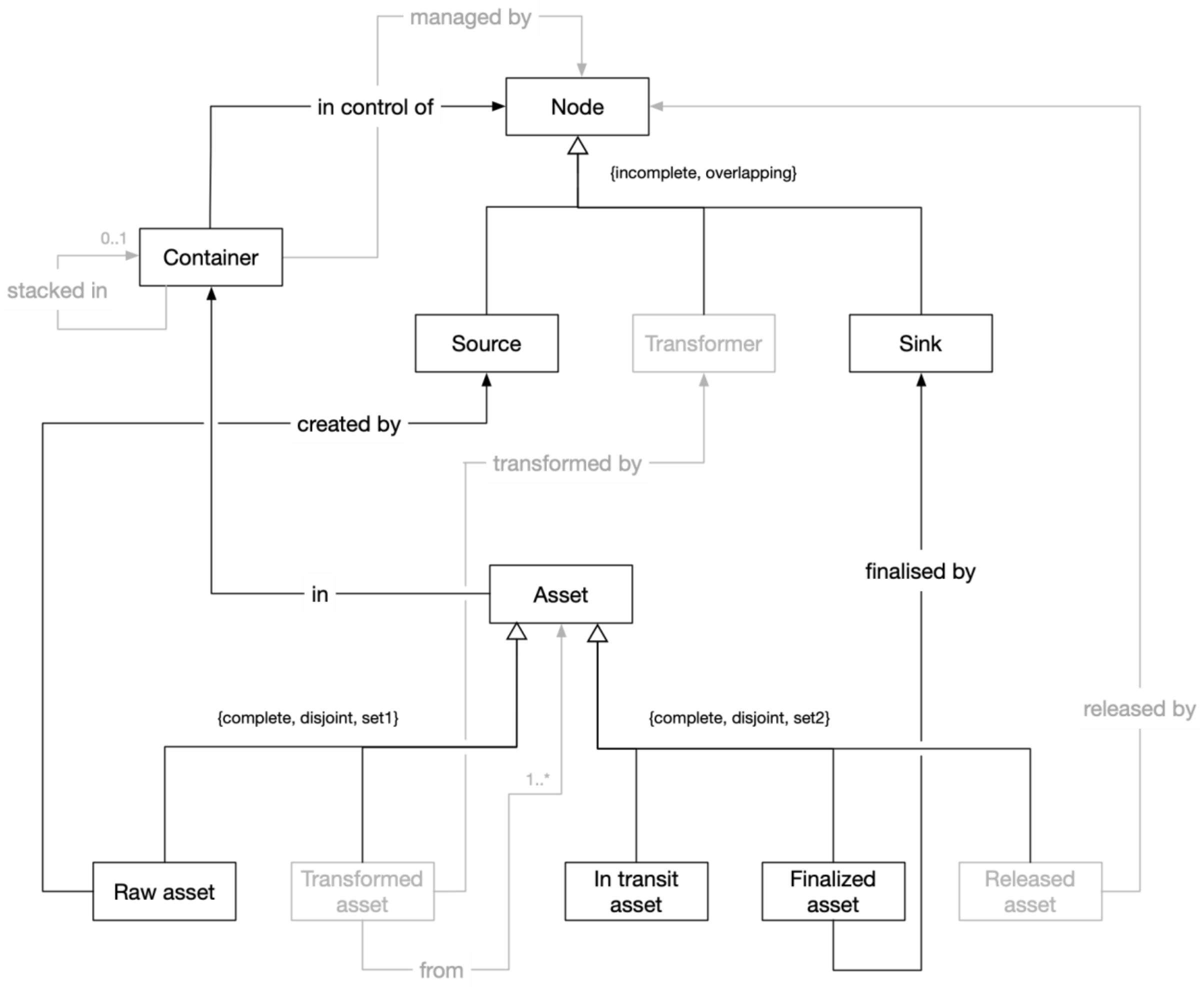


Figure 11: Entities and their relationships in the system. The difference between black and gray portions depends on the scope of the system (see examples later).

Note that the complete/disjoint/overlapping/set inheritance modifiers mean that a member can be simultaneously both a transformer and a sink, or just a member (such as a warehouse). These can be thought as roles that a member assumes as needed. Similarly, the modifiers for an asset's inheritance mean that an asset is either a raw asset, or a transformed asset, and also simultaneously one of transit, finalized or released asset types.

Raw assets are always created by a specific source, and transformed assets have to originate from one or more other assets (not necessarily raw assets!), which have been transformed by a transformer.

It is important to note that entities within this system are essentially *companies* and not individuals. While the asset is being manipulated during picking, transit or in store by a specific individual, tracking this information is the responsibility of a particular entity (if they wish to do so).

5.4 Claims, information, purposes and the provenance chain

5.4.1 Overview

While the sections above describe how assets flow within a network of nodes, this flow needs to be augmented by information that relates to the assets and their handling. First, we have to define more specifically what is "information", "purpose" and "claim" in this context.

5.4.2 Purposes of information use

In the logistics chain, there is a lot of information that **may** be necessary to be disclosed at some point, but usually **not immediately,** and often ano **audit trail of access** also needs to be defined, in addition to the specific purpose of information access. Consider the following example:

Grapes are picked and stored in a smart container at the field. The information associated with the grapes consists of: their type and variety (grapes), picking time, growth condition history, identity of the growing farm, identity of the picker(s). While the container is in transit, the information associated with the truck and transit operation includes temperature of the truck cargo area, driver identities, truck identity etc. When the container of grapes is at a warehouse, the location of the container in the warehouse and temperature profile of the storage area may be recorded. Also, all hand-offs of the grapes between different parties have information on the persons doing the hand-off, potentially location data, time and date etc.

Now, some portion of this information is either confidential or under GDPR or both. The identities of the pickers, truck drivers, people doing hand-offs and warehouse employees are most definitely under GDPR, and should not be normally exposed to other parties and not to the consumer. Yet, this information may be needed during disputes or audits.

Some of the information would have to be visible to active parties (current holder, or recipient of a hand-off process), such as the asset information itself, and information of the container. The recipient at the warehouse needs to be able to check that 1) the container contains what is claimed that it contains (grapes vs. apples) and 2) the container contains what the warehouse or store actually ordered from the farmer (comparing against internal ledger).

Some of this information is relevant to consumers, but should be visible only when the asset is in the "finalized" state (being sold off out from the visibility of tracking) to avoid other parties from "scraping" information unnecessarily.

Thus, all claims need to include the "purpose" of their use. Some of these purposes may be such that the underlying information is not accessible at all normally — the claimant is making a statement about its internal state for audit purposes (or even as a proof of its flawless operations for potential criminal investigations) where the underlying information is exposed through manual (verified) processes.

While there may be many purposes, the table below identifies some purposes that have been identified as (potentially) necessary:

Table 5: Description of identified claim purposes.

Purpose name	Scope	Description
consumer	Accessible only if the asset is finalized	Information that is relevant for the consumer, including the asset type, but any relevant handling and processing

		information (the actual data is very much specific to the type of asset and the logistics chain, and what information they wish to provide to increase consumer trust in the product)
asset	Historically or currently active nodes	Basic asset information that helps to identify it during transit
internal	Internal for the node	For information that the entity wants to make a public claim (attestation) of, but is not made publicly available
handoff	During handoff, visible to the recipient or sender (depending on which side makes the claim), and if transfer accepted, also visible later	Any information pertaining to the handoff in particular (such as the name of person whose identity needs to be confirmed, if needed for security purposes)

Some of these claims make only sense at specific points of time. A description of the asset itself can only be made when it is created in the system (raw asset or transformed asset).

A mechanism needs to be in place to have *scopes* for purpose identifiers so that entities within the system are able to create purposes without the risk of conflict within a single logistics chain, or between different logistics chains. These *scoped purposes* could be used for example between a warehouse and a trusted transporter to allow them to (securely and auditably) share more information than with other parties.

5.4.3 Claims and information

Claims in their core are about irrefutable statements that the claimant has specific information that, if later revealed, can be confirmed to have been the information referred in the claim. In more concrete terms, the claim must include a hash of the information, or some other undeniable and unforgeable mechanism to do so.

In general, the claim is a structure of <claim body, signature>, where the author and signature may be implicit in the case of blockchains (the claim body being a signed transaction), or explicit for other forms of claims. The author would identify the entity making the claim.

The claim body then consists of <asset, author, timestamp, purpose, validator type, validator data> and the validator is described in the table below.

Table 6: Description of different claim validator formats.

Validator type	Scope	Validator data	Information storage format	Description
ethereum-json	Information stored in Ethereum blockchain	<block hash, transaction hash>	JSON-encoded string in the transaction data field	The claim's information is stored as a (dummy) transaction as part of the entity's private blockchain
ethereum-contract	Information stored in a smart contract state in the Ethereum	<block hash, contract address, view call parameters>	Returns a list in the form of (key1, value1, key2, value2, …)	The claim's information is stored as part of the given smart contract's state which can be retrieved with the given call parameters

	blockchain			(view method)
sha512-json	SHA512 hash of a JSON string	<id, hash>	JSON that can be generated from the entity's own internal data identifier <id>	The underlying storage may be anything (such as a database row) as long as it can be reliably identified by the given id and always serialized to the same JSON string (stable serialization requirement)

Note that in general, an entity may only make a claim of an asset when:

- It is in possession of (through holding the container that the asset is stored in)
- It is in the process of receiving an asset (container)
- It is the owner of a container (smart containers with their own sensing are often not owned by those who are handling them)
- It creates it through transforming an existing asset (refining or repackaging)

5.5 The "System"

While all the above does not mention a centralized "system" entity, in practice one is needed to coordinate interactions between different members. While **theoretically** members could do all interactions bilaterally, this leads both to usability problems (discovery, trust, etc.) and also to loss of visibility of the asset provenance data in situations where a member refuses to release the preimage of claims they have made during asset transit. Thus, it is likely that (at least for some cases) a centralized "system" is deployed.

How does this work if the goal is to have a decentralized system? The way is to let the centralized system operate in a *transparent and verifiable* manner. Thus the goal of centralization is to increase the performance of the system, but make its actions verifiable by all entities interacting with the centralized portion.

5.6 Interactions

5.6.1 Overview

Note that having a ledger and smart contract here will include additional steps when the interactions need to be recorded there. The interactions below are focused on "user visible" actions. The detailed ledger-related operations (micro-operations) are described later.

5.6.2 Hand-off of assets

The sequence diagram below shows a possible sequence of events. The farmer (source) registers a shipment of pomegranates, and a truck (transport) picks it up. The truck driver checks the shipment and checks (theoretically at least) the shipment against what is specified in the hand-off claim made by the source (as part of the transfer initiation). After acceptance, the asset is marked to be in possession of the transporter. The transporter may update the claims (*their* claims) during transport, recording, for example, temperature data and such.

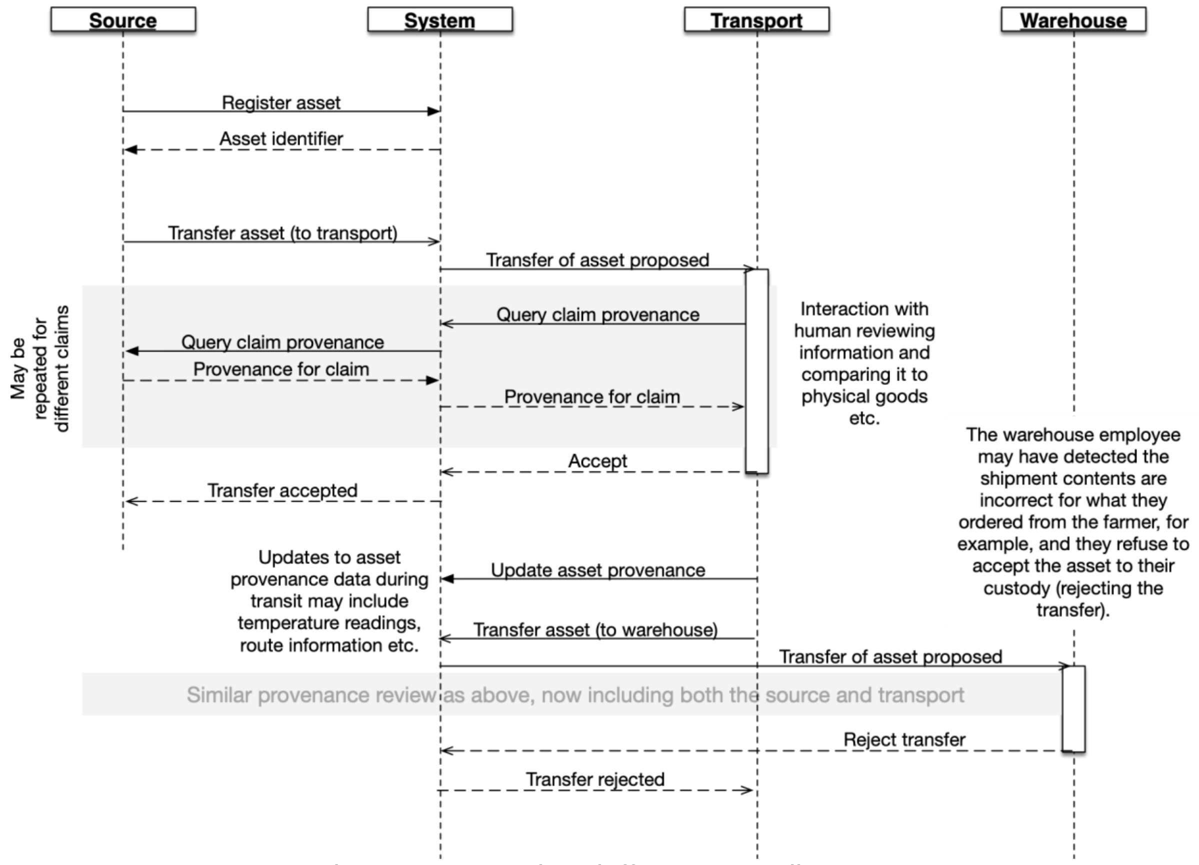

Source
System
Transport
Warehouse
Register asset
Asset identifier
Transfer asset (to transport)
Transfer of asset proposed
May be repeated for different claims
Query claim provenance
Query claim provenance
Provenance for claim
Provenance for claim
Interaction with human reviewing information and comparing it to physical goods etc.
Accept
Transfer accepted
The warehouse employee may have detected the shipment contents are incorrect for what they ordered from the farmer, for example, and they refuse to accept the asset to their custody (rejecting the transfer).
Updates to asset provenance data during transit may include temperature readings, route information etc.
Update asset provenance
Transfer asset (to warehouse)
Transfer of asset proposed
Similar provenance review as above, now including both the source and transport
Reject transfer
Transfer rejected

Figure 12: Asset handoff sequence diagram.

Finally the shipment arrives at the warehouse. The warehouse employee checks the shipment, compares its contents, the provenance information for the claims made against the asset by the source, and notices that the original order was for grapes and not pomegranates. The employee refuses to accept the shipment, rejecting the transfer initiated by the transport. The rejection may include information (claims) about the reason for rejection.

What is not shown in the diagram, but is likely to follow, is that the transport company would negotiate with the source about the fate of the pomegranates, and either the transport company would dump them (“release” operation), take them to another retailer or warehouse, or take them back to the farm (transfer). If the truck driver tries to be enterprising and shift the surplus pomegranates themselves, and the farmer realizes they have not been paid for, the farmer can see from the ledger that they were rejected, and are still in the possession of the transport company (which would then do some internal auditing on the dispute).

5.6.3 Asset creation

This is a very straightforward operation for sources — they will just define the asset claims, and submit those with the asset creation. The originator may update other claims, and update the container of the asset before it is handed off to the next entity. Regardless, the original asset claim itself should remain immutable thereafter.

For transformed assets, this is a bit more complicated. The source assets (there may be more than one) must be identified. The source assets must be useful (not finalized or released) and in possession of the transforming entity. The asset claim itself is provided by the transformer.

5.6.4 Asset release

There may be a need to release assets (as in "released to outside the system's traceability") without finalizing them. This may be due to spoilage, sale of the asset to an entity outside the system scope etc.

The asset must be in control of the releasing entity.

5.6.5 Asset finalization

Finalization of an asset changes its state. After finalization any customer claims of the asset or its source assets (and so on) can now be accessed publicly. To access any of the customer claims a proof of finalization is required (part of the finalization process).

5.6.6 Containers and container-specific claims

Containers are handled a little differently from assets. While an asset is unique over its lifetime, e.g. asset identifiers are not reused, a container can be used to ship different assets over its lifetime. While a container is typically the unit of reference during hand-offs (e.g. "transfer container XYZ and its assets to the other party"), normally claims are made against the asset by whoever is in possession of the container. Containers may also be located within other containers (grapes in a smart box stacked in a cage pushed into a cargo container).

However, a container itself is not necessarily owned by any party in the logistics chain. The container may also be a "smart box", that is able to actively monitor its own environment and send this information to its owner. Thus, a container may be owned by a defined entity, and the container owner is able to create claims that relate to the container's state at a specific time. This ties these claims to the particular asset.

5.6.7 Consumer interaction

The consumer is a special type of **anonymous** entity in the system that is allowed to access provenance data for "consumer" labeled claims, but only after the asset has been finalized. This requires a few things:

1. Asset finalization is irrevocable
2. Asset finalization results in a proof (or a claim, or similar) of the fact that can be passed to members as an access token (after possible mutual authentication) to retrieve provenance data
3. Provenance retrieval itself gives non-repudiable proof of the request

These are similar requirements for other state operations in the system, but worth reiterating here.

The diagram below shows a sequence of operations where the store initially finalizes an asset (that they are in possession of). The consumer interaction pattern may differ depending on whether the consumer accesses the system's web page to retrieve the information (or in JSON format for a mobile application), or whether this functionality is provided by the store itself (through a customer loyalty app, for example).

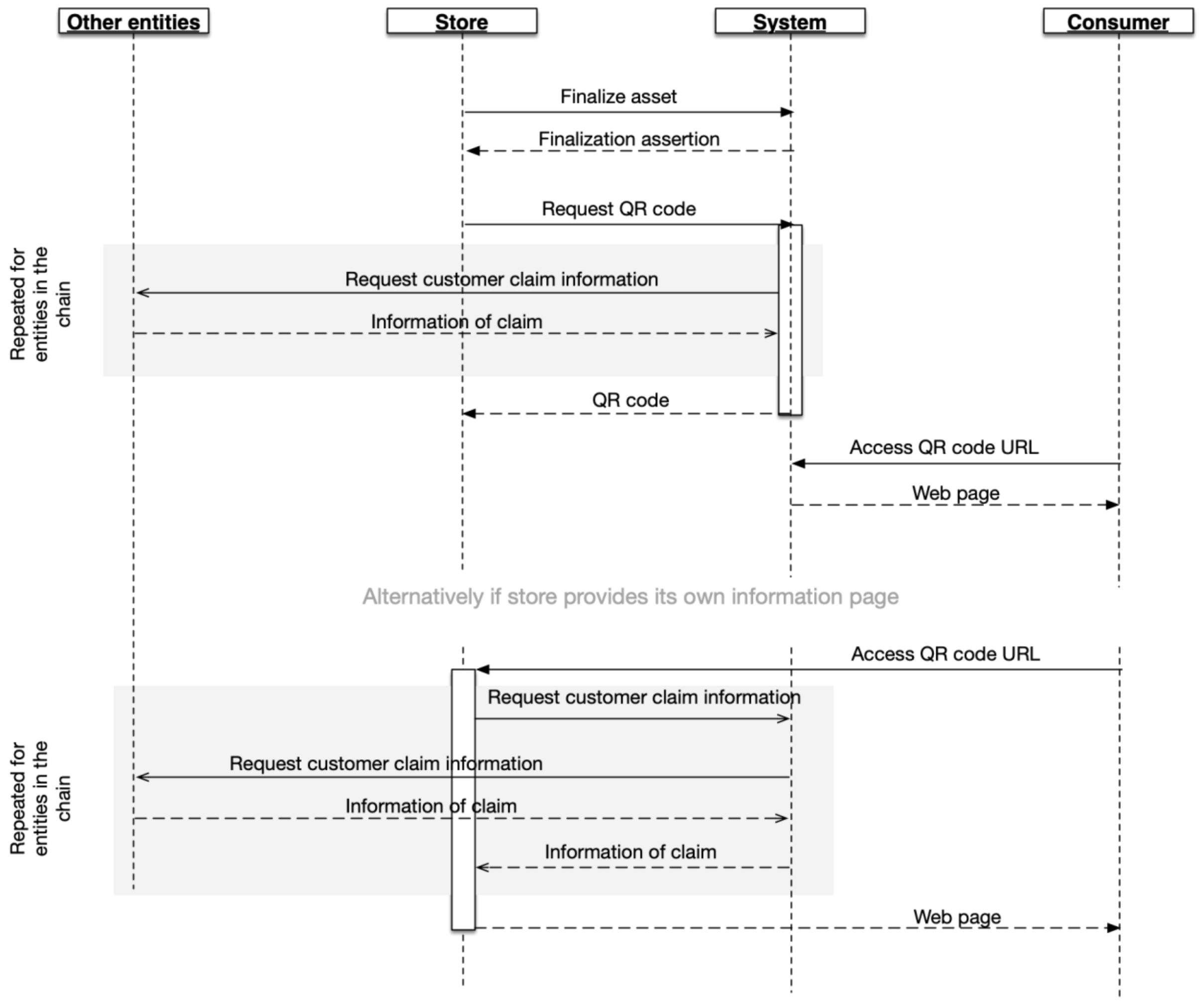

Other entities
Store
System
Consumer
Finalize asset
Finalization assertion
Request QR code
Repeated for entities in the chain
Request customer claim information
Information of claim
QR code
Access QR code URL
Web page
Alternatively if store provides its own information page
Access QR code URL
Request customer claim information
Repeated for entities in the chain
Request customer claim information
Information of claim
Information of claim
Web page

Figure 13: Asset finalization and customer fetching detailed information about a product.

In addition, it is possible to have different encodings and information on the QR code (or other URL), also at varying levels of auditability (although one must realize that on average, a customer is satisfied with a web page with the information, trusting the provider implicitly for its correctness). (The ID listed is not necessarily the asset id, but a “finalization id” created at finalization time.)

Table 7: Potential data embedded in an URL/QR code.

	Embedded information			
Scenario	**ID only**	**ID and customer information**	**ID, customer information, claims**	**ID, customer information, claims and claim verification**
No application No network	No information shown	No information shown	No information shown	No information shown
No application SPA cached locally No network	No information shown	Can show embedded customer information	Can show embedded customer information	Can show embedded customer information and

				validate claims locally
No application Network available	Can show all information	Can show embedded information, show updates from network and potentially additional information	Can show embedded information, show updates from network and potentially additional information, can request verification data for claims	As left, plus can perform claim validation
Application No network	No information shown	Can show embedded customer information	Can show embedded customer information	Can show embedded customer information and validate claims locally
Application Network available	Can show all information	Can show embedded information, show updates from network and potentially additional information	Can show embedded information, show updates from network and potentially additional information, can request verification data for claims	As left, plus can perform claim validation

The embedded data should be signed by whoever is providing them (store or system) to prevent the data from being altered or misused. A misuse scenario, for example, is another store copying another store's QR code. Since the data cannot be altered, the asset would be shown as being finalized in the original store, allowing the customer to be alerted to the misuse.

5.6.8 Dispute resolution

While not described in detail, the system should allow entities in the system to open disputes. These would be logged in the system. An open dispute gives the disputer potential to fetch additional claims not normally available. The system may enforce limits on this process, although these are more of a management and consortium governance setup than a technical protocol issue.

Generally, a dispute could be handled by one of the following scenarios (for this example, the dispute creator is a store, although it could be any entity in the logistic chain):

1. Store opens a dispute, analyzes the information behind the claim and decides internally not to pursue the dispute, and closes it. (It is likely that a store opening a high

number of internally resolved disputes would be subject to some governance oversight later…)

2. Store opens a dispute, analyzes the information, and discusses directly with another entity which agrees to a resolution. The dispute resolution is proposed to include the other entity, which then accedes to the resolution. (They may have agreed on a compensation over mishandled shipment, for example.)
3. Store opens a dispute, analyzes the information, and escalates it. At this point the store needs to provide its case to the governing body ("system") at which point the resolution rights are restricted to the governing body, who thereafter handles the dispute and its resolution.

Finally, it is of course possible for a store to handle a dispute out-system, directly with the warehouse or a transport company. In fact, this is the most likely outcome, with the formal and technological dispute mechanism used only for more serious situations.

5.7 Quality control

While out of scope of this BP, it should be noted that some auditing operations in the system can be automated and thus made more cost-efficient. While, by default, the information behind a claim is not shared, it would be possible to have a third auditing party with special access rights (with all accesses logged, of course)

5.8 Request traceability

During hand-off, the originator of the requestor needs to include all claims it has made in the transfer request (if they are tracked by the system, this can be implicit). This is required to allow the asset flow to be reconstructable even in the situation that the "system" component would fail to comply. Here, if the store has recorded (internally) the transfer from a transporter, it can go back and look for the claims made by the transport company. These in turn should contain information linking to the warehouse and so on until asset sources.

5.9 Examples

The above BP description attempts to encompass a large amount of variability and configurability. Such a system may not be feasible to build in a single step, thus a description of a subset of such system that still retains the essential features of the BP is described below.

In particular, let's imagine a system where:

- Only a single type of container is used throughout the flow of an asset
- Containers cannot contain other containers
- The asset does not change its container during its lifecycle (from field to store shelf in the same container)
- Asset lifecycle always ends up in a finalization step (no release)
- No asset transformations occur, e.g. all assets are "raw" assets (generated at source)

This system can be seen in the earlier entity diagram where it is defined by the black diagram part only with the gray portions omitted.

6. IoT resource access

6.1 Overview

The BP for IoT resource access provides clients the authorization to access a Thing (resource) after they have paid or otherwise compensated[15] the resource owner for the requested access. Such a service can be, for example, part of the Mobile Gaming Pilot scenarios where games involve interaction with IoT devices [SOFIE D5.1]. Although we assume that the client requesting resource access is the entity that pays the resource owner, this can be adapted, if needed, by the specific use case. For example, in the Mobile Gaming scenario the game developer, rather than the player, can be responsible for handling payments to the IoT device owners. Resource access can involve reading or writing data, as well as actuation. Also, access can be authorized for different time intervals.

The generic model of the BP has the following properties:

- Things are constrained, hence they cannot perform complex cryptographic functions or directly interact with the blockchain
- Things are not connected to the Internet
- The communication channel between the Client and the Thing is not secured
- Existing authorization and authentication standards are utilized
- Smart contracts can verify that authentication servers and Things share a secret key (hence there is a trust relationship between them)
- Information stored on the blockchain, which is immutable and non-disputable, can be used for accounting, auditing, and in the case of disputes
- Low complexity
- Client privacy protection

A discussion of constraints in IoT environments is contained in [RFC7228]. Various use cases for IoT constrained environments are discussed in [RFC7744].

6.2 Process flows

6.2.1 Configuration and smart contract creation

During this phase the resource owner interacts with the backend (authorization server) and the Thing to configure the latter with a secret key that is shared with the Thing and the backend; this is necessary because, after its configuration, the Thing can be disconnected. The backend is responsible for handling authorization requests, and in particular will perform actions that require secret keys or involve private information which cannot be published on the blockchain. Next, the resource owner creates a smart contract with information that binds the backend and the Thing (resource server), and can include the price for different types of resource access.

The specific steps are the following:

- The resource owner registers the Thing with the backend and receives a secret key
- The resource owner configures the Thing with the secret key
- The resource owner creates a smart contract and updates it with information that includes the following:
    - Backend public key
    - Resource server (Thing) URI

[15] From here on, we refer as "payment" any compensation method that may be used, including ones where the compensation takes place in completely non-monetary means.

- ○ Resource access price. There can be a different price for different levels of access (e.g. read or write) and/or access intervals

Note that the first two steps above do not involve interaction with the blockchain, and can follow an appropriate configuration protocol. Also, the resource owner may need to be authenticated.

As part of the smart contract management, the resource owner can terminate the contract and the provision of the authorization services by the backend. Also, the owner can change the backend that provides authorization services to its resource, by binding the Thing to the new backend and updating the corresponding information in the smart contract.

6.2.2 Setup

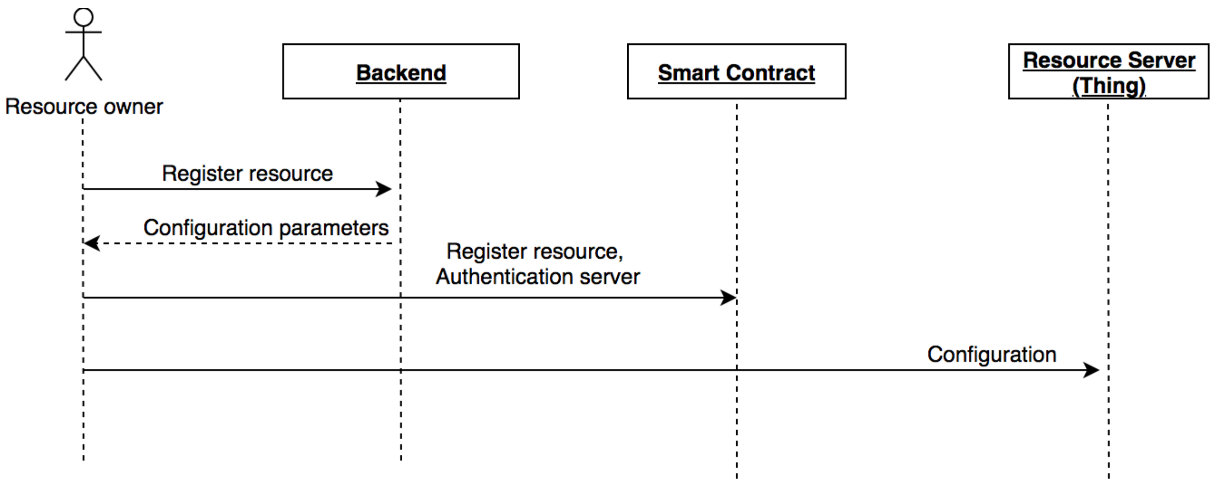


Figure 14: Resource access configuration and smart contract creation.

6.2.3 Resource access

Resource access can be based on authorization protocols, such as OAuth2 and User-Managed Access (UMA2), through which the client obtains an access token that it can use to access the Thing. The addition of a blockchain serves to associate access authorization with the appropriate payment. Moreover, smart contracts can be used to encode different authorization policies, such as prices for different types and time intervals of resource access. Even more interesting is the case where authorization policies include dependence on various IoT events (or transactions) that are recorded either on the same DLT that is used for payments or on different DLTs; in the latter case, the dependence across different DLTs can be achieved using inter-ledger mechanisms, such as hash and time-locks. Such more elaborate smart contracts are currently being investigated and will be described in the next versions of the SOFIE Framework.

The client can initiate resource access by sending a request to the Thing, e.g. to obtain the address or ABI (Application Binary Interface) of the smart contract that is responsible for handling access to the Thing. Other alternatives are also possible, e.g. the client can send the request to the backend; this alternative assumes the client knows the backend or can find the backend utilizing a registry service, e.g. the Ethereum registry services. Moreover, the request sent by the client can include a challenge, to which the Thing can provide a response that can be used for authenticating the Thing but also for verifying the integrity of the Thing's internal state (attestation). The actual verification of the Thing's response can be performed by the backend, which can obtain the Thing's response to the challenge through the smart contract.

In the next step, the client interacts with the smart contract, which can involve depositing the amount of money that corresponds to the type and duration of access that is requested, in addition to sending the Thing's response to a challenge, as mentioned above. Once the deposit is performed, a request to the backend is made to create the access token and key. These can be encrypted and sent by the backend to the client through the smart contract (as shown in the figure) or alternatively from the backend directly to the client. The latter case can support off-chain payments, where the client and backend exchange directly payment transactions, without requiring them to be published on the blockchain, thus avoiding the corresponding costs.

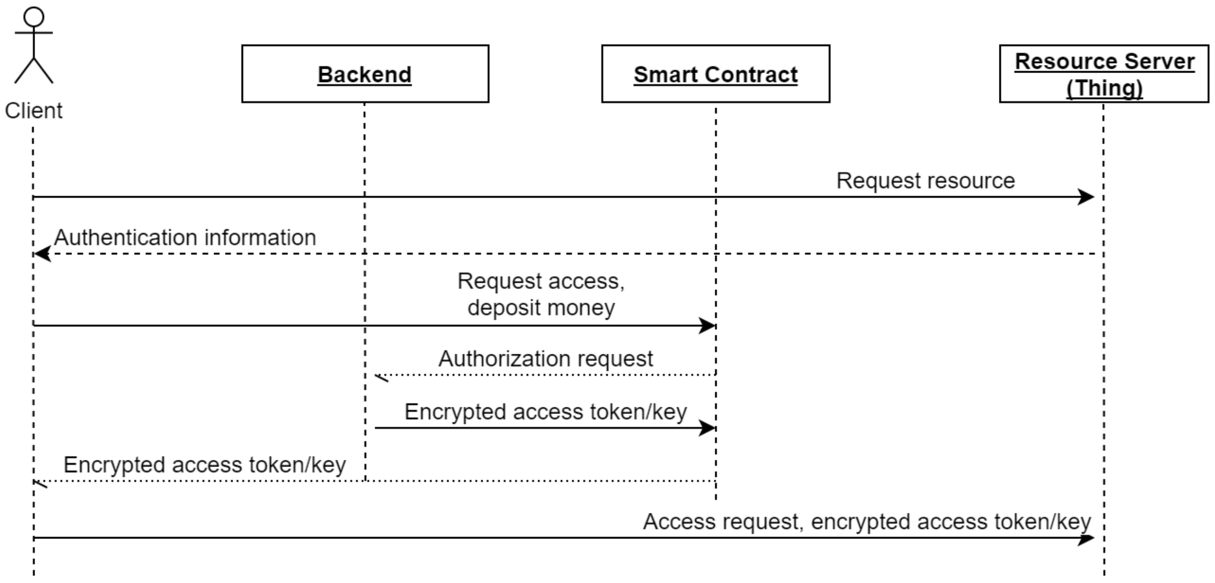

Client
Backend
Smart Contract
Resource Server (Thing)
Request resource
Authentication information
Request access, deposit money
Authorization request
Encrypted access token/key
Encrypted access token/key
Access request, encrypted access token/key

Figure 15: Resource access payment and authorization

When transmitted from the backend to the client, the access token and key can be encrypted. These alternatives correspond to a different privacy, and tradeoff between privacy and transparency. Note that all policy related actions can be implemented in the smart contract, except actions that require private keys, such signing or encrypting messages with a private key, which need to be performed off-chain (in the backend).

7. Conclusions

This deliverable presented the first version of the SOFIE federation framework. It describes framework's components along with three business platforms (offer marketplace, provenance chain for trackable assets, and IoT resource access). The updated versions of this deliverable will be released in 2019 and 2020.

References

[RFC7228] C. Bormann, M. Ersue, A. Keranen, "Terminology for Constrained-Node Networks,'' RFC 7228, IETF, May 2014.

[RFC7744] L. Seitz et al. "Use Cases for Authentication and Authorization in Constrained Environments," RFC 7744, IETF, January 2016.

[SOFIE D5.1] I. Oikonomidis et al. "Baseline System and Measurements", SOFIE Deliverable D5.1, June 2018. Available at: http://media.voog.com/0000/0042/0957/files/SOFIE_D5.1-Baseline_System_and_Measurements.pdf .