SupplyTree - The Inter-company Tamper-evidence Protocol for Supply Chain Traceability

Authors

Matthias Guenther, Robert Bosch GmbH, Economy of Things

Dominic Woerner, Robert Bosch Switzerland, Economy of Things

Abstract / Executive Summary

In a global economy supply chains are complex networks with heterogenous participants. From the perspective of a finished product it is hard to trace down the individual suppliers of various hardware and software components. This is problematic in particular in case of a recall, where affected products and responsible suppliers have to be identified. There are different architectural patterns to approach this problem of supply chain traceability. These range from centralized architectures to decentralized architectures based on blockchain technology and federated systems. Recently, blockchain-based systems have become popular due to their properties of auditability and immutability (i.e. tamper evidence). However, we will argue that these properties can also be achieved with a simpler, federated system called SupplyTree that utilizes concepts also common in blockchain systems, like cryptographic commitments, hash chains and (optionally) digital signatures, but abandons the notion of a global shared state. In contrast to Blockchains which are heavily replicated and require extensive standardization or a common code base, SupplyTree is distributed and loosely-coupled and hence provides much better scaling behavior - both from a technological and from an organizational perspective.

Introduction

Supply chains are complex networks containing a multitude of participants around the world. Participants range from small and medium sized companies to multi-national cooperations. Today, most participants in the supply chain interact only with their immediate suppliers and customers, and data exchange may happen over various channels from paper documents, e-mails and telephone calls to proprietary and standardized electronic data interchange (EDI) solutions.

Due to the heterogenous nature of these networks it becomes very cumbersome, time consuming and costly to track down the origin of a malfunction that appears in a finished product. To tackle these issues systems for supply chain traceability have been introduced.

In [1] a system for supply chain traceability is introduced as follows: "From an abstract viewpoint, a traceability information system can be thought of a single massive, centralized data storage capturing all the information about each lot along each stage of the supply chain."

However, the authors also point out that a centralized architecture has scaling issues and can hardly meet the dynamic requirements of a heterogenous group of supply chain participants. We may further add that a centralized architecture represents a single point of failure and thus a honeypot for attackers. In addition the central operator may exploit its position to extract unreasonable rents in the long run.

In recent years, decentralized architectures based on blockchain technology are being suggested as a novel alternative to the centralized architecture. Blockchains are peer-to-peer networks that are logically centralized but can be decentralized from a technical and organizational point of view. They are replicated state machines where state changes are represented by appending a set of signed transactions coined as block. Blocks are chained by cryptographic hashes. Therefore, the history is auditable and tamper-evident. These properties of logical centralization, i.e. to have a single source of truth, and the auditability make the technology appealing for supply chain applications. However, in contrast to traditional distributed systems, workload and storage is not shared by the peers, but replicated. Hence, each participant carries the burden of all other participants. This overhead is necessary to achieve a global shared state, which is not required for supply chain traceability from our perspective.

Furthermore, in most supply chains, participants are not willing to share data with all participants and may only want to share because of specific contractual pressure. Therefore, additional measures have to be taken to shield sensitive data.

In this paper, we suggest a loosely-coupled decentralized system based on federation, where data is kept at the source and only links with cryptographic commitments are travelling downstream in the supply chain. Thereby data is only shared if required and the company that owns the data stays in control of the data. Still, the data becomes tamper-evident due to sharing of the commitment, and the supply chain becomes traceable because of linking.

In contrast to centralized and blockchain architectures, the federated system is much more scalable. This is obviously true from a technological point of view, since it is distributed in the traditional sense, but also from an organizational point of view, since the participants are only loosely coupled by a minimal set of standardized web APIs.

The Protocol

SupplyTree is a combination of an append-only distributed data structure and a set of standardized APIs to create a federated system for the exchange of tamper-evident supply chain data. The goal is to keep data in control of the originating organization, but to share web links with cryptographic commitments that allow to easily retrieve associated data and detect modifications. A simple representation of the SupplyTree system is shown in figure 1.

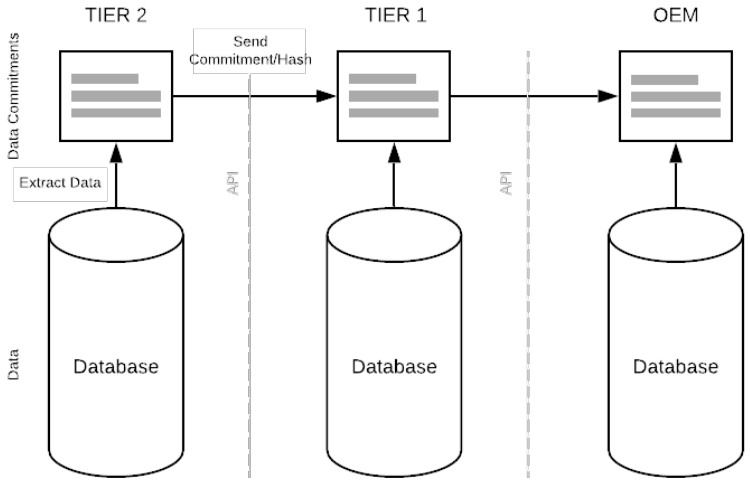

Figure 1: A simplified representation of the SupplyTree system connecting information systems of companies in a supply chain.

In the following, we will explore the data structure and protocol in more detail. From the perspective of a finished product, a supply chain has the structure of a tree. The finished product represents the root node.

The root node has directed edges to various child nodes representing different parts from different suppliers. This structure may continue until we end up with raw materials at the leaf nodes. A representation of this structure across organizational boarders is shown in figure 2.

Each node is created by a company at the specific Tier X (e.g. Tier 2) in the supply chain and is communicated to the next company in Tier X-1, its customer (e.g. Tier 1). Thereby the tree gets built up from the leafs to the root node through the production process with all participants adding their own data objects (“D”) to the structure.

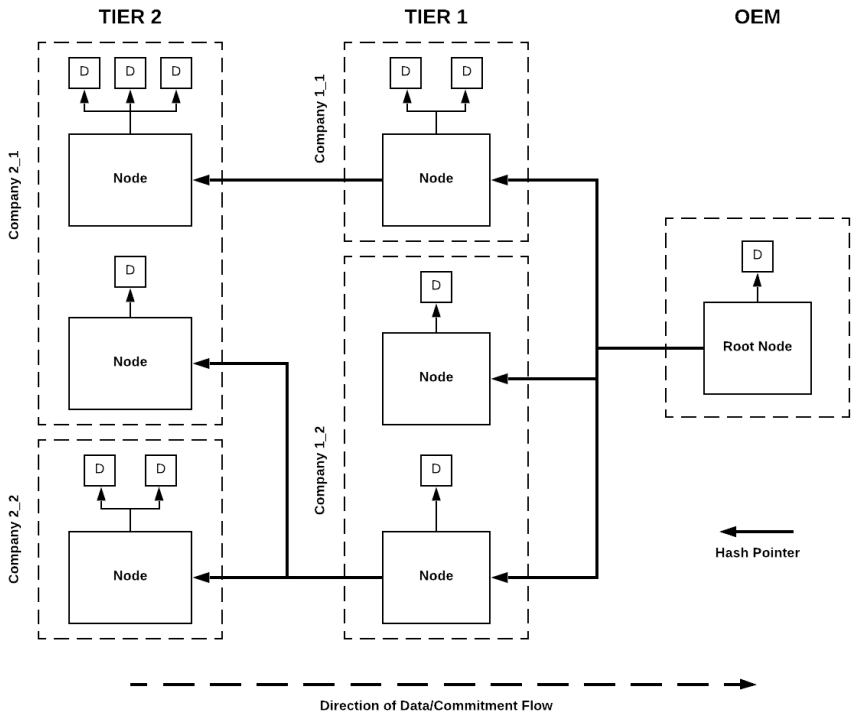

Figure 2: SupplyTree data structure. D represents data objects. Arrows represent hash pointers.

An edge is represented by a cryptographic hash of the respective child. Hence, the data structure becomes immutable, or more precisely, tamper-evident.

A simple JSON representation of a node may look as follows

```
{
    "reference" : {
        "0" : "https://supplytree.tierX+1.com/node/<SHA512_OF_TREE_OBJECT>",
        "1" : "https://supplytree.tierX+1.com/node/<SHA512_OF_TREE_OBJECT>"
    },
    "data" : {
        "0" : "https://supplytree.tierX.com/data/<SHA512_OF_DATA_OBJECT>",
        "1" : "https://supplytree.tierX.com/data/<SHA512_OF_DATA_OBJECT>",
        "2" : "https://supplytree.tierX.com/data/<SHA512_OF_DATA_OBJECT>"
    },
    "uuid": "https://supplytree.tierX.com/uuid/9973494216984ee5f78a",
}
```

The reference section contains the links to the child nodes. In addition, there is a data section. It contains a list of data objects that belong to the current node. These data objects are also referenced by cryptographic hashes in order to prevent modification of the data later on. Thus, a company down the supply chain has a commitment to the data but does not necessarily need to retrieve and store the data itself. Instead, the data can be retrieved if and when actually required, e.g. in the case of a recall. Then, the data object hash can be compared with the hash in the node object and any modification would become evident.

Although the node is uniquely identified by its cryptographic hash, there is the need for another unique identifier (uuid). This identifier allows to link the node to the physical item. At the point of production of the physical item, it receives an identifier. Often printed on a label or tagged with RFID. Existing standards might be used for the identifier (e.g. SGTIN, GS1 Digital Link [3]), but it cannot be the hash of the node object since the data belonging to the node might not be complete and adding data would lead to a new hash of the data object, and the node object subsequently.

Example from the Automotive Industry

In this example we'll go through an automotive example to show how the flow of data would look like in a real world use case.

We assume the following use case: An OEM (car manufacturer) receives an infotainment system to build it into the car. The infotainment system consists of several parts which are sourced from suppliers, e.g. the ECU (Electronic Control Unit). This use case can be also seen as the *digital vehicle* file which documents the parts built into a specific car. The example consists of two parts. The first part is the production process where the physical items travel along the

supply chain and get assembled. At this stage detailed production information is typically not required upstream. The second part is a recall scenario, where a malfunction in a part has been detected and the OEM has to identify which products are affected.

Supply Chain Participants

- **Tier 2:** Delivers the ECU
- **Tier 1:** Assembles the ECU into the infotainment system and delivers to the OEM
- **OEM:** Assembles the infotainment system into the car.

The production process

The data flow in figure 3 shows that Tier 2 creates data objects, hashes them and includes those links into the node object. The data typically comes from existing data sources like a MES (Manufacturing Execution System) or ERP (Enterprise Resource Planing) system.

In the next step the node object is hashed and made available through a link. The link is sent to the next party in the supply chain, Tier 1. This can be done in two ways:

- the link to the node object is sent,
- or the aforementioned UUID is printed on the physical product as a link through which the data is retrieved from the previous supply chain step.

In the example above, the link to the node object is sent to the next step in the chain.

Tier 1 also creates its own data object and adds both, the link to the data object and the link to the Tier 2 node object to a new node object. The new node object is hashed and sent as a link to the OEM.

The OEM does the same with its own data objects. The chain is now linked together and at any point in the future, any modifications to the objects can be detected and thus, be used as evidence in case of a dispute.

The relevant objects are listed in the appendix.

The recall

Now let's look into a recall situation. The OEM requests the node object content from Tier 1. Tier 1 sends back the content to the OEM. The OEM verifies that the content matches the hash of the object to detect any possible modifications. Afterwards, the references to other node objects are parsed from the node object

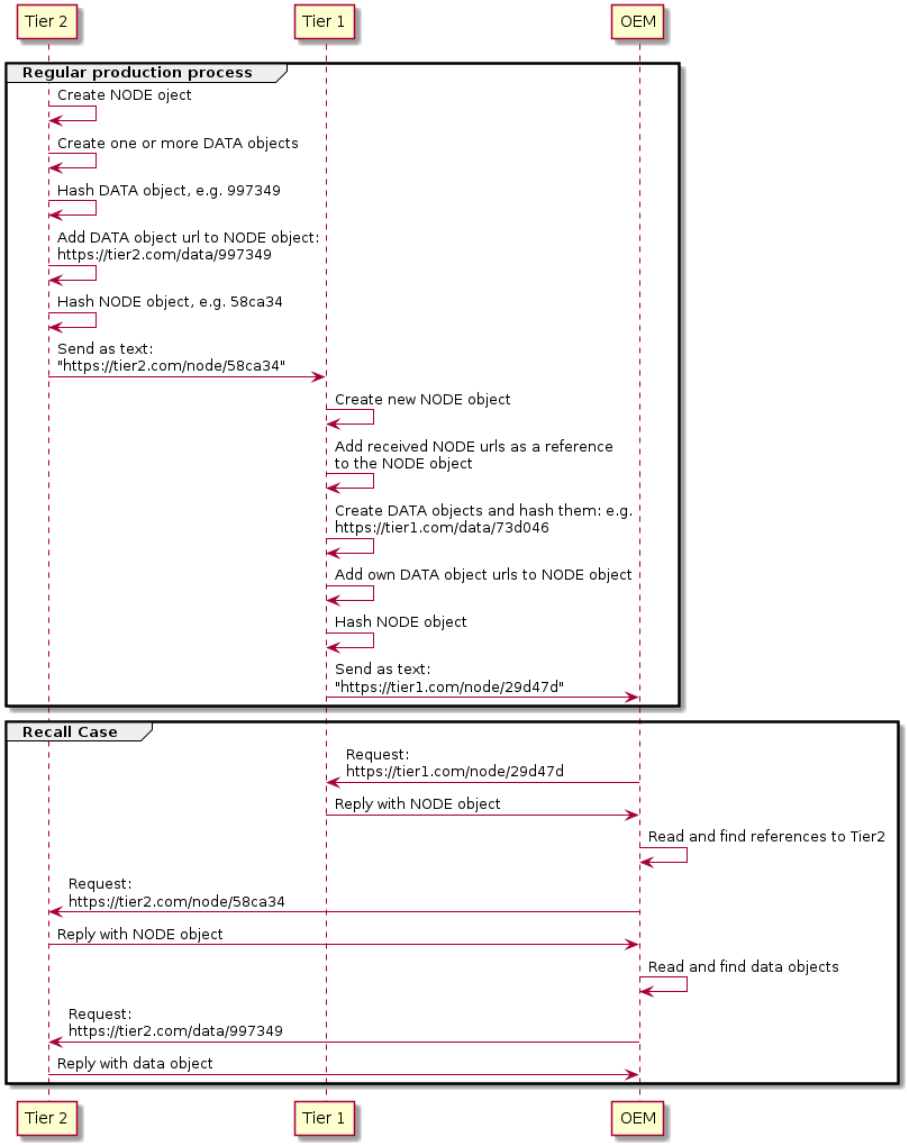

Figure 3: Data flow example

and a reference to the node object from Tier 2 is detected. The OEM requests the Tier 2 node object content, checks the hash and parses its content. The result shows a data object that is fetched from Tier 2.

Because of the hashes and links between the objects, any change to the data since the time when the hashes were forwarded along the supply chain, can be detected.

In this example we assume, that the ECU was manufactured with material that later on has been identified as being dangerous. Now, since the OEM can read this information, the OEM can recall only the affected cars from the market because of the given product instance data.

Integration into the existing System Landscape

Figure 4 shows how the integration of SupplyTree into an existing landscape would look like. On the left side, a simplified version of the *automation pyramid* [2] shows the relevant IT systems in a factory. At the top, the EDI system shows how companies already communicate digitally in the *order to cash* process. In the middle, the arrows show the information flow from an EDI system, through the different layers of the automation pyramid down to the assembly line and back up to the EDI system.

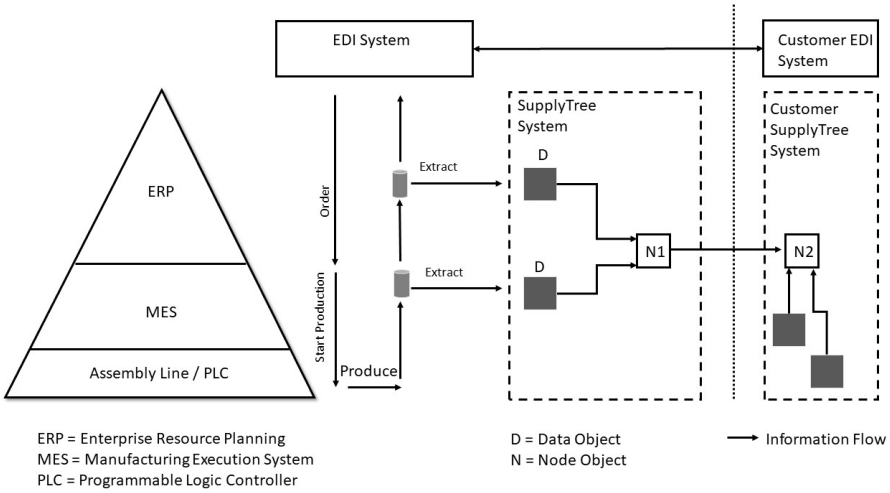

Figure 4: System Integration

The SupplyTree protocol doesn't change this flow of information. It rather integrates well into this landscape. Data can be extracted from different existing systems, namely ERP and MES. The right side of the picture shows how

SupplyTree uses the extracted data to create a tamper-evident traceable chain of that data, across multiple companies.

Extensions

The SupplyTree protocol as described, brings tamper-evidence into the supply chain. In real world use cases, some extensions can be helpful.

Non-repudiation with Digital Signatures

The SupplyTree protocol is based on the assumption, that data and node objects will exist forever, or a very long defined time - at least as long as anyone might want to traverse the tree structure. In real world, participants in the supply chain might delete such objects, either by accident or on purpose. In this case, proof is needed, that at the moment of building up the tree, a certain object has existed.

This can be reached by digital signatures on the links that are included as *references* in the *node object*. The extension to the protocol looks as follows:

```
{
    "reference": {},
    "data" : {},
    "referenceSignature" : {
        "0" : {
            "type" : "RsaSignature2018",
            "signature" : "..."
        },
        "1" : {
            "type" : "RsaSignature2018",
            "signature" : "..."
        }
    }
}
```

The **referenceSignatures** are linked to the **reference** items. By adding a digital signature to the reference item, the sender commits to the existence of the object. As part of the node object, the signatures are part of the tamper-evident chain.

The verification of digital signatures requires some kind of public key infrastructure (PKI) to provide a trusted link between supply chain companies and public keys which is out of scope of the SupplyTree protocol.

Access Control

In order to traverse a SupplyTree, APIs of web servers hosted by several companies have to be accessed. Since the URLs of node and data objects include cryptographic hashes, the root node can be viewed as a capability to access to the resources along the tree. Some companies might be reluctant to rely solely on this kind of access control that allows everyone in the possession of the respective node object to access the data objects. In this case additional access control mechanisms can be implemented at the concerning API. However, we strongly suggest that the node resources stay unprotected, such that everyone down the supply chain is able to traverse the tree and can retrieve signed node objects. This entails that a company at a lower Tier gets information about the suppliers of its suppliers. Sometimes this might not be desired. If this is the case, access could also be restricted for the node resources and made only available to the direct customer or as an additional feature, only for requests signed by a defined legal authority, e.g. Federal Motor Transport Authority ("KBA" in Germany).

An *Access Control* field for data object references might be an extension to the SupplyTree protocol.

Hidden Fields and References

As promising a fully transparent supply chain looks like for many use cases, often the reality is different and such transparency is not desired. Companies don't want to reveal their suppliers to their customers. Still, there are situations in which those information must be revealed, e.g. in case of a recall of cars, food, medical drugs or other goods.

For such cases, the RFC6920 [3] *Naming Things with Hashes* is a good mechanism that perfectly fits into the SupplyTree protocol. It allows certain fields to be hidden behind a hashed value and only be revealed on a demand basis.

```
{
    "reference" : {
        "0" : "https://supplytree.tier1.com/node/8e25e2ea9bb159dab1006c3b703cf272a271d40818eec4187ba3e2c1a93ff71e"
    },
}
```

```
{
    "reference" : {
        "0" : "ni:///sha-256;9d47304ab729fd13b52c1e683d8933dcad3217aa08532cac0aeef48b54dbe211"
    },
}
```

The SupplyTree hash chain is built from node objects with such hidden fields. Later, in case such information must be revealed, a request is sent to the API including a list of such hidden fields and if access is granted, the plain text is returned to the requester. There, the hash can be calculated from the plain text to verify it has been used to create the hidden field hash and has not been changed in between.

Details of how to create hashes and request / reveal hashes are listed in the appendix.

Machine-readable Data Objects

Data objects in SupplyTree are binary blobs and the content is in principle unrestricted. In some cases this might be pdf documents or images. In order to inform the client that accesses a specific data resource what to expect the content type should be defined. However, the real power can be unlocked if data objects are machine readable. Therefore, standardized JSON templates or JSON-LD documents are recommended. Another approach would be to leverage the EPCIS [4] event structure.

A *Content Type* field for data object references are an extension to the SupplyTree protocol.

IoT-based real-time Data

In some cases the transport of goods should be monitored in real-time, or sensor data is collected over the course of the transport. These data might be the location of the goods, shock or vibration data or environmental conditions, such as temperature or humidity. In this case a special type of data link can be added to the node object. This link can not be identified by its hash, because the content is changing. However, if the measurement values should be non-repudiable,then they should be digitally signed.

Conclusion

SupplyTree is a lightweight protocol to solve trust issues along the supply chain. It does this by focusing on a tamper-evident data chain along the supply chain, that requires the participating parties to only agree on a minimalistic set of API standardization. No central - or logical central - instance and therefrom derived additional effort is required.

Appendix

Relevant Objects

Node 58ca34

```
{
    "reference" : {
    },
    "data" : {
        "0" : "https://tier2.com/data/997349"
    }
}
```

Node 29d47d

```
{
    "reference" : {
        "0" : "https://tier2.com/node/58ca34",
    },
    "data" : {
        "0" : "https://tier1.com/data/73d046"
    }
}
```

POST request from Tier 2 to Tier 1

```
{
    "node" : "https://tier2.com/node/58ca34"
}
```

POST request from Tier 1 to the OEM

```
{
    "node" : "https://tier1.com/node/29d47d"
}
```

Hidden Fields

Plain Text Field

```
{
    "reference" : {
        "0" : "https://supplytree.tier1.com/node/8e25e2ea9bb159dab1006c3b703cf2
        72a271d40818eec4187ba3e2c1a93ff71e"
    },
}
```

Creating the hash

```
$ echo -n "https://supplytree.tier1.com/node/8e25e2ea9bb159dab1006c3b703cf272a2
71d40818eec4187ba3e2c1a93ff71e" | sha256sum

9d47304ab729fd13b52c1e683d8933dcad3217aa08532cac0aeef48b54dbe211  -
```

Using the hashed field value as a hidden field

```
{
    "reference" : {
        "0" : "ni:///sha-256;9d47304ab729fd13b52c1e683d8933dcad3217aa08532cac0a
        eef48b54dbe211"
    },
}
```

POST request to reveal hidden fields

```
{
    "ni:///sha-256;9d47304ab729fd13b52c1e683d8933dcad3217aa08532cac0aeef48b54dbe211" :
    ""
}
```

Reply with revealed field content

```
{
    "ni:///sha-256;9d47304ab729fd13b52c1e683d8933dcad3217aa08532cac0aeef48b54dbe211" :
    "https://supplytree.tier1.com/node/8e25e2ea9bb159dab1006c3b703cf272a271d408
    18eec4187ba3e2c1a93ff71e"
}
```

References

[1] Bechini, Alessio, et al. "Patterns and technologies for enabling supply chain traceability through collaborative e-business." Information and Software Technology 50.4 (2008): 342-359.

[2] Hollender, Martin. Collaborative process automation systems. ISA, 2010. Page 5.

[3] GS1 Digital Link Standard, https://www.gs1.org/standards/gs1-digital-link

[4] GS1 EPC Information Services (EPCIS) Version 1.1 specification, https://www.gs1.org/epcis/epcis/1-1

[5] Internet Engineering Task Force (IETF) RFC6920, Naming Things with Hashes, https://tools.ietf.org/html/rfc6920