Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

MASTER'S THESIS IN INFORMATICS

A Trustworthy Process-Tracing System for B2B-Applications based on Blockchain Technology

Stefanos Georgiou

Technical University of Munich

Department of Informatics

Master's Thesis in Informatics

A Trustworthy Process-Tracing System for B2B-Applications based on Blockchain Technology

Ein auf Blockchain-Technologie basierendes vertrauenswürdiges Prozessüberwachungssystem für B2B-Anwendungen

Author:	Stefanos Georgiou
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Dr. Holger Kinkelin Dr. Heiko Niedermayer Sree Harsha Totakura, M. Sc.
Date:	May 14, 2018

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, May 14, 2018

Location, Date

Signature

Abstract

Supply chain management is the active process of managing all supply chain activities, from point of origin to point of consumption. It is an essential part of businesses and is crucial not only for the benefit of the companies involved, but for their customers too. Nowadays, organizations participating in the supply chain transact with each other but maintain separate records of these transactions. These private data silos are costly to maintain and inefficient due to data redundancies as well as the lack of information sharing. Databases which store these records are additionally vulnerable to attacks due to the centralized methods of storing data. Through such attacks data can be altered or lost. Furthermore, performing an audit on the supply chain can often be a highly demanding process as it can take takes days to establish provenance of items. These auditing procedures are also often carried out with a lack of trust towards the auditee.

We investigate the area of 3D printing and its digital supply chain. In this industry, spare parts and customized tools can be remotely printed on demand. Licensing of 3D printed parts before they can be legally installed on aircraft requires access to an accurate and transparent history of these parts. These provenance records may include references to material used for the printing or the conditions under which they were printed.

This thesis introduces TPTS, a Trustworthy Process-Tracing System which leverages the potential of the Blockchain technology. It enables organizations to track and trace items throughout the supply chain in a reliable manner. A unique approach employing Attribute-Based Encryption is used to keep the transactional data confidential from unauthorized stakeholders. This privacy mechanism offers anonymity to the entities and supports a key escrow scheme enabling auditing authorities to have access to these data.

TPTS is based on the recently released Hyperledger Fabric platform which implements the Blockchain technology. The platform itself acts as a distributed database where records of all transactions are added through a consensus of nodes and kept in an append-only fashion. In this way TPTS offers tamper-resistance guarantees and ensures that stored data have not been altered since their initial addition to the ledger.

Acknowledgements

I would like to thank my advisors Dr. Holger Kinkelin, Dr. Heiko Niedermayer and Sree Harsha Totakura. Their exceedingly valuable input truly supported my work.

I would also like to thank Prof. D.-Ing. Georg Carle for his supervision and his insightful feedback.

Lastly, I would like to express my gratitude towards the Greek Foundation for Education and European Culture for their support throughout my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Topic

Supply chain management deals with the governance of all supply chain activities in order to increase profits as well as customer value and obtain an advantage over competing organizations. However, each organization stores information regarding transactions of the supply chain in their own private data silos. Shippers and other actors in the supply chain cannot access such valuable information in a timely manner. due to hindered visibility of such data. Additionally, the phenomenon of "mirroring" – the same documents being in the possession of multiple stakeholders in the supply chain – leads to unnecessary redundancies and further detracts from efficiency in the supply chain operations.

In parallel, 3D printing is an upcoming industry which is currently disrupting the classic supply chains. New products are digitally designed and can be produced on demand and just in time independently from the geographical location. The customer receives a digital file containing all the required printing data from the manufacturer and can print the part on the spot. In this process, today, people are using emails and USB sticks to transfer the intellectual property from one end to the other. This data distribution method is unreliable as well as error prone and often happens to print the wrong file on the wrong printer. It is also not traceable and has many risks for the manufacturer since she cannot control who and how many times has printed the part.

This digital supply chain with Additive Manufacturing – also known as 3D printing – enables optimized processes and reduction of logistics costs. For example. spare parts and customized tools can be remotely printed on demand. Licensing of 3D parts

before they can be legally installed on aircraft has a high need for reliably retrievable and transparent history. Furthermore, auditing can often be an unnecessarily long and opaque process, often carried out with a lack of trust towards the auditee. It is therefore evidently imperative to conceive a method that enables complete, trustable and persistent traceability of the orders and all the according manufacturing steps in 3D printing.

A new emerging technology called Blockchain has attracted attention in research and industry areas. The Blockchain can be regarded as a decentralized database that keeps public records in an append-only fashion. Absolutely no changes may be done on stored records. The main component of Blockchain is a distributed ledger that records all the transactions that take place in the network. This ledger is replicated across many network participants, each of whom contributes on its maintenance through a secure consensus mechanism.

Industrial efforts surrounding the supply chain provenance and its traceability while leveraging the Blockchain technology have gained traction recently [12, 1, 19] in areas such as the food and the shipping industry, with major companies driving these advances. However, a public ledger is inherently prohibitive for data which stakeholders wish to keep private from certain entities. The majority of current approaches in the area of Blockchain lack a privacy mechanism to satisfy requirements for confidentiality and anonymity.

1.2 Goals

The focus of this work is examining the feasibility and implementation of a novel system which supports secure reflection of the 3d Printing supply chain process into the Blockchain. This system should enable the following use cases:

- Track each part throughout the chain of the 3D Printing process from the point of creation of the part's model by the Original Equipment manufacturer, to the purchase of a number of prints by a client, to its final delivery to the client
- Trace the above chain back after its completion

The Blockchain technology is used to create a tamper resistant accounting ledger for all the transactions – file creation, printing request, quality information etc – generated during the manufacturing process, from the moment the file has been registered to the actual printing process. However, certain transactions and their details should be kept confidential and only accessible to stakeholders involved – directly or indirectly – in the

transaction as well as auditing authorities. For instance, a printshop should not be able to see what another printshop has printed and a customer should not be allowed to see the price another customer paid for a part.

The goals of this thesis are twofold:

1. The study of the state-of-the-art in the area of Blockchain technologies for the industrial application. Exploring a new approach for reflecting the supply chain transactions on a Blockchain.
2. The analysis, design and implementation of this approach, adapted specifically for the field of 3D Printing and enhanced with privacy features

1.3 Outline

The rest of this thesis is structured as follows:

Chapter 2 elaborates on necessary background knowledge that will facilitate the comprehension of topics analyzed further on in this thesis.

Chapter 3 analyzes the current state of supply chain management in combination with Additive Manufacturing and examines methods through which to incorporate the desired requirements into a Blockchain-based system.

Chapter 4 presents the design and the components of our Trustworthy Process-Tracing System (TPTS). The findings of Chapter 3 are applied to describe design principles satisfying the listed requirements.

An overview of the implementation of the derived system is presented in Chapter 5. This chapter analyzes the methods through which Hyperledger Fabric is leveraged, and describes the cryptographic implementation of the privacy mechanism of TPTS.

Chapter 6 presents the evaluation results based on the implementation from chapter 5. We include quantitative methods including performance and longevity measurements as well as qualitative comparisons of the core features of our system against other similar systems.

Chapter 7 compares the work of this thesis to related research efforts and indicates the novel contribution of TPTS.

Finally, Chapter 8 concludes this work and discusses the findings of this thesis.

Chapter 2

Background

This Chapter gives valuable background information on concepts that will be utilized further on in this thesis. An introduction is given to the Blockchain technology, Cryptography fundamentals – including Attribute-based Encryption – as well as the Hyperledger Fabric platform and its architecture.

2.1 Introduction to the Blockchain Technology

2.1.1 Basic Concepts

The first mention of a chain of blocks cryptographically linked and secured occured in 1991 [10]. A Blockchain is a continuously growing list of records called blocks, which are linked together and cryptographically secured. Each block includes:

- a link to the previous block in the form of a hash pointer,
- a timestamp,
- transaction data

The Blockchain is a distributed ledger able to store records of transactions executed between parties in a verifiable and permanent way. Typically managed by a peer-to-peer network, the blockchain adheres to a protocol for the validation of new blocks. Once stored on the Blockchain, these blocks and their transactions cannot be edited without also altering all subsequent blocks, which would lead to conflict with the other network peers and the overall consensus.

The Blockchain is therefore a novel approach to aggregating data between parties who don't trust each other but are able to trust the underlying technology behind the Blockchain, while also preserving the qualities of a regular distributed and decentralized database. Concerning the architecture of a Blockchain as well as the protocols followed to append transactions to the ledger, each platform implements its own consensus mechanism.

2.1.2 Consensus Mechanisms

It is appropriate at this point to describe the concept of consensus in the context of the Blockchain. A core issue of distributed computing is how to reach overall system reliability. The process of getting participants of a Blockchain network to agree on a single data value, often in the presence of faults, is termed *consensus*. Whenever a new transaction is broadcast to the network, every node has the option to include that transaction to their copy of their ledger or to ignore it. When the majority of nodes which comprise the network decide on a single state, consensus is achieved.

In 1982, a generalized version of the "Two Generals Problem" was published [15], called "The Byzantine Generals Problem". The characteristic which defines a system that tolerates the class of failures that belong to the Byzantine Generals Problem is termed Byzantine Fault Tolerance (BFT). Byzantine Faults are the most severe and difficult to deal with. This is of high importance, since Blockchains are decentralized and not controlled by a central authority. Malicious users have high incentives to attempt to trigger faults and without the presence of a consensus mechanism such as BFT a peer would be able to execute false and invalid transactions, effectively negating the Blockchain's purpose and reliability. Therefore the consensus mechanism chosen by each platform must achieve Byzantine Fault Tolerance through its design.

There are a number of methods which are used for reaching consensus in a Blockchain. A multitude of new Blockchain technologies have come up with novel consensus techniques depending on the intent and sought for benefits. Two of the most common ones are examined below.

Proof-of-Work

In Proof-of-Work (PoW), in order for an actor to be elected as a leader and choose the next block to be added to the Blockchain they have to find a solution to a particular mathematical problem. Probabilistically speaking, the actor who is first to solve this problem is normally the one who has access to the most computing power. These actors

are also called miners. The miner of a block gets rewarded with some currency as a motive to keep mining. Other nodes verify the validity of the block before adding it to their copy of the Blockchain.

(Delegate) Proof-of-Stake

Proof of Stake (PoS) is the most common alternative to PoW. In PoS the weight of each validator's vote depends on the size of their deposit (i.e. stake). Of course, if the next block's selection was done purely by account balance then a user holding the majority of the total deposits would essentially have permanent control, something which understandably would be undesirable. For this purpose, a variety of different combinations of methods have been devised, such as selection based on coin age, as well as a Delegated Proof of Stake variant (DPoS). In DPoS, coin holders select their so-called "delegates", who are then responsible for validating transactions and maintaining the Blockchain.

2.2 Hyperledger Fabric

Hyperledger Fabric (HLF) [3] is a permissioned blockchain platform and it is one of the projects supported by the Hyperledger Framework. Contrary to open Blockchains, in permissioned Blockchains there is an additional access permission layer that controls which entity is allowed to interact with the blockchain. Permissioned Blockchains have proven to be a viable alternative to open Blockchains, particularly in the world of business where higher requirements of security and privacy are observed. Additionally, permissioned ledgers offer better overall scalability in terms of throughput, as BFT consensus mechanisms are better applicable in networks with a limited number of nodes ([20], Table 1).

Within the HLF network, actors can create and manage digitized assets by invoking transactions. These transactions are processed by chaincodes (or smart contracts), which is the piece of code that is installed and instantiated onto the HLF Peers. These chaincodes serve as a business logic layer between the user and the shared ledger. The block structure of HLF can be seen in Fig. 2.1 in shortened form to include the most important fields. HLF supports pluggable implementations of different components, including different consensus mechanisms. Chaincodes can be developed in multiple languages, peers can be deployed in different layouts, and different storage mechanisms can be leveraged.

One such component offered by HLF is a Membership Service Provider (MSP) that offers an abstraction of a membership operation architecture. An MSP takes care of all necessary background mechanisms for issuing and validating certificates as well as user authentication. Each MSP has to be associated with one or more CA's and each actor interacting with an HLF network can be represented by a distinct MSP.

In the rest of this section further underlying technical details of HLF are examined.

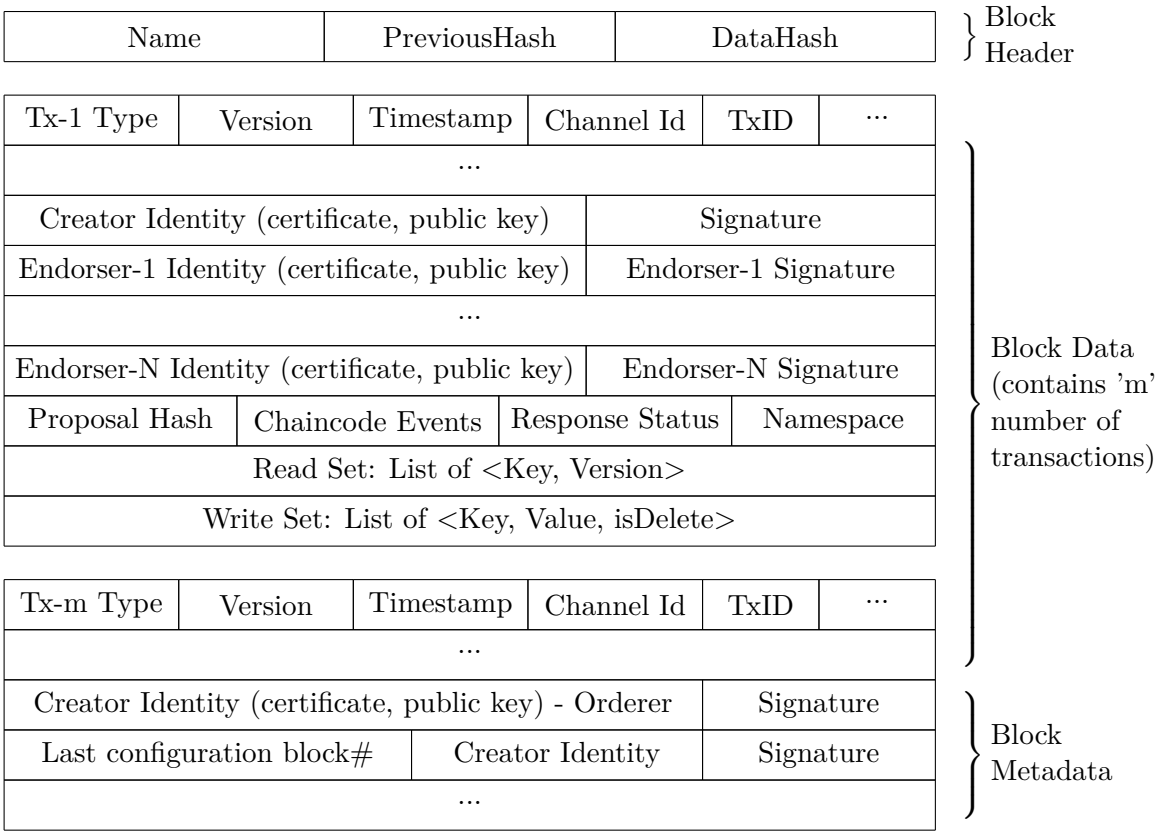


Figure 2.1: Hyperledger Fabric block structure

2.2.1 Architecture

There exist three main components in a HLF network:

1. Peers
2. Orderers
3. Clients

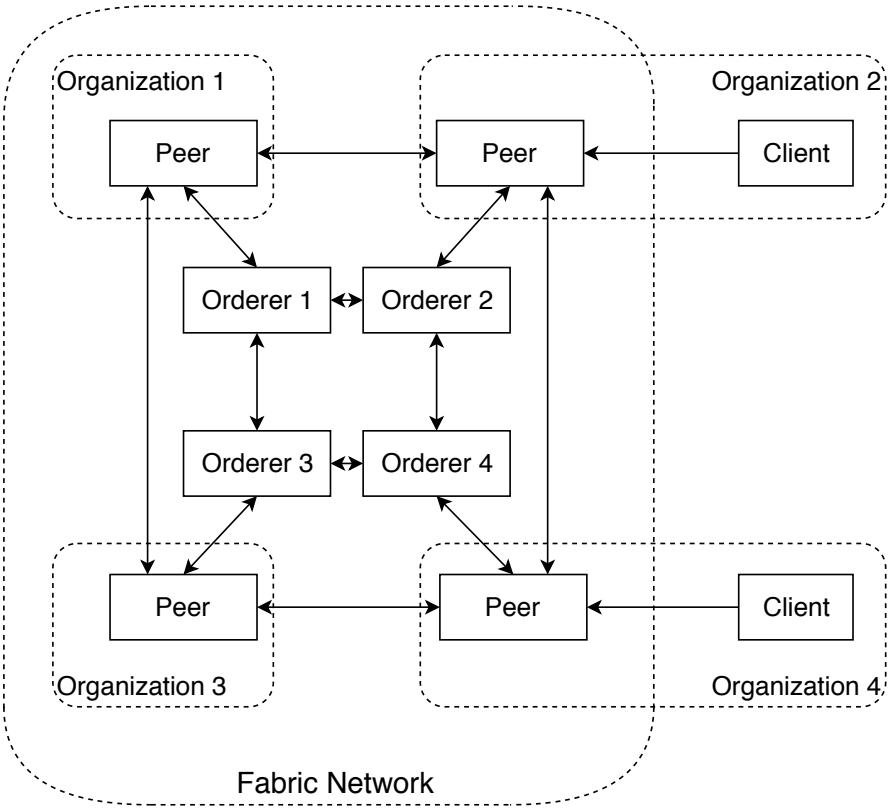


Figure 2.2: A simple Hyperledger Fabric network layout

In more detail, peers are nodes which commit transactions and maintain the state of the ledger, as well as a copy of it. Peers can additionally have a special role termed as *endorser*. Orderers on the other hand are nodes operating the communication service which implements a delivery guarantee. The role of the endorsing peers and orderers is explained below in subsection 2.2.4. Lastly, clients submit a transaction or invocation to the endorsers, and broadcast the received transaction proposals to the ordering service. A sample HLF layout of nodes can be seen in Fig. 2.2.

2.2.2 Ordering Service

In HLF, consensus algorithms defining how the ordering service operates can be plugged at will. As of May 2018, HLF provides two distinct options for setting up an ordering service, although of course users of the platform are free to implement and plug their own consensus mechanisms.

1. Solo
2. Kafka-based

The first approach utilizing a solo orderer is not intended to be used for production and is included strictly to facilitate testing. With this method there exists a single process which serves all clients which means that technically there is no consensus required. Naturally there are no features of high availability or even scalability present, as this approach is not fault tolerant in any form.

The Kafka-based ordering service on the other hand provides ordering procedures in a crash fault tolerant manner. This service consists of a Kafka cluster with its corresponding ZooKeeper ensemble as well as a set of ordering service nodes that stand between the clients of the ordering service and the Kafka cluster. The ordering nodes only communicate through the Kafka cluster and not directly with each other. While this ordering service is crash tolerant, it does not provide Byzantine Fault Tolerance.

A Byzantine Fault Tolerant ordering service was previously in development but was dropped before the 1.0 release of HLF. However an implementation of a new BFT service is on the roadmap of the HLF project with an unknown release date.

2.2.3 Endorsement Policies

Each chaincode is instantiated with a specific endorsement policy on endorsing peers. Before a peer adds a block to its ledger, it validates the block's transactions. One part of this validation process is to verify that the so called *endorsement policy* is fulfilled. These endorsement policies essentially dictate which peers need to agree on the results of a transaction before it can be added to the ledger.

Two components form these endorsement policies: a principal and a threshold gate. Principal refers to the entity which is doing the endorsement. A threshold gate on the other hand is comprised of an integer t and a list of n principals or other threshold gates. The gate itself dictates that the policy is fulfilled if and only if out of the n principals or gates t must be satisfied. For instance, an endorsement policy can have the form: *AND('Org1.member', 'Org2.member', 'Org3.member')*. This would require one endorsement from a member peer of each organization among Org1, Org2 and Org3.

2.2.4 Transaction Flow

HLF uses a novel approach for handling transactions and their subsequent storage onto the Blockchain Network. This sequence can be viewed in Fig. 2.3. Events are executed as follows:

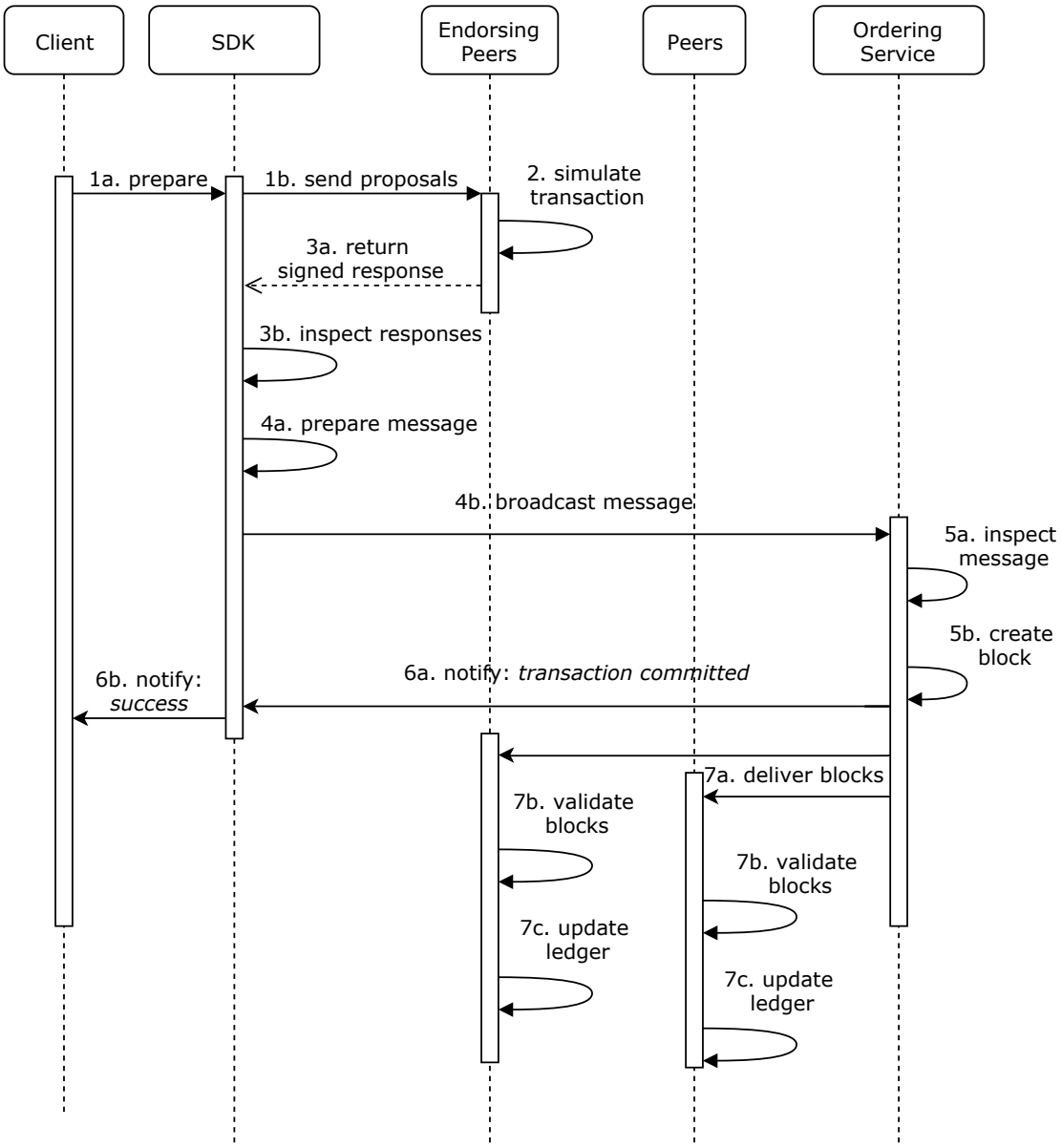


Figure 2.3: Transaction flow in Hyperledger Fabric

1. A client initiates a transaction by contacting one or more peers
2. Endorsing peers verify the signature of the client and execute the appropriate chaincode against the current state of the ledger. No updates are made to the ledger yet
3. The signed proposal responses are sent to the client for inspection in order to verify that the endorsement policy is fulfilled

4. The client assembles endorsements into a transaction and transmits it to the ordering service
5. The transaction is validated and added to one of the next blocks to be committed
6. The client is notified that the transaction has been commited by the orderers.
7. Peers receive the new blocks, validate the data and update the ledger.

The reason behind this novel design according to the developers of HLF is to promote scalability since transactions can be executed before they are put in order. This enables parallel execution of transactions, which can improve throughput. Therefore since the two processes of endorsement and ordering are each carried out by a different set of peers, the system may scale better than if each node had both responsibilities.

2.2.5 Fabric Certificate Authority

The Fabric Certificate Authority (FCA) plays a role for both authentication and MSP certificate generation. Through the enroll API Fabric CA issues enrollment certificates and populates the appropriate MSP directories. HLF can operate either with an external CA or with FCA as a root or intermediate CA. The FCA server can act as its own registry or it can communicate with an LDAP server as the user registry. Interaction with a FCA server can be done either through the FCA client or one of the Fabric SDKs.

2.3 Cryptography

Due to privacy and anonymity requirements explained in the following chapters, cryptographic techniques must be leveraged in this work. This section provides some necessary background in the area of cryptography as well as Attribute-Based Encryption.

2.3.1 Basic Principles of modern Cryptography

Encryption

Encryption is the process of converting data into an unreadable form. This protects the privacy of our data, while transmitting data from one end to another. The inverse process is termed decryption. On the receiver's end, the data is decrypted and the original form is restored. These two processes require an additional piece of information to function. This information is known as a cryptographic key. The following sections

explain how some types of encryption and decryption utilize a single key, while others may require different keys for each process.

Message Authentication

The second major principle in cryptography is Message Authentication. In short, authentication ensures that the message originated from the entity claimed in the message. For instance, suppose that Alice sends a message to Bob and Bob wants to verify that the message has been indeed sent by Alice. This can be made possible if Alice performs some action on message that Bob knows only Alice can do.

Integrity

One of the problems that a communication system can face is the loss of integrity of messages being sent from sender to receiver. Cryptography can ensure that a received message has not been altered anywhere on the communication path. This is normally achieved by using the concept of cryptographic hash [18].

2.3.2 Symmetric and Asymmetric Encryption

In symmetric encryption a single key is used for encrypting and decrypting the data. This key is shared between the parties communicating with each other. In asymmetric encryption, also known as public key cryptography, each user holds two related keys to deal with encryption and decryption. In symmetric Encryption, anyone with knowledge of the shared can decrypt the messages. For this reason, asymmetric encryption uses two related keys for leveraging security in the following way: a *public key* is made freely available to anyone who might want to send a message to user A. The second key, termed as *private key*, is kept secret and accessible only to user A.

A message directed to user A is encrypted using his public key and can only be decrypted using his private key. Similarly, a message encrypted using a private key can be decrypted using a public key (a technique used in digital signatures). Asymmetric encryption offers higher security for the information transmitted during communication, however it is orders of magnitude slower than symmetric encryption.

A hybrid approach is often used, where a random symmetric key is created to encrypt the message and subsequently this symmetric key is encrypted using the public key of the user who will receive the encrypted data. This allows us to leverage the security offered by asymmetric cryptography, while still keeping the encryption and decryption time of the message to the low levels that symmetric cryptography allows.

2.3.3 Attribute-Based Encryption: Ciphertext-Policy and Key-Policy

Attribute-Based Encryption (ABE) is a unique encryption scheme in which the private keys of users as well as the ciphertexts are dependent upon attributes. Messages can be encrypted with respect to subsets of attributes (Key-Policy ABE) or access policies defined over a set of attributes (Ciphertext-Policy ABE).

In *Ciphertext-Policy ABE* (CP-ABE)[6], each user has a private key which contains a set of attributes. Each ciphertext specifies an access policy that has to be satisfied for decryption to be possible. A user decrypts the ciphertext if and only if his attributes satisfy the policy of the respective ciphertext. Policies may contain conjunctions, disjunctions as well as threshold gates (e.g. k out of n attributes have to be present in a key). For instance, let us assume the scenario where the set of the globally allowed attributes is {A,B,C,D,E}, user 1 receives a key to attributes {A,B} and user 2 to attributes {C,E}. Let the access policy be $(A \cap B) \cup D$. Then user 1 will be able to successfully decrypt the ciphertext, while user 2 will not be able decrypt the data.

In the above manner, CP-ABE enables us to introduce implicit authorization, as the encrypted data are effectively infused with authorization policies. Only entities with a combination of attributes which satisfy the associated policy can decrypt these data. Another benefit of CP-ABE is that keys created after the encryption is done are still able to decrypt these data, assuming of course the respective keys hold the right attributes. Therefore, data can be encrypted without explicit knowledge of the actual set of users that will be able to decrypt, but solely specifying the policy which allows decryption to occur. Users who are given a key in the future with respect to attributes such that the policy can be satisfied will then be able to decrypt the data.

Key-Policy (KP-ABE)[9] on the other hand is essentially the dual problem to CP-ABE. An access policy is encoded into a user's secret key (e.g. $(A \cap B) \cup D$) and each ciphertext is computed with respect to a set of attributes (e.g., {A,C}). In this example the user would not be able to decrypt this ciphertext but would for instance be able to decrypt a ciphertext with respect to {A,D}.

One important property which has to be achieved by both CP-ABE and KP-ABE is termed *collusion resistance*. Essentially this means that it should not be possible for distinct users to combine their secret keys in such a way that they could together "collect" the necessary attributes to decrypt a ciphertext that neither of them could decrypt on their own.

Chapter 3

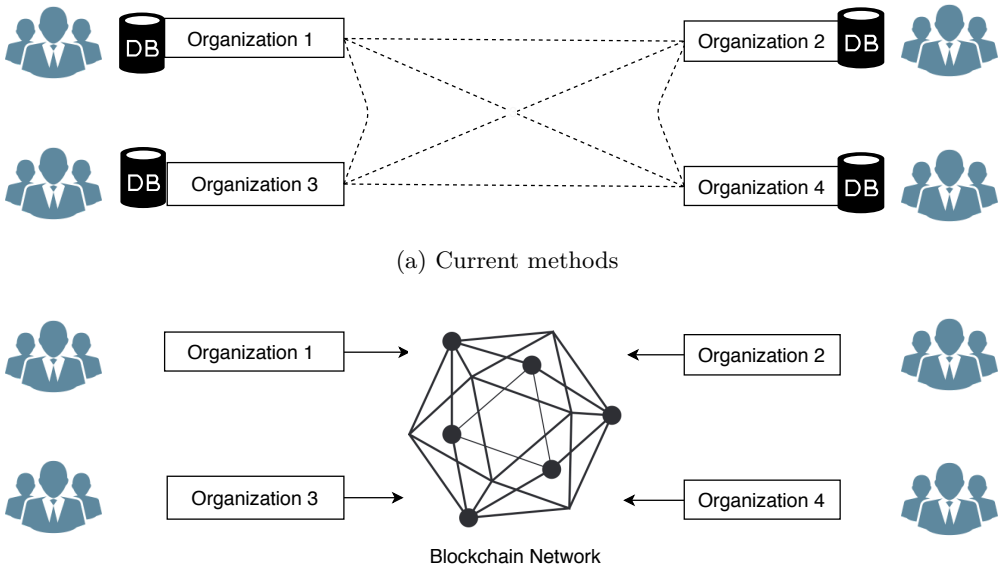
Analysis

This Chapter examines the core problem which we are looking to tackle through this work. We analyze the requirements the system should fulfill and discuss different approaches for accomplishing these goals.

3.1 Problem Statement

Supply chain management is an essential part of businesses and is crucial not only for the benefit of the companies involved, but for customers too. Supply chain entities can be categorized based on their ownership stake in the product. Entities which own the goods at one or more specific stages of the supply chain are considered stakeholders. These stakeholders of the supply chain can be roughly enumerated into the following groups: the retailers, distributors, manufacturers, suppliers, shippers and finally the end users or consumers. Besides the customers, all other groups of stakeholders each represent their own distinct organizations in the majority of cases.

Each of these organizations stores data regarding their transactions with the other actors of the supply chain. However, these data are often spread throughout multiple databases belonging to each organization, with a noticeable overlap of information and without effective collaboration present between different actors of the supply chain. Geting access to another stakeholder's data involves long and arduous procedures and a lot of paperwork. The process of sharing data can highly contribute to the effectiveness and performance of organizations participating in a supply chain, depending on the level and quality of this shared information [14].



(a) Current methods

(b) The envisioned system

Figure 3.1: Collaboration between organizations and stakeholders in the industrial supply chain

Fig. 3.1a presents the current situation. Actors within the supply chain collaborate in the absence of effective data sharing, since most information is kept in private data silos. Fig. 3.1b presents the envisioned methods regarding how data should be shared and accessed. More specifically, all data which concerns more than one parties of the supply chain are stored in a system which can be accessed by all interested entities. This chapter examines a number of additional features which we aim to infuse this system with.

It should be emphasized, that this lack of collaborative processes for data is a common issue among many industrial areas. However, in the following sections and for the remainder of this work, the concrete case study of Additive Manufacturing (AM) in combination with the need for supply chain traceability is analyzed.

3D printing, or AM, is an industry that has been gaining a lot of traction the past few years, with billions of investments in the aerospace and medical industries from organizations such as Airbus, Rolls Royce and NASA. These 3D printed parts are designed to be placed for instance on aircraft and satellites or to create low-cost medical prosthetics, implants and organs. Such parts offer better strength-to-weight ratio as well as lower manufacturing costs while still able to follow strict production specifications and standards.

Additionally, parts can be printed remotely and on demand on a global scale. This means that someone can order a part which was designed in one country, sent digitally to a printshop in another country where it is printed and then shipped to a third country. It is evident then that it is not trivial to keep track of this supply chain. Furthermore, the processes which are followed with regards to aggregating and reporting the printing data are often:

1. Inefficient and insecure, since files of 3D models are often carried by hand to the printer machine through USB sticks.
2. Unverifiable, as operators of printer machines often have to manually write down reports of observed parameters during the printing operation in pen and paper.
3. Lacking authenticity and accountability, since there are often no secure ways of knowing who carried out a process.
4. Error-prone, due to laborious and manual record keeping

Since such 3D printed parts end up in aircraft or medical devices, tracing a part's origins and tracking history is essential for the purposes of audits, recalling faulty parts or licensing. In order for a part to be installed on aircraft it has to go through a tedious procedure of licensing that covers verifying the material's properties and manufacturing process, especially since there are human lives at stake. However, based on the lacking nature of the followed approaches as explained above, we believe that the digital processes for the purpose of asset traceability are currently inadequate, particularly from a reliability and tamper-resistance perspective. The claim which this works investigates is that the Blockchain Technology can solve the above problems.

In AM, material is joined or solidified under computer control in order to create a three-dimensional object, with material being added together (for instance, liquid molecules or powder grains fused together). The three-dimensional item is built from a digital design, usually by successively adding material, layer by layer. These 3D objects can be of almost any shape or geometry, normally produced based on data from a digital 3D model or another data source such as an Additive Manufacturing File. One of the most common file types that is used for 3D Printing is Stereolithography (STL).

There exist three general principles in the area of AM. These are:

1. Modeling: 3D Printable models can be created with a computer–aided design (CAD) package, via a 3D scanner, or by a plain digital camera and photogrammetry software. When a CAD is used, a reduced rate of errors can be expected and verification of the design is possible before printing.

2. Printing: Before a 3D model is printed from an STL file, it must be examined for errors and possibly enter a repair procedure. 3D Printing of a model with modern methods can take anywhere from a few hours to a few days, depending on the type of machine used and the size and number of models produced in parallel.
3. Finishing: After the printing is complete, certain post-processing techniques are possible. For instance, in certain occasions printing a desired object in standard resolution and subsequently removing material using a higher-resolution subtractive process can achieve greater precision. Painting the object is also a possibility.

The following sections and chapters exhibit how the above distinct phases map to steps in the supply chain and how these phases are logged in the Trustworthy Process-Tracing System (TPTS).

3.2 Reliable Asset Tracking in the Supply Chain

As explained above, the main goal in our work is to reflect the individual steps of the supply chain on the Blockchain. This enables a multitude of scenarios to take place in the area of AM.

3.2.1 Use Cases

Any entity interested in retrieving the data concerning a 3D printed part can be considered as a regular use case for our system. Nevertheless, the two major use cases where we identify the additional value of tamper-resistance features are explained below. Each of the two scenarios contain smaller individual use cases that TPTS should support, such as registering a model or storing the printing parameters of a 3D printed part.

Use Case 1: Model Lookup

Let us assume a defect has been identified in a number of 3D printed parts and the source of the issue has been found to be the printing process. An operator working for an Original Equipment manufacturer (OEM) which designs 3D models wants to know

which printers have executed prints of a model [1]. The OEM operator should be able to retrieve all the 3D printed asset data, including the information about the printshop and individual printer the parts were printed on. These data could include information such as laser intensity and oxygen degrees or any other parameter. If any of these parameters deviated at all from certain standard values, the quality of the end-result could be compromised. By examining these values and identifying potential anomalies the OEM could then cooperate with the responsible printshops to amend the situation, or in extreme cases possibly even follow legal action in case of intentional and deliberate manipulation of the printer configuration. Additionally, data should be readily available at all times so that an OEM could retrieve this information immediately. This means that data replication is a major requirement and that a centralized database is not an acceptable system.

Use Case 2: Serial Number Lookup

Let us assume an in-flight malfunction has occured on an aircraft which forced an emergency landing. An official authority is carrying out an investigation to determine the factors that led to it. Experts have reached the conclusion that one of the 3D parts which were installed on the aircraft was the culprit which caused the eventual failure. They would therefore like to know if the fault lies with the OEM (in the case of a faulty model), or with the printshop (in case the standards that were supposed to be followed during the printing process were not followed). The auditors should be able to query for the serial number of the part and verify both the model file which was used to produce the part, as well as the conditions under which the part was printed. That information should be acquired in a manner that provides some data-integrity guarantees. In this way, the auditors could be convinced that the data which were retrieved are accurately reflecting the real model file which was used as well as the real printing conditions. Therefore, the possibility of malicious data alteration should be prevented once more.

3.2.2 Leveraging the Blockchain Technology for Asset Traceability

Having analyzed the features and architecture of the Blockchain Technology in section 2.1, its advantages compared to a traditional database system might not be immediately

[1] In order to clarify the terminology, when using the term *model* we refer to the model file (e.g. .cad file). On the other hand with the term *part* we refer to the actual physical object that was created through the 3D printing process that used the model file as a source.

evident to the reader. In use cases where trust and robustness are not an issue, there is nothing a Blockchain can do that a regular database cannot. Particularly in our case however, a number of parties that do not necessarily trust each other (OEM's, customers, printshops, Suppliers) must cooperate in order to create and maintain a reliable transaction history. Blockchain transactions contain their own proof of validity and their own proof of authorization, instead of requiring some centralized application logic to enforce those constraints. A Byzantine Fault Tolerant Blockchain platform is required for this purpose, as it guarantees the correct operation of the system even if a number of competing entities are acting maliciously. The actors of a Blockchain system can therefore trust the underlying technology instead of the other actors interacting with the system. An auditor or any other interested party can be assured that the transactions happened as they appear on the Blockchain and that the transactional data could not have possibly been modified since they were added. Additionally, the cost and time necessary to conduct an audit itself would decline considerably. This would occur because the transaction layout would be standardized between the different manufacturers of 3D parts as well as the 3D printshops. For these reasons a system based on the Blockchain technology provides the necessary architecture for reliable and tamper-resistant data storage that are required for this use case.

A simple abstracted view of how Blockchain is involved for tracing assets throughout the supply chain can be seen in Fig. 3.2. This flow chart depicts the timeline between the creation of the item and the consumption of this item by the customer. The main benefits which can be gained by utilizing the Blockchain technology for the purpose of tracking assets throughout the supply chain are its reliable and tamper-resistant storage, combined with the fact that the involved parties can collaborate even through a lack of trust.

Based on the two main use cases which were included in section 3.2, we identify a number of requirements that our application has to meet:

R1 The system must offer reliable and tamper-resistant traceability of an item throughout the supply chain

R2 The chosen platform on which the system will be based should provide Byzantine Fault Tolerance as well as highly available access to the data

3.2.3 Evaluation of available Blockchain Platforms

One of the first decisions to make is deciding which Blockchain platform this work will be based on. The different Blockchain technologies are assessed in an attempt to choose

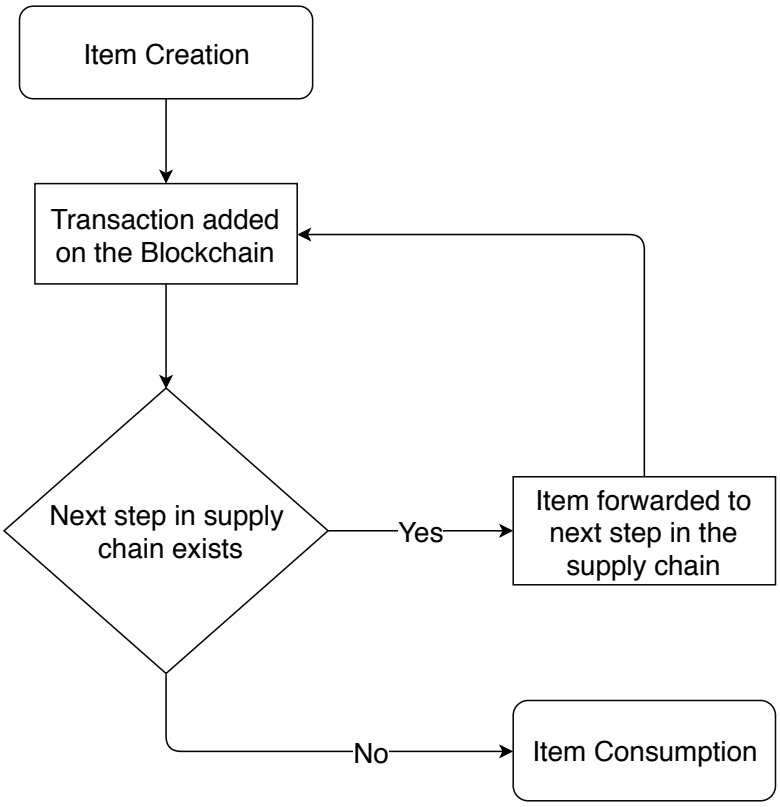


Figure 3.2: Item lifecycle and supply chain traceability on the Blockchain

one that is expected to best fit the use case. These are evaluated based on the following criteria:

- An open-source platform should be selected
- Maturity and support must be sufficient to fulfill the stated requirements
- The platform should operate under an efficient and secure Byzantine Fault Tolerant consensus protocol
- Decentralization should be a strong focus
- The platform should be able to maintain cryptographically secure and tamper-proof history of transactions
- The architecture should allow us to plug in our own privacy mechanism
- The platform should be flexible, modular and fit to the use case
- Presence of cryptocurrency is unimportant, but invoking transactions should not impose a charge

In section 2.1 the idea behind *consensus mechanisms* was explained. In our specific use case however, Blockchain platforms that utilize a Proof-of-Work or Proof-of-Stake consensus can already be filtered out. Proof-of-Work requires participating users to solve difficult mathematical problems to validate and authenticate transactions. This means that it is a highly energy demanding approach, while also enabling an adversary to attack our network as long as he controls more than 50% of the total computing power of the network. When utilizing Proof-of-Stake on the other hand (as well as Delegated Proof-of-Stake), the creator of the next block is chosen in a deterministic (pseudo-random) way, and the chance that an account is chosen depends on its wealth. Since cryptocurrencies or any other type of token will not be incorporated in this work, the PoS consensus is deemed inappropriate. Other consensus techniques are clearly a better design decision, all other factors considered.

Based on preliminary findings, two main platforms that meet the above criteria have been identified:

- Hyperledger Fabric (HLF): a modular approach directed to enterprises supporting confidential channels within the Blockchain. This is a platform which was only recently released in August 2017, therefore its maturity is not the highest between the two candidates. However, its features for asset and user management are promising and very fitting to our use case.
- Ethereum quorum: a permissioned implementation of the public Ethereum platform supporting data privacy. It is a more mature platform compared to HLF, however the permission management for the network is not as configurable as in HLF. More importantly, quorum nodes who are not authorized to access the transactions don't keep these records at all in their storage. This of course is in not in favor of requirements for data consistency and reliability. It also means that every stakeholder must own an active node which participates in the network. This does not align with goals for designing a system which is easy to join and use without the necessity to contribute one's hardware to the network.

We have opted to work with Hyperledger Fabric going forward. More details about its benefits are explained further in Chapter 5.

3.3 Data Management

3.3.1 Privacy and Anonymity

One of the features of a traditional Blockchain system is that everyone with access to the network can read everything. While in many cases this is a sought-after feature, in TPTS some of the data that we are planning to store on the Blockchain should only be visible to certain entities: the parties directly involved in the transaction and potential auditors. Particularly, we wish for auditing authorities that join the system at a specific point in time to be able to view private data of past as well as future transactions.

An organization will naturally not want their competitors to know the details regarding which part they have sold, their total income or the observed trends concerning which types of parts seem to be in high demand. It would also be unacceptable for a company to be allowed to view what purchases their customer has done from another competitor. Lastly, a customer should be prohibited from viewing the printing configuration another customer used for a specific part. Conceiving such configuration is often a very fine tuned and demanding process, therefore replicating it or copying it for free would be unwelcome.

Similar efforts [21, 8, 13] have proposed a number of solutions examined from various perspectives and we have to devise the approach that best suits this work.

Five distinct categories of actors can be identified in the system:

- Customers
- OEMs
- Printshops
- Shipping agencies
- Auditors

As a general rule, the only entities that should be able to read the data of a transaction should be those directly involved in the real world transaction, as well as assigned auditing authorities. This means that besides auditors, only the specific OEM, customer, printshop and shipping agency involved should be able to view the data that concerns a specific printed part. In parallel with privacy, anonymity is also required in the system, or at the very least pseudonymity; that is, entities should not be able to tell *who* specifically is involved in a transaction they did not participate in.

Approaches

For the purposes of privacy a number of different approaches are examined:

1. PGP-based: The intended data is encrypted with a random symmetric key and then this symmetric key is iteratively encrypted with the public key of each recipient.

2. HLF Channels: HLF also contains its own confidentiality mechanism in the form of the so-called SideDB feature which is currently in experimental phase [1]. When using this technique, evidence of each transaction – in the form of a hash – is exposed to the entire channel, while the actual data of the transaction itself is kept private in specific peers, according to a defined policy.

3. Attribute Based Encryption: An access policy can be specified for each transaction regarding which entities in our system should be able to decrypt the data. This approach can even enable users that haven't joined the system yet – and thus do not have any cryptographic keys yet – to decrypt the data in the future.

The PGP-based approach would likely require the least additional complexity in terms of the encryption process, however we lose the ability to encrypt data for users that have not joined the system at the time of encryption. This is an essential feature for the system, particularly in order to allow auditors joining the system in the future to be able to view past transactions. Therefore this approach is unable to meet key requirements.

Additionally, the size of transactions would scale with the number of recipients which is a particularly major disadvantage, especially if at a future point we would like to extend our system to include more categories of actors involved. As explained in subsection 3.3.4 the goal is to reduce as much as possible the data inserted in each transaction.

Regarding the SideDB feature of HLF, its goal is to expose solely the evidence of each transaction to the chain, ordering service, and channel peers while the data itself is distributed to peers based on a desired privacy policy. Utilizing this approach would essentially mean working with a privacy mechanism very similar to what Ethereum Quorum provides, a platform which we already decided not to use. In TPTS the goal is for the policy to authorize data access from specific users, not specific network nodes. Additionally, this feature is still in experimental status at the time of writing, therefore it is not fit for usage.

[1] https://jira.hyperledger.org/browse/FAB-8718

The case of Attribute Based Encryption (ABE) fits well into TPTS. There exists a plethora of available schemes in the area of ABE, each with a different approach and benefits. Additionally, attributes supporting date of key creation are desired so that policies can also be adjusted to include date comparisons. For instance, a policy such as **(USER32 and CREATION_DATE > 201612)** would prevent the entity owning a key with attributes [USER32, 201601] from decrypting the ciphertext protected by this policy.

Regarding the selection between Key-Policy ABE (KP-ABE) and Ciphertext-Policy ABE (CP-ABE), we remark that in KP-ABE the encryptor does not have direct control over who will be able to decrypt the data and must trust the authority distributing the keys to issue the appropriate keys to grant or deny access to the appropriate users. In TPTS the aim is to allow the entities themselves to possess the authority to grant permissions according to their discretion. Therefore, the primary approach for cryptographic needs in TPTS employs CP-ABE.

In this section the following requirements have been defined:

R3 The stored data must be confidential. Only users authorized through a specified access policy should be given access to specific data.

R4 The system should support a key escrow scheme for auditing authorities to be able to view private data. New auditors joining the system should additionally be able to view private transactions executed in the past

R5 The stored data must preserve anonymity for the users. Users should not be able to have knowledge of which entities were involved in a transaction they are not authorized to access

Distribution of Cryptographic Keys

When including privacy features in TPTS it is reasonable to expect cryptographic keys to be required for encryption and decryption operations. The system should incorporate a secure mechanism through which users who join the system are able to receive any necessary cryptographic material that allows them to interact with the network and properly decrypt or encrypt the transactional data. This process can occur both when the user joins the system for the first time as well as in cases where new keys need to be provided to them to replace old ones.

Transmitting these keys over unencrypted e-mail to the appropriate recipients is expectably deemed as a very insecure approach and is not considered acceptable. One

valid method of distributing keys could be setting up a system for each key authority which will carry out this task of distributing keys, however that evidently implies additional user management costs for each such authority. Another way would be to physically distribute the keys in secure devices, although that would incur large overhead in key delivery times and costs. Additionally this approach might be not feasible when dealing with large distances between the user and the distributor due to risks of key compromise. A third method would be to utilize a central authority which, solely for user identification purposes, can be trusted by all parties. This CA will hold identification details mapping users to their personal public keys. The authorities wishing to distribute the private keys can use these personal public keys to securely encrypt and transfer the privacy-related keys directly to the users in a secure manner. Using this method however, it would still be unwise to use the Blockchain network to store these encrypted private keys for users to retrieve. Ultimately one should not forget that records cannot be removed or altered on the Blockchain, therefore it would be unfit to store private keys on the ledger, even when these keys are encrypted prior to storage.

Thus the following requirement is defined:

R6 The system should offer a mechanism which allows secure distribution of cryptographic material to all users

3.3.2 Verifiability and Data Validation

There is additionally a necessity for a mechanism that validates data stored on the system. For example, an OEM should not be able to add a confirmed transaction that claims they sold a part to a customer, if that customer did not actually purchase such a part from that OEM. This mechanism should require verification from multiple, if not all the parties involved in a transaction and should additionally respect the privacy requirements expressed in subsection 3.3.1.

The next requirement is therefore:

R7 The transactional payload must be verified by a certain mechanism before being considered valid

3.3.3 Accessing Data

Users belonging to organizations which are part of the system should have one or more means of viewing the transactions which have been stored on the Blockchain network. For this purpose there should be a user-friendly interface for retrieving the transactions

a user is interested in, based on a selected identifier such as a serial number. This retrieval mechanism should naturally respect the privacy requirements established in subsection 3.3.1.

The next requirement then is:

R8 Users and organizations must be able to query for an item based on a selected identifier and retrieve all data related to this item

3.3.4 System Longevity Analysis

All the data stored in the Blockchain has to be available at all times and on all – or almost all – the network nodes. This means that data accumulated over many years can account for a considerable amount of total data volume. The system should therefore be feasibly maintainable throughout a large period of time. We set this period to fifty years minimum for the purposes of the evaluation process. The number of fields within the transactions as well as the amount of data stored in each of these fields has to be closely examined and optimized in order to achieve the envisioned longevity for our Blockchain network's operations. The selected system should also be able to handle that amount of transactional data without noticeable effects in its performance concerning responding to invocations and queries.

Another detail to consider regarding this topic is whether it would be preferential to keep the actual bulk of data on a traditional database system and only store references as well as a checksum of these data for the purposes of data integrity verification. The goal in this scenario would be to incur considerably lower storage requirements on the Blockchain network peers, which might provide better performance and longevity. Evidently this would have potential consequences in terms of data availability and integrity concerns, since TPTS would depend on an external system to access the data. It should be investigated if this could possibly be deemed more appropriate than storing everything on the Blockchain, by thoroughly evaluating the benefits and disadvantages of either approach.

R9 The system should be tuned in a way which allows it to operate over a number of decades without overwhelming its resources or exhibiting performance loss, while in parallel maximizing the amount of valuable data it can hold.

3.4 Summary of Requirements

To summarize, we reiterate the identified requirements:

R1 The system must offer reliable and tamper-resistant traceability of an item throughout the supply chain

R2 The chosen platform on which the system will be based should provide Byzantine Fault Tolerance as well as highly available access to the data

R3 The stored data must be confidential. Only users authorized through a specified access policy should be given access to specific data.

R4 The system should support a key escrow scheme for auditing authorities to be able to view private data. New auditors joining the system should additionally be able to view private transactions executed in the past

R5 The stored data must preserve anonymity for the users. Users should not be able to have knowledge of which entities were involved in a transaction they are not authorized to access

R6 The system should offer a mechanism which allows secure distribution of cryptographic material to all users

R7 The transactional data must be processed by an adequate verification mechanism before being considered valid

R8 Users and organizations must be able to query for an item based on a selected identifier and retrieve all data related to this item

R9 The system should be tuned in a way which allows it to operate over a number of decades without overwhelming its resources or exhibiting performance loss, while in parallel maximizing the amount of valuable data it can hold.

Chapter 4

Design

4.1 Overview

Having declared a series of requirements that the Trustworthy Process-Tracing System (TPTS) should fulfill, this chapter presents the design of this system. The Blockchain network itself is a major component in this regard, therefore its iternal design is also largely affecting the other components in TPTS. We direct the reader to view the design of Hyperledger Fabric and its underlying technology in Chapter 2.2.

The main focus of this chapter is illustrating the processes which are followed with respect to interacting with the Blockchain network as well as the methods for data exchange through these processes. We begin by analyzing decisions taken for integrating the required privacy features and continue with the conceptual description of the protocol for the transactions as well as exhibit how the data model is formed. We subsequently discuss how users are managed in TPTS and how the cryptographic material which they will need is transmitted to them.

4.2 Reflecting the Supply Chain on the Blockchain

The system design reflects five basic steps of the supply chain that a 3D part goes through. These are namely:

1. Registration of the model by the OEM.
2. Part ordering by a customer

3. Printing by a service provider (printshop)
4. Post-processing by a service provider (optional)
5. Shipping of the part to the customer by a shipping agency

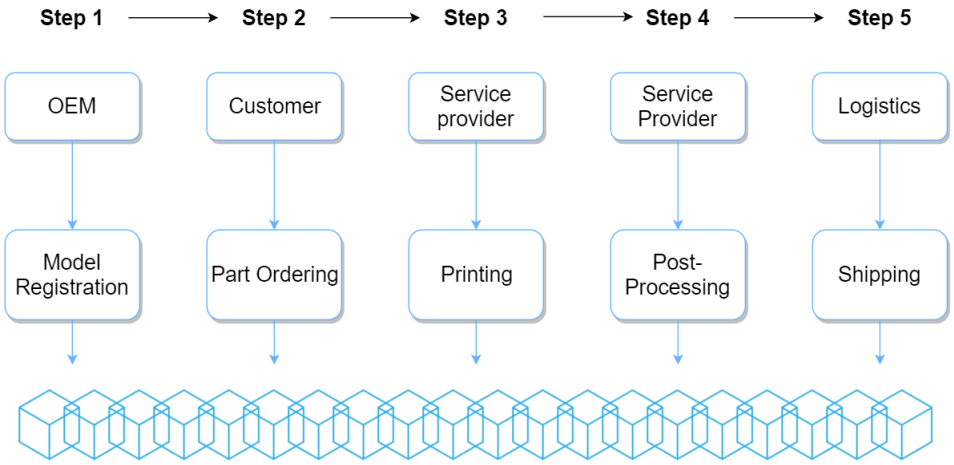


Figure 4.1: Supply chain steps and interaction with the Blockchain

This sequence is also visualised in Fig. 4.1. The main concept is that the status and data linked to the printed 3D part will be stored on the Blockchain network in an aggregated fashion as soon as it has gone through the corresponding step of the supply chain. For instance, as soon as a part has been printed, its representation on the Blockchain will be updated through the appropriate transaction.

We now elaborate on the above scheme and present the sequence of events during a 3D part's lifecycle and progress throughout the supply chain in combination with those events being reflected on the Blockchain. In this scenario we assume that an OEM has previously registered this model on the Blockchain, therefore the use case begins from the second step as shown in section 4.2. This flow is visualized in Fig. 4.2, where the type of interaction per step in this event sequence can be interpreted as follows:

- The solid lines represent processes which are external to our system and out of scope from this work.
- The dashed lines indicate an interaction with the Blockchain in order to invoke a specific transaction.
- The dotted lines express on-demand queries which any of the system's actors can execute at any point during or after the 3D part's production.

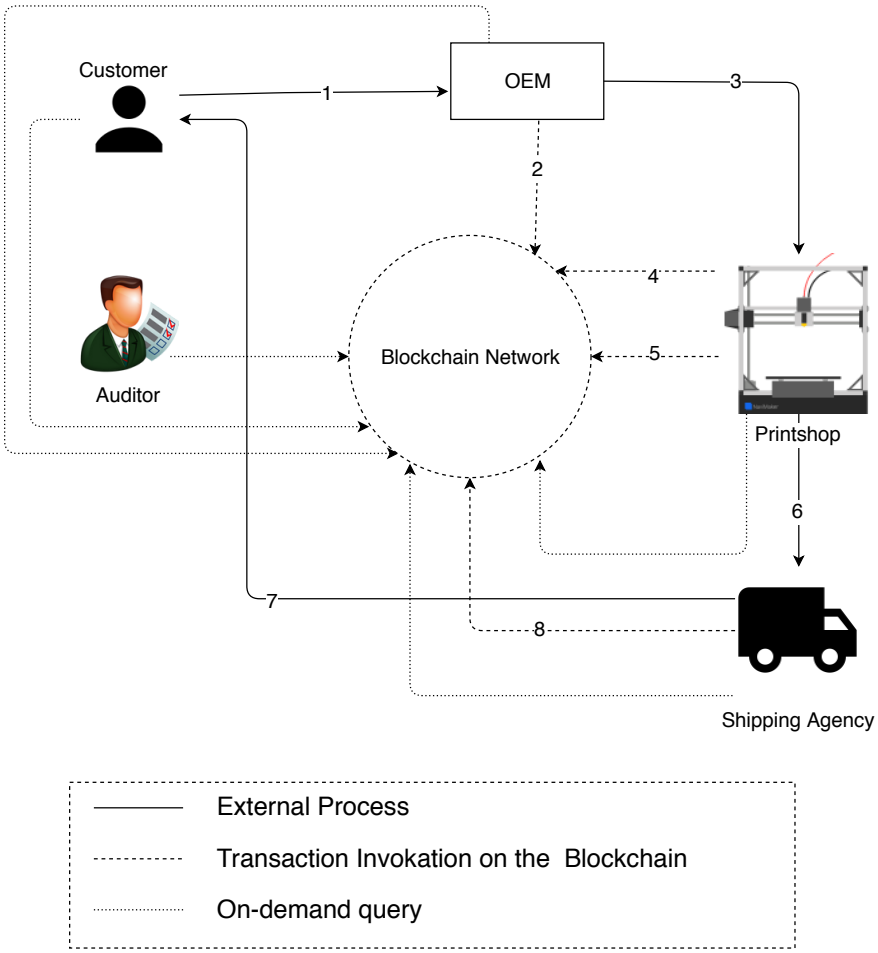


Figure 4.2: Event sequence for a 3D printed part

For better clarity, further explanation for each step follows:

1. The customer purchases a print of a 3D model.
2. The OEM invokes a transaction on the Blockchain network with the details of this purchase in encrypted form.
3. The model file is streamed to the designated printshop for printing
4. The part is printed. When the printing process has been completed, a second transaction containing the encrypted printing details is invoked on the Blockchain network
5. The part optionally goes through a post-processing process (e.g. painting of the part) and afterwards a third transaction including the post-processing details in encrypted form is invoked.

6. The part is transferred to the shipping agency
7. The part is shipped to the customer
8. A fourth transaction including the encrypted shipping details is invoked by the shipping agency.

More information about the contents of the above transactions follow below in section 4.4.2. At any time in the future or in between any of the above steps an auditor or any other interested party can query the network with the part's serial number and retrieve all the above information. Naturally, only entities among the ones involved with this particular 3D part will be able to decrypt the private data.

4.3 Privacy Principles

In terms of privacy the system primarily supports data confidentiality as well as pseudonymity. In order to satisfy our confidentiality requirement TPTS leverages the offerings of CP-ABE. Every OEM, printshop, customer, shipping agency and auditor using our system has an attribute accredited to them. This attribute enables them to decrypt data according to specified data policies. In more detail, the data is first encrypted with a random symmetric key and subsequently this symmetric key is encrypted with a CP-ABE scheme. The transaction which is sent to the Blockchain contains the ciphertext and the encrypted symmetric key as well as any other necessary cryptographic material. Any user of our system who holds an ABE-specific key which contains the appropriate attributes can then decrypt the symmetric key and with that key decrypt the ciphertext.

Since an ABE authority is in possession of a master key from which all private keys will be created, all entities would have to fully trust this authority. In our system, each OEM will hold the role of an ABE authority that provides private keys to customers, printshops, shipping agencies and auditors. We have made this design decision in order to limit the amount of trust these OEMs would have to place to a more central authority. In parallel, we expect that the other user types would accept this arrangement and trust the OEM they are purchasing parts from. This means that each customer, each printshop, each shipping agency and each auditing authority would have in their possession one private ABE key per OEM which they are interacting with in any manner (for instance buying a part from that OEM, printing a part or auditing the trail of a part throughout the supply chain). We also emphasize that an entity may not have two private keys from one OEM authority.

Having multiple OEMs in our system means that there exist multiple coexisting ABE authorities. It can thus be expected that the attribute mapped to each entity will be different among OEM's. For instance, a printshop collaborating with OEM Y and OEM X could receive a private key with attribute Printshop54 from OEM Y and a private key with attribute Printshop32 from OEM X. This is a desired discrepancy which will leverage anonymity among the entities and does not negatively effect the functionality of the system.

It would also be possible for each OEM to choose their own naming conventions for the attribute names, however in our work we maintain a uniform format of attribute names for the sake of simplicity.

Each entity is able to encrypt and decrypt data to be stored or retrieved from the Blockchain by themselves, by utilizing the software client of TPTS. Upon joining the system for the first time, each entity receives his private keys which reflect their attributes. This process for key retrieval is clarified further in subsection 4.5.2. It should be noted that the creation of an entity's private key is the only process where the private master key of each OEM will be utilized. We also point out that anonymity is further increased, since the fields declaring which entities are involved in a transaction (e.g. the customer's name) are encrypted and only decryptable by particular users.

Since there is no direct way to revoke ABE keys, if a user left an organization while having a private key with a certain attribute, she would still be able to read newly submitted data decryptable by that attribute, assuming she did not destroy the key herself. This is an undesired situation since it would allow for data leak to entities that should not have access to this new data. For that reason we also include within the policies a separate attribute stating the date when the private key was issued. For instance such an access policy might look like *((OEM234 or PRINTSHOP28 or CUSTOMER312 or AUDITOR) and (KEY_DATE > November 2017))* . We believe it is reasonable to allow for at least six months of grace period between the date of transactional invocations and the KEY_DATE specified. This means that if the transaction is executed on December 2017 it would be deemed acceptable to include a KEY_DATE of June 2017 within the access policy. Smaller acceptable time windows might also be appropriate if a tighter control is required on who is allowed to view new transactions, however that evidently means that keys would have to be renewed more often. It is therefore up to each individual OEM to decide how often keys should be renewed, while keeping in mind the trade-offs that would incur between security and additional key management.

Unlike private keys of all entity types which are expected to be renewed after a period of time, under normal circumstances an OEM would have to create their master key

only once. In case this master key is compromised then of course a master key renewal is mandatory. Private keys created through a new master key however will naturally be unable to decrypt old data even if they contain the same attribute.

4.4 Data Model

On the Blockchain, data are managed and stored in the form of assets. Three individual types of assets have been designed:

1. User Asset
2. Model Asset
3. Part Asset

Each asset is comprised of a *Key* and its *Value*, where Key is a string and Value can be any arbitrary data type. Queries for an asset are mainly done based on their Key. In Table 4.1 we present the functions which are available for creating and subsequently managing each type of asset. Each of these functions is a distinct transaction type which can be invoked on the Blockchain. The following subsections further explain the purpose of each asset type and the protocol that should be followed for proper usage.

Asset Type	**Transaction Types**
User	addUser queryUser
Model	registerModel updateModel queryModel
Part	createPart updateAfterPrinting updateAfterPostProcessing updateAfterShipping returnHistory queryPart

Table 4.1: Assets and available transaction types

4.4.1 User Asset

This asset is utilized for the goal of maintaining an index mapping user names with a user's public key. The purpose for this asset is solely to aid the distribution of ABE

keys by the OEMs and should not be confused with the registration of the user to the Blockchain Network itself. User assets are structured as follows:

- User name (Key): the identification used to interact with the Blockchain
- User's public key

We assume users own a pair of personal public and private key. Each user has the responsibility to create an asset for themselves by simply invoking the *addUser* function. For this transaction the user only has to provide the personal public key which they have created, since their username is automatically retrieved by the Blockchain system.

In turn, OEMs can query for a user name via the *queryUser* function to retrieve their public key and use it to encrypt the private ABE keys. These private keys will then be sent to the user over an external channel of choice. Finally, the user uses their personal private key to decrypt the data containing the private ABE key provided by the OEM.

4.4.2 Part Asset

This is the main asset in TPTS. Each 3D part's designated serial number is utilized as a *Key* and all other data are stored as parameters contained as *Value*. We focus on the main anticipated sequence of transactions, which refers to the scenario where a customer purchases a print of a model from an OEM and wishes to have it printed at a specific printshop. Essentially steps two to five of the supply chain are mapped on the Blockchain, as explained in section 4.2. The details for each of the four transaction types are analyzed below:

First Transaction: part ordering

The first transaction creates the asset and each subsequent transaction appends data to it. This transaction type is executed by the OEM as soon as a client purchases a print of a part. The fields of this transaction are as follows:

- Serial Number (Key): The serial number which will be assigned to the 3D printed part. Defined by the OEM itself. To be used by the system in order to link all the transactions that refer to this asset
- Purchasing details, which are stored in encrypted form and which include:
  - OEM data: the ID and organizational name of the OEM
  - Model ID: a unique ID assigned to the particular model. One model ID can be used to print multiple parts, each with a different serial number.

  - File hash: the checksum hash of the 3D model file. Used for data integrity verification purposes
  - Printshop data: the ID and name of the printshop the file will be sent to for printing
  - Customer data: the ID, name and address of the customer who ordered the part
  - Shippping agency data: the ID and name of the agency selected for shipping the part
  - Price quoted: the price agreed upon between customer and OEM for purchasing one print of this file
  - Password: an encrypted password, used to validate the transaction by the printshop, as explained below.
- PwdHash: a hash of the plaintext password, also used to validate the transaction
- Privacy parameters: data which enable encryption/decryption capabilities. In this transaction this is a symmetric key encrypted with an ABE scheme, as well as any other necessary cryptographic data.

Note that each instance of a part asset refers to a single print of a model file. In other words, when a customer purchases 10 prints of the same model, 10 assets will be created by the OEM (each with a distinct serial number as Key).

The ABE access policy which is used to encrypt this transaction has the following form: *(OEMA or CustomerB or PrintshopC or Auditor) and (KEY_DATE > YEAR.MONTH)*. The shipping agency is excluded from this policy since they should not have access to the above data. The asset status is initialized to "Ordered".

At this point it is valuable to describe the protocol for data verification of a part asset, which can be seen in Fig. 4.3. In more detail:

1. The OEM encrypts the data with a policy that reflects the entities allowed to decrypt the data. The OEM includes within the transaction a random string termed *password* which will be stored encrypted, as well as a hash that is the result of hashing this password, termed *PwdHash*. This hash, which will be included in the asset data in plaintext, will act as a “password quiz” that the printshop will have to solve on the second transaction as a method of verifying the data. The idea is that only the printshop and the customer are able to read this password and confirm the validity of the data of the transaction.

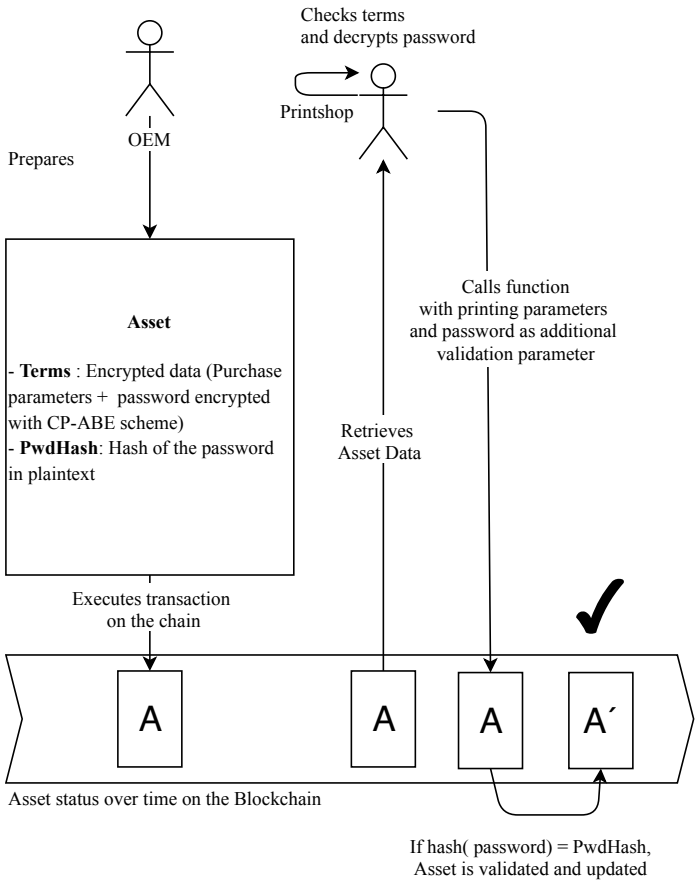


Figure 4.3: Data verification scheme

2. The part is then printed. Subsequently the printshop retrieves the asset data from the network, decrypts the secret password and executes a second transaction which adds the printing parameters to the created asset, while also providing the secret password in plaintext. During this step, the Blockchain nodes verify that the hash of the secret password provided by the printshop matches the *PwdHash* value from the first transaction and then update the asset data and status accordingly. Regarding requirement R7 of section 3.4, an updated status at this point signifies the validation of the asset and confirmation of correctness of the first transaction invoked by the OEM.

Second Transaction: part printing

The second transaction type is executed by the printshop, as soon as the printing process is completed. Ideally this would be triggered automatically upon completion

of the printing, but in practice this depends on the API that is available from the 3D printer in use. The fields of this transaction are as follows:

- Serial number
- Printing details, which are stored in encrypted form and which include:
  - Printer serial: the serial number of the particular printer that completed the printing process
  - Print price: the price the customer will be charged for printing this file
  - Printing parameters: duration of print, oxygen level etc.
- Privacy parameters: any necessary cryptographic data

The asset status is updated to "Printed". It is noted that the application which executes this transaction also outputs a QR code image with a serialized string which specifies the printshop ID, the customer ID, the OEM ID which produced the model as well as the serial number of the part. This image will be used in the last transaction type by the shipping agency as explained below.

Third Transaction: post-processing

The third transaction type is executed again by the printshop, as soon as the optional post-processing process is complete. The fields of this transaction are as follows:

- Serial number
- Post-processing details which are stored in encrypted form
- Privacy parameters

The asset status is updated to "Post-processed".

Fourth Transaction: shipping to customer

The fourth and final transaction is executed by the shipping agency, as soon as they receive the packaged part from the printshop, together with the QR code the printshop produced after the printing. Using a GUI, this QR code is then scanned to decode the necessary parameters: the customer ID, OEM ID and printshop ID which will enable the agency to produce the required access policy for the ABE encryption of the shipping data, as well as the serial number of the part to invoke the transaction for. Through this GUI the agency manually inputs the tracking number of the package and the data

is automatically encrypted with the above policy. The application behind the GUI then executes the transaction with the following fields:

- Serial number
- Shipping details which are stored encrypted and which include the tracking number for the package
- Privacy parameters: a new symmetric key encrypted with ABE which was used to encrypt the shipping details as well as any other necessary cryptographic data. This key is appended to the asset data without overwriting the original key from the first transaction

As an exception, the policy for the ABE encryption for this transaction includes the shipping agency. Therefore it has the form *(OEMA or CustomerB or PrintshopC or ShippingAgencyD or Auditor) and (KEY_DATE > YEAR.MONTH)*. Since the shipping agency does not have access to the original encryption key, a new symmetric key must be created and encrypted with this extended policy. Finally, the asset status is updated to "Shipped".

To summarize, after being shipped a part asset will be structured with the following data:

- Serial number
- Purchasing details
- Printing details
- Post-processing details
- Shipping details
- Privacy parameters
- Password hash
- Status: Shipped

4.4.3 Model Asset

Whenever an OEM creates a new 3D model that will be available to be purchased by customers and printed, they are able to register this model on the Blockchain. The main purpose for this asset type is to have a reference for the proper checksum hash

of a model file. If during an auditing process a 3D part – tracked on the Blockchain through an instance of a part asset – is found to have been printed using a 3D model file with a different checksum than the model asset registered by the OEM, then this means that the file transferred to the printshop was corrupted or altered.

In order to create a model asset one transaction with the following fields is necessary:

- Model ID (Key)
- Model name
- Owner: the name of the OEM who designed this model
- File hash: the checksum of the model file
- Model version

Any OEM can register their own 3D Model (with a unique ID) by invoking a *registerModel* transaction. A model can be updated with a new version only from the same OEM by invoking an *updateModel* transaction, in which case the file hash and version will be updated. Model assets can be retrieved through a *queryModel* transaction.

4.5 User Management and Distribution of Cryptographic Material

Each user in our system is identified by their cryptographic certificate – which would be generated the first time they join our system – and is able to decrypt transactional data according to the attribute injected in his private ABE key.

Before moving forward we note that introducing a new type of user to the system besides the five types already described – including attaching an attribute to the new user type entity – can be easily done when necessary. For instance, support for suppliers of printing powder could be added to the system for a more transparent traceability of the substance origins.

4.5.1 User Registration with the Blockchain Network

When a user wishes to join our system, they first need to get registered on a Certification Authority (CA) of the network before they are able to interact with it. The actions required for that purpose are as follows:

1. An admin user belonging to a CA of an organization operating within our system registers the user
2. The user receives from the admin his one-time credentials – a simple username and password pair – through an external channel that can be decided on a per-organization basis
3. These credentials are then used by the user to contact the CA in order to enroll herself to our network and retrieve her certificate

After the three above steps have been completed, the user can interact with the network by using his certificate. His original one-time credentials are considered invalid at that point. If the user loses his certificate or if these private files are compromised, a completely new registration process will have to take place and the old certificates will have to be added to a Certificate Revocation List. On the other hand, if the user's password is somehow leaked after she has already completed the enrollment there is no concern since it cannot be used more than once. This means that the security of the method of choice for transferring this password to the user is not crucial, as long as this method is direct and quick. A simple phone call could suffice in this scenario.

4.5.2 Private ABE Key Retrieval

After the user has completed their registration and is able to use the Blockchain network, they will need to receive a private ABE key from every OEM she wants to collaborate with. This key retrieval process is executed in the following manner:

1. The user creates a user asset for themselves on the Blockchain, as explained in subsection 4.4.1
2. An OEM is informed through an external channel about the interest of an entity – customer, printshop, shipping agency or auditing authority – to collaborate with them
3. The OEM then queries the network for the user asset linked to the user who requested the collaboration and retrieves the user's public key
4. The OEM creates a private key with the appropriate attribute this user should have and encrypts this private key with the public key of the user.
5. Through an external channel which is up to the OEM to decide, the encrypted private ABE key is then sent to the user

When these five steps have been completed, the user has full access to the network and is additionally able to decrypt future and past transactions involving her. Since our ABE approach supports attributes indicating the date of key issuance through the KEY_DATE attribute, it is expected that private ABE key renewals might occur. In that scenario, the entire above process would have to be carried out again.

4.6 Conclusion

In this Chapter the design decisions that have been taken in order to form TPTS have been examined. Through this design the Blockchain Network is thus shaped into a component with distinct sub-units, termed *chaincodes* in Hyperledger Fabric. These sub-units will provide the business process between the 5 types of entities in our system and the process of data storage. The following Chapter concerning the implementational details shows how such a system is built and analyzes its internals.

Chapter 5

Implementation

5.1 Overview

This chapter examines the implementation of the Blockchain network in combination with the client-side package which includes the library for CP-ABE support. The methods through which the Trustworthy Process-Tracing System (TPTS) can be built upon the Hyperledger Fabric (HLF) platform are analyzed. Additionally, the functionality and purpose of the three chaincodes which have been implemented is explained. All three chaincodes will be installed on every node of the Blockchain network and all stakeholders will be able to use the client-side application to interact with the network. A visualization of the interaction between the components of the implementation can be seen in Fig. 5.1. The *Node.js* bundle using the Node SDK[1] is responsible for interacting directly with the chaincodes. In parallel, the Python code provides all encryption capabilities through the *cpabe* script as well as all parsing processes for the data returned by the Node.js scripts. Data can be retrieved in three manners. Firstly through the *query_chaincode* script if decryption in a processed format is desired. Secondly as raw encrypted transactions through the *interact* script and thirdly through a browser GUI provided by the Blockchain Explorer component which also presents decrypted asset data.

[1] https://github.com/hyperledger/fabric-sdk-node

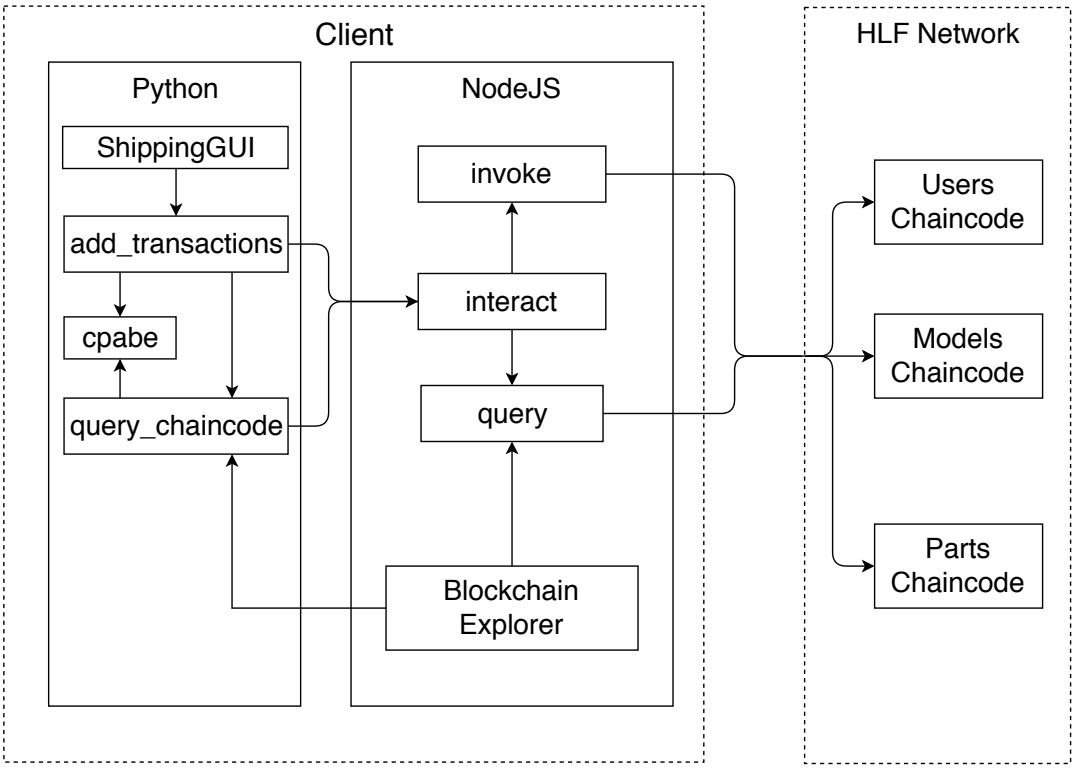


Figure 5.1: Inter-component interaction

5.2 Platform Selection

As stated already, out of the three platforms included in the Analysis chapter, the candidate we chose to proceed with is Hyperledger Fabric. The main reasons for this decision are:

- Its flexibility and modularity
- HLF is structured as a permissioned Network and fit for enterprise usage, this architecture allows us to know who has access to the network
- Identity verification and blacklisting is provided based on asymmetric key cryptography:
  - HLF offers a standalone Certificate Authority module (Fabric CA)
  - Future support is expected for distinct certificates per transaction. This would further elevate entity anonymity

- Entities can be stored as assets and updated on demand. Each 3D printed part can be regarded as an asset and tracked very conveniently throughout the supply chain
- Consensus protocol is pluggable and is decidable
- All data are replicated among all nodes which have joined a channel. This is important to us for the purpose of data accessibility and reliability

5.3 Development Environment

Due to the nature of the Hyperledger Fabric in combination with the libraries required to provide ABE support, numerous programming languages and platforms are employed in TPTS:

- Go (v1.8.7) is utilized to implement the chaincodes.
- Node.js (v8.10) is used to implement the application component which is utilized to directly access the peers of the Blockchain network.
- For Attribute Based Encryption, the pyPEBEL[1] wrapper for the CHARM library[2][2] is used , both of which are implemented in Python. The code for encryption and decryption processes can be found in the trust/crypto folder, also implemented in Python (v2.7).
- All HLF components are launched as Docker containers.[3] For this purpose, docker version 17.12-ce is utilized. Docker's "Swarm Mode" is used to deploy the HLF network in a distributed manner over different physical machines [4].
- Installation of a MySQL server is only required if the Blockchain Explorer is used, as it is necessary for locally indexing blocks and transactions

Regarding the HLF Docker images, the system was implemented and tested with the following image versions:

[1] https://github.com/jfdm/pyPEBEL

[2] https://github.com/JHUISI/charm/commits/dev

[3] For the deployment process we updated and adapted the framework provided by David Khala (https://github.com/davidkhala/delphi-fabric)

[4] In case Docker version prior to 1.12.0 must be utilized, you can use standalone swarm

- hyperledger/fabric-ca x86_64-1.1.0
- hyperledger/fabric-tools x86_64-1.1.0
- hyperledger/fabric-orderer x86_64-1.1.0
- hyperledger/fabric-peer x86_64-1.1.0
- hyperledger/fabric-javaenv x86_64-1.1.0
- hyperledger/fabric-ccenv x86_64-1.1.0
- hyperledger/fabric-zookeeper x86_64-0.4.6
- hyperledger/fabric-kafka x86_64-0.4.6
- hyperledger/fabric-baseos x86_64-0.4.6

In terms of libraries provided by HLF, the following are used:

- fabric-client@1.1.0
- fabric-ca-client@1.1.0

5.4 Infrastructure Setup and Deployment

This section describes how the Blockchain network is configured and deployed. The network itself can be deployed simply based on a number of parameters included in a single configuration file. This file includes parameters such as:

- Consensus mechanism
- Organizations operating in the network
- Peers and CA's belonging to each organization
- Ordering nodes

Additionally, the Blockchain network is extensible in a number of different ways:

- The three chaincodes responsible for processing the transactions for users, models and parts can be updated at will.
- A completely new chaincode can also be added if desired
- New nodes and organizations can be added to the network dynamically

All the above operations can be carried out on the fly without having to take down the network for upgrades.

A bootstrapper which takes care of all steps required to launch a HLF network has been created. All deployed nodes have TLS enabled and can be launched either locally on the host's docker or as services over docker swarm, enabling them to be deployed over multiple physical machines. The bootstrapper will proceed to connect all nodes to a channel, install and instantiate the chaincodes on them. After this automated process, the network is ready to properly receive invocations and send responses. Fabric CA[1] (FCA) is utilized for creating and managing users interacting with the Blockchain network through ECDSA Certificates [11]. Each user receives these certificates upon enrollment with a FCA.

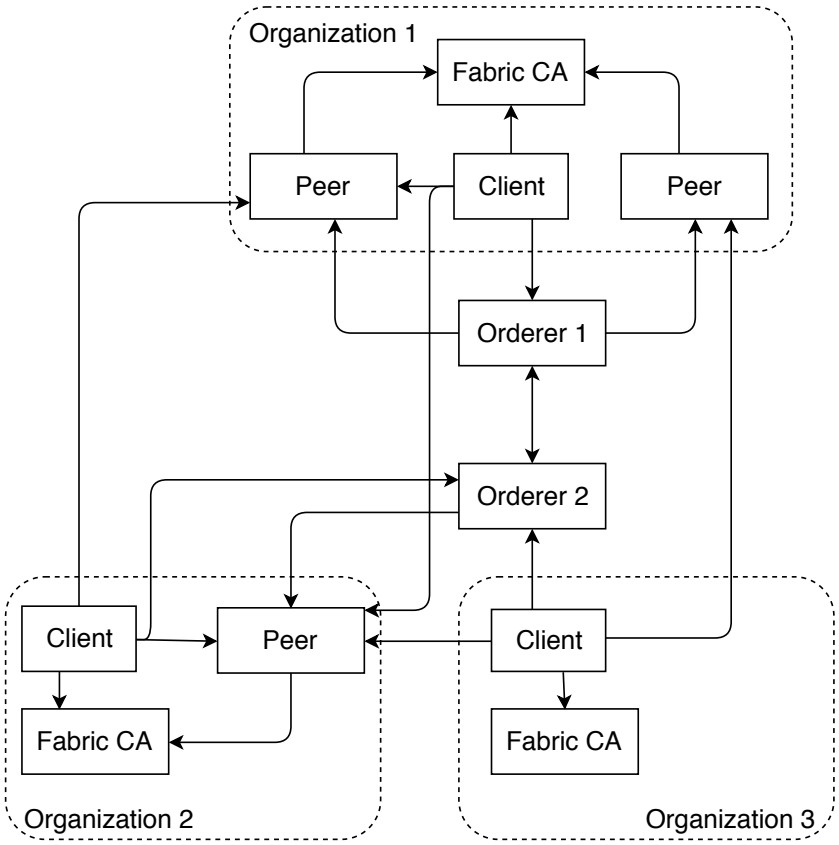


Figure 5.2: Hyperledger Fabric network layout

[1] https://github.com/hyperledger/fabric-ca

A default configuration layout for the network has been included and can be seen in Fig. 5.2. The HLF network is set up with a number of different organizations, each with their own peers and one FCA node per organization. A few important details can be noted through this network layout:

- An organization can have zero or multiple network peers
- A user belongs to exactly one organization in order to use the application client
- To successfully invoke a transaction a client will have to contact one or more peers, depending on the endorsement policy the chaincode has been instantiated with. For instance, one can specify a "2-of-N Orgs" endorsement policy and demand that one peer from at least two different organizations has to endorse the transaction. An example can be seen in Fig. 5.3.

```
'endorsement-policy': {
    identities: [
        { role: { name: 'member', mspId: ORGS[0].id }},
        { role: { name: 'member', mspId: ORGS[1].id }},
        { role: { name: 'member', mspId: ORGS[2].id }}
    ],
    policy: {
        '2-of': [
            { 'signed-by': 0 },
            { 'signed-by': 1 },
            { 'signed-by': 2 }
        ]
    }
}
```

Figure 5.3: A sample 2-of-N-Orgs endorsement policy

The orderers are set up with a Kafka-based ordering service [1]. All operations are carried out on a single HLF channel which all peers and orderers join, therefore all data in this channel will be replicated on all peers of every organization.

For testing purposes the certificates and keys for the peers and orderers are currently created all together locally through the cryptogen tool that is provided by the HLF binaries. This tool is not designed to be used in production. Instead, for security

[1] At the time of writing there was no implementation for Practical Byzantine Fault Tolerance on HLF, although it is planned for the near future. PBFT ordering consensus should most definitely be selected instead of Kafka when it is available

purposes this material should be created through the FCA on each individual node that requires it.

5.5 Chaincode Implementation

Three distinct chaincodes have been implemented. The first serves the purpose of an index for user assets and their public keys. The second is used for registering model assets and the third one is used for tracing 3D printed part assets. These chaincodes follow Table 4.1 as depicted in the design Chapter. Additional chaincodes can be easily created and effortlessly installed on the network peers, since HLF is designed to be efficiently extensible. Each of the functions available from the chaincodes can be invoked via the *interact* script with the appropriate parameters and arguments.

5.5.1 Users Chaincode

This chaincode implements the user asset described in subsection 4.4.1 and serves the purpose of the first step towards the retrieval of the private ABE key by a user, as explained in subsection 4.5.2. The user is responsible for invoking the *addUser* function of this chaincode in order to create a user asset with the Common Name (CN) field of her certificate as well as her RSA public key.

5.5.2 Parts Chaincode

This can be considered as the main chaincode of TPTS, since it is managing the 3D printed parts and tracking their progress throughout the supply chain. This chaincode implements the part asset described in subsection 4.4.2.

While the fields of this chaincode asset follow the design indicated in subsection 4.4.2, we ought to elaborate on the fields which contain cryptographic data. The part asset is therefore structured as follows:

- Serial number: used as key of the asset
- Purchasing details: an encrypted json object which contains all the information of the first transaction type as explained in subsection 4.4.2
- Encryption key: a symmetric key, encrypted with an ABE scheme with the appropriate policy

- IV1: the random IV that was used in conjunction with the encryption key to encrypt the purchasing details field
- Password hash: the hash of the secret password created for data validation purposes, as explained in subsection 4.4.2
- Printing details: an encrypted json object which contains the information of the second transaction type
- IV2: the random IV that was used in conjunction with the encryption key to encrypt the printing details field
- Post-processing details: an encrypted json object which contains the information of the third transaction type
- IV3: the random IV that was used in conjunction with the encryption key to encrypt the post-processing details field
- Shipping details: an encrypted json object which contains the information of the fourth transaction type
- Encryption key 2: a second symmetric key, encrypted with an ABE scheme with the appropriate policy.
- IV4: the random IV that was used in conjunction with the second encryption key to encrypt the shipping details field
- Status: an enumeration field with choices among:
  1. Ordered
  2. Printed
  3. Post-processed
  4. Shipped

  Status is internally controlled by the chaincode and changed as the part goes through the supply chain.

When a 3D printed part has finally reached the customer, one can expect all asset fields to contain data of the according supply chain step. The only field that might not be filled is the *post-processing details*, since post-processing is an optional step.

5.5.3 Models Chaincode

This chaincode is designed to be used strictly by OEM's wishing to register their newly created 3D models on the Blockchain network. The structure of this asset and the possible functions that an OEM can invoke upon the models chaincode follow the principles described in subsection 4.4.3. Each model file designed by an OEM is mapped to a model asset and each of these model assets contain the following information:

- Model ID: used as key of the asset
- Model name
- Owner: the ID of the OEM that registered this model
- File hash: the sha-256 checksum of the model file
- Model version

5.6 Client-Side Implementation

This section describes the implementational details of the scripts (written in Python & Node.js) that a user can utilize to interact with an existing operational HLF network. The interaction between these scripts can be seen in Fig. 5.1, while their functions, parameters and parameter types can be reviewed in Table 5.1.

These scripts utilize a *config.json* file which the user has to adapt. Within this configuration file, the following parameters can be specified:

- Username
- User password for certificate retrieval
- Addresses of peers that can be contacted (of the same or different organizations)
- Addresses of orderers
- Address of the FCA

An exact example of how this json file should look like can be found in Appendix A.

5.6.1 Authenticating with the HLF Network

There are two distinct processes which have to be executed before a user is able to interact with the HLF Network. These are the registration and the enrollment process,

Script	**Parameters**	
add_transactions.py	tx_type	*{1,2,3,4}*
	serial	*string*
	json_data	*json*
	access_policy	*string*
	oemid	*integer*
	extra_args	*string array*
cpabe.py	function	*{setup,create, encrypt,decrypt}*
	message	*string*
	symmetrickey	*string*
	policy	*string*
	oemid	*integer*
	attributes	*string*
	iv	*integer*
query_chaincode.py	function	*string*
	chaincodeid	*string*
	args	*string*
	oemid	*integer*
interact.js	function	*string*
	chaincodeid	*string*
	channel	*string*
	transient	*string*
	args	*string*
	type	*{query, invoke}*

Table 5.1: Functions and parameters of available Client-Side scripts

executed in this order. Registration has to be done once per user and is executed by an admin of the organization the user belongs to. This registration process returns a pair of credentials the user can utilize to enroll. The enrollment process itself on the other hand is carried out by the user herself, by providing the credentials acquired during registration. After enrolling herself, the user is in possession of the certificates which will allow her to interact with the HLF network. In TPTS the FCA is configured to only allow one enrollment per user, to enforce the design decision for one-time credentials. This means that after the user is enrolled and has received their credentials, their password is no longer valid for any further action.

In order to register a user, an admin of an organization has to first enroll themselves through the *enrollAdmin.js* script. Subsequently they are able to register a user by using the *registerUser.js* script, which on success returns a one-time password. We expect that

users would receive their credentials (username and password) via a different physical or digital channel, however we leave that decision to each organization.

Each user would then adjust the config.json file with their credentials and execute the *enrollUser.js* script, which would in turn enroll the user on the HLF network and retrieve their appropriate certificates from the indicated FCA Server. Beyond this point, a user is allowed to invoke transactions and execute queries towards the network.

5.6.2 CP-ABE Implementation

Encryption and decryption of data is required in the *add_transactions* and *query_chaincode* scripts accordingly. As depicted in Fig. 5.1, these procedures are carried out by the *cpabe* script. This subsection examines the four functions provided by this script: setup, create, encrypt and decrypt.

For all ABE procedures we utilize the implementation of [6] which is included in the CHARM library. In order to convert numerical dates into ABE attributes and subsequently include them in access policies, the system utilizes the approach followed by the pyPEBEL library [1]. The CP-ABE mechanism is leveraged exclusively for part assets, since the other two asset types do not have privacy requirements and their data can be left public.

All generated ABE keys adhere to the following naming conventions:

- An OEM with ID X (i.e. OEMX) will have these keys:
  1. X: master key of OEMX, generated through the "setup" function.
  2. X.pub: public key of OEMX, generated through the "setup" function
  3. X.key: The OEM's private key in their own ABE authority "realm", generated through the "derive_personal_key" function
- An entity of any type (printshop, customer, shipping agency or auditor) will have the following pair of keys for every OEM they are collaborating with:
  1. X.pub: public key of OEMX
  2. X.key: The entity's private key in the "realm" of OEMX

[1] https://github.com/jfdm/pyPEBEL/blob/master/pebel/policy.py

To clarify further, in the scenario where a printshop collaborates with OEMX and OEMZ then this printshop will have in their possession 4 files: X.pub, X.key, Z.pub, Z.key.

We now focus on the two main functions of encryption and decryption. When encryption of data is requested, the following parameters are required by the cpabe script:

- The public key of the OEM authority that the data will be encrypted with
- The plaintext data
- An access policy the data should be encrypted with

A random symmetric key is generated based on a group pairing initialized in the elliptic curve setting with a 512-bit base field. Using this key and a random IV, the plaintext data provided by the user is encrypted with Advanced Encryption Standard in Cipher Feedback Mode (AES CFB). The symmetric key is subsequently encrypted with CP-ABE and a json object with three keys is returned:

- The serialized encrypted symmetric key
- The ciphertext of the plaintext data, in base64 encoding
- The random IV in base64 encoding

For the decryption process on the other hand, the following parameters are required:

- The public key of the OEM authority that the data was encrypted with
- The user's private key for that OEM authority
- The ciphertext
- The symmetric key encrypted with CP-ABE
- The IV used for encrypting the plaintext

Assuming the private key provided contains the required attributes, then the symmetric key is successfully decrypted. This enables subsequent decryption of the ciphertext to reveal the original plaintext, which is returned by the script. At this point, the end-to-end processes for encrypting and decrypting data has been fully covered.

5.6.3 Creating and updating a Part Asset

A user can utilize the add_transactions script to invoke transactions on the Blockchain network which create or update a part asset. The user supplies the script with the type

of transaction they want to invoke (one of the four explained in subsection 4.4.2) as well as the parameters required for each type.

Part purchase

Let's assume a customer just purchased a print of a specific model. The sequence of events is as follows:

Step 1:

The purchase of a part print triggers execution of the add_transactions script, supplying the following parameters:

- Type: the type of the transaction. In this case "1"
- Serial_number
- Purchasing_details: provided in a json object
- File_hash
- Access_policy: the CP-ABE policy to encrypt with
- OEM_ID: the OEM authority for which to encrypt the data

Step 2:

The *add_transactions* script will then pass the purchasing details to the *cpabe* script for encryption by triggering the *encrypt* function, together with a desired CP-ABE encryption policy and the public key of the OEM authority we should encrypt for. The encrypt function then returns the encrypted symmetric key, the IV as well as the ciphertext of the json object with the purchasing details.

Step 3:

The script then executes a chaincode invocation by triggering the *interact* script. The parameters supplied to this script are:

1. the chaincode name to invoke: parts
2. the chaincode function: createPart
3. the type of transaction: invoke
4. the additional arguments:

- the serial number of the part
- the above ciphertext
- the encrypted symmetric key
- the IV that was used to encrypt this ciphertext
- the hash of the password field which was included in the purchasing details. This will assist in data validation in the next steps of the supply chain

The HLF network peers – as configured in the config.json file – are then contacted and the procedure of endorsement as described in section 2.2 is carried out. Note that depending on the endorsement policy of the HLF Network, endorsements from multiple peers might be required. In this case the user ought to specify the necessary peers – for instance one from each individual organization – within the configuration file. At this point, and assuming no errors occurred, a new part asset describing this 3D part has been created with an internal state of *Ordered* and stored on the network.

Part printing

Let's now assume the above purchased part has been printed at the designated printshop.

Step 1:

The completion of the printing process will trigger execution of the *add_transactions* script just like above, but with different parameters. This time, the part's serial number will also be provided to the script so that the asset data created the previous step can be acquired from the network. The exact parameters that are required are:

- Type: 2
- Serial_number
- Printing_details: provided in a json object

The script will query the HLF network with the serial number to retrieve the part asset and decrypt the purchasing data to restore the secret password. Subsequently, the printing_details will be encrypted by the *cpabe* script using the retrieved symmetric key and a new random IV (IV2) to form a new ciphertext.

Step 2:

On this step we are leveraging a feature of HLF termed *Transient Map*. This is an optional map that can be used by the chaincode but not saved in the ledger and can be useful for passing arguments containing sensitive data such as cryptographic information for encryption.

The add_transactions script triggers an invocation to the parts chaincode by calling the *interact* script. The parameters supplied to this script are:

1. the chaincode name to invoke: parts
2. the chaincode function: updateAfterPrinting
3. the type of transaction: invoke
4. the additional arguments:
    - The above ciphertext with the printing details
    - The IV2 that was used to encrypt this ciphertext.
    - The password, to be passed on not as a regular argument to the chaincode but as a transient map

The asset has now been updated with the new data and an internal state of *Printed*.

Post-processing

This type of transaction is processed similarly to the second. The only difference is that we no longer supply the password value to the chaincode, since this transaction is executed by the same printshop, at least in this scenario. As stated in subsection 4.4.2, the payload data that will be added to the asset with this transaction are the post-processing details and the asset's internal state will be updated to *Post-Processed*.

Shipping

For this transaction the shipping agency uses a simple GUI which has been implemented with the wxPython toolkit. This GUI prototype is set up to scan QR codes through a web-cam and deserialize a formatted string. Upon submission of the shipping identification by an operator, the application combines the tracking and QR data and submits these parameters to the *add_transactions* script. This in turn triggers an updateAfterShipping transaction. The asset is then updated with the last data and an internal state of *Shipped*.

5.6.4 Retrieving Data from the Blockchain Network

Data retrieval on HLF is channel-based. This means that clients are essentially querying a HLF channel and not the individual peers for the data they need. This enables users to always have access to this data as long as at least one of the peers of the channel is responsive. In order to satisfy the requirement of viewing transactional data as noted in section 3.4 (R8), three distinct methods are supported, depending on the use case.

Viewing decrypted data

For the first method the parameterized script *query_chaincode* was implemented. It allows the user to interact with the chaincode in order to query for 3D parts based on their serial number, or alternatively query for transactions by providing the transaction ID. We retrieve the encrypted ciphertexts from the network for the target asset or transaction and decrypt it by using the appropriate ABE key in order to reveal the plaintext data. We additionally apply some formatting to the retrieved information before returning it to the user.

Viewing raw transactional data

The ability to view raw transactions exactly as they were recorded on the ledger with their encrypted data is also provided through the *interact* script. Naturally the data returned with this method include signatures, certificates and metadata besides the payload, so the readability of the retrieved information is not high. However it is important to provide access to these raw data, in cases where the exact original transactional data are required by an auditor for instance.

Inspecting the ledger with Blockchain-Explorer

For overall inspection of the network status, TPTS also includes a front-end application utilizing the open source tool Blockhain-Explorer [1] which belongs to the Hyperledger ecosystem. This allows users to have a total overview of the Blockchain network including connected peers, installed chaincodes as well as total blocks and their transactions. In addition to the already offered query functionalities based on block number and transaction ID that the Explorer's interface already provides, the project has been adapted to additionally include serial-number based search in order to facilitate part

[1] https://github.com/hyperledger/blockchain-explorer

asset lookups. We have also integrated the application with the decryption mechanism for the encrypted information, therefore the presented data appear decrypted to the end user.

The Blockchain-explorer has to be configured with the addresses of peers which can be contacted for queries, the HLF channel that the chaincodes reside in, as well as the certificate path of a user with sufficient access to that channel. The concept here is that an organization participating in TPTS would host one of these front-ends within their premises, should they decide they want to utilize this component.

Chapter 6

Evaluation

In this chapter TPTS is evaluated in terms of performance, security, usability as well as overall requirement satisfaction based on the requirements defined in section 3.4.

6.1 Performance Evaluation

In order for TPTS to operate correctly, the Blockchain component must be reliably responsive to queries and invocations, particularly in the presence of high load, unexpected issues or failures. In this section TPTS is tested through a number of quantitative metrics to evaluate the capabilities of the Hyperledger Fabric (HLF) network and its potential limitations.

6.1.1 Evaluation Environment

A HLF network is set up over two distinct collocated virtual machines cooperating over docker swarm. The baseline configuration for the network layout is as follows:

- 3 Organizations with 2 peers each
- 4 Orderers
- 3 Kafka brokers
- 3 Zookeeper nodes for the Kafka brokers to use
- 3 Fabric CA's, one for each Organization

The above nodes are distributed between the two virtual machines operating in our test system. All operations are executed on an Intel 6700K processor with 16GB RAM and an SSD for storage. The *add_transactions* script is employed for the purpose of creating part assets in a multi-threaded environment, inserting random data and executing all four transaction types for each part.

6.1.2 Throughput

A valuable metric is the network's ability to process a large amount of messages in a short period of time. HLF accepts a *Batch Timeout* parameter which can be configured within the configtx.yaml file used for deployment. This parameter dictates how often the ordering nodes reach a consensus regarding the transactions which will go into the next blocks. *Max Message Count* is another configurable threshold for controlling how many transactions can fit into a single block. Should the orderers receive an amount of transactions that reaches beyond this maximum threshold, many blocks can be created and ordered at the same time in order to fit all transactions. Unfortunately we did not have access to a large cluster from which to launch a considerable burst of invocations, therefore transaction execution was tested with a an amount of 50 transactions executed in parallel – keeping in mind the 8-thread count maximum of the processor used for testing – against one or more endorsing peer, with a Batch Timeout of 2 seconds and a Max Message count of 10 messages. Findings show that the system remained stable and responsive, as all transactions were submitted and stored successfully without delays or inconsistencies.

While HLF could perform adequately even with a much larger burst of invocations, it should most definitely be noted that the limits of this system are considerably lower than those of a traditional database system. HLF was measured with a maximum throughput of approximately 4000 transactions per second (tps) when using 32 virtual CPUs [3]. This is noticeably lower than the capabilities of a MySQL server, but considerably higher than public Blockchains such as Bitcoin (7tps [7]) and Ethereum (7-15tps currently).

We remark nevertheless that throughput simply has to be kept over a reasonable threshold. This metric was not a core objective for this work and in a production setting different industries would use different networks and peers to conduct their operations. In this regard we deduct that the performance of HLF is adequate in terms of throughput.

6.1.3 Resilience Testing

On occasion, we can expect that the network over which nodes are communicating is facing unexpected issues for a number of reasons. It is critical that the HLF network is resilient enough to operate under these sub-optimal circumstances. In order to evaluate system behaviour under these conditions we employ Pumba [1], a testing tool which is able to simulate poor network conditions among docker containers.

Using Pumba, latency as well as packet loss (PL) were injected into the system on all containers for the entire duration of the resilience testing process. During these tests the system behaved correctly even with high PL and only exhibited issues with an extreme PL of 90%. Positive results were also observed with latency testing, where the peers maintained mostly proper functionality and were able to process most requests. Transactions only started occasionally timing out when a consistent latency of 2 seconds was introduced to the network for a period of 5 minutes. However a simple re-invocation was sufficient to successfully store the asset data.

The behaviour of the HLF system when randomly taking down and restarting network nodes is also tested. HLF nodes that restart after being shut down will automatically attempt to rejoin the channel they had joined before their shutdown, so the system continues to behave correctly as long as at least one node of each type is accessible. It is also noted that in a scenario where invocation of a transaction is attempted on multiple endorsing peers, a number of which are not responding, then the transaction will still go through to the ordering service as long as enough endorsing peers are accessible for the endorsement policy requirements to be met.

6.2 Requirement Satisfaction

In this section TPTS is evaluated from a standpoint of the quality of its features and how they satisfy thetated requirements.

[1] https://github.com/alexei-led/pumba

6.2.1 Evaluation of Main Goals

Requirements 1 and 8 respectively state:

R1 The system must offer reliable and tamper-resistant traceability of an item throughout the supply chain

R8 Users and organizations must be able to query for an item based on a selected identifier and retrieve all data related to this item

Concerning *R1*, the design of TPTS supports the desired features for traceability: the distributed network enables storage of data and subsequent retrieval of transactions in a reliable manner. The sequence of events begins with the registration of a 3D model by an OEM, after which the system tracks each individual print order of this model through four additional supply chain steps. For each of these steps a record is created which includes the parameters that the stakeholders would like to track.

It is noteable to remark that the 3D printing use case was investigated simply as a proof of work and that the system can be extended to other industries. The chaincode is mostly unaware of what payload data are included in the transaction of each supply chain step as most information is encrypted in the appropriate fields. Therefore the implementation can be easily adapted from the client-side to include different parameters per transaction. Additionally, the chaincodes can be effortlessly extended to include more supply chain steps and more functions.

Regarding tamper-resistance, it is possible through tools provided by HLF to inspect the stored ledger and verify its integrity to make sure that data has not been altered. However one weak link in the system is the consensus mechanism for the ordering service currently used by TPTS, which is based on Kafka. This ordering service is crash tolerant but does not provide the required Byzantine Fault Tolerance. An implementation of a BFT ordering service has already been proposed [5] and it is of major importance that the Kafka consensus is replaced by a BFT consensus as soon as it is released by the HLF developers. Based on the above, we consider *R1* to be only partially satisfied at the time of writing but we note that simply plugging in a BFT ordering service in the near future would lead to proper tamper-resistance for data.

In subsection 5.6.4 the three options for retrieving data from the Blockchain network were examined. These approaches enable viewing data with different pre-processing done, ranging from completely raw transactional data to decrypted and formatted data for easy readability. Queries can be executed either by using a serial number of a part, a name of a model, or even a specific transaction id. Additionally, these queries respect the privacy requirements since decryption of data is done on the client side.

Unlike centralized approaches, data in TPTS is always available for retrieval given the distributed nature of the Blockchain network. This means that regardless of potential node failures queries should always return the requested data. In this regard *R8* is fully satisfied.

6.2.2 Privacy

Requirements 3 and 4 state:

R3 The stored data must be private. Only users authorized through a specified access policy should be given access to specific data.

R4 The system should support a key escrow scheme for auditing authorities to be able to view private data. New auditors joining the system should additionally be able to view private transactions executed in the past

Chapter 4 describes how privacy is incorporated in TPTS through ABE as well as how attributes are mapped to each of the different type of stakeholders in the system. By encrypting the data with a symmetric key and subsequently encrypting this symmetric key with ABE, a mechanism is produced that provides the desired privacy features while maintaining considerably low encryption and decryption times: approximately 90ms for encryption and 15ms for decryption, with data volumes following the sample of Fig. 6.1. This approach alone provides the required confidentiality since, to the best of our knowledge, there have not been any vulnerabilities documented against this ABE scheme.

One potential issue is having a private key with certain attributes given to an entity that should not have access to these attributes. For this situation TPTS has defenses in place. Firstly, a system operator has full knowledge of the identity of each participating user since all entities are enrolled in one of the system's Certificate Authorities. Assuming each individual organization sufficiently controls who gets registered and enrolled, it should not be possible for an outsider to receive certificates that enable her to interact with the network. Secondly, the only way for a user to obtain a private ABE key is by creating a user asset for themselves. Since the username is retrieved automatically by the chaincode from the user's certificate, forging one's username on a user asset is not possible. Therefore, as long as an OEM – acting as an ABE-authority – makes sure to always send private keys containing the right ABE attributes to the right usernames, this scenario cannot threaten the system. In conclusion, *R3* and *R4* are fully satisfied through the design of TPTS.

Nevertheless, we ought to discuss the trade-offs between privacy and transparency. Encrypting data for specific entities means that other actors – who could potentially benefit themselves or the original entities involved in the transaction – don't have access to this information. For instance, let us suppose that a retailer using TPTS as a customer is ordering 3D models of car parts and has them printed. An alternative transparent system allowing all data to be public would permit all potential customers to be able to see the standards under which the parts were printed. In return, this might encourage more sales just because of the added trust it would place upon the retailer when the customers know they are purchasing quality parts. There are of course middle ground solutions such as the retailer offering on-premise access to the network to interested customers, but an entirely public system would naturally be a lot more accessible by external users.

Additionally, encrypting data means that it is not possible to provide support for complex queries on assets. HLF offers the possibility to use CouchDB as a state database instead of LevelDB which is the default. When chaincode values in assets are modeled as JSON data, CouchDB enables rich queries against these values. There exist a number of use cases where dealing with plaintext data and being able to execute rich queries would be beneficial for certain actors. For example, a printer manufacturer might wish to review how many hours a specific printer which encountered a malfunction has spent printing parts. The current design, where all data is stored in encrypted form, does not support this use case. On the other hand, if the printer manufacturer was added to the access policy this would violate the privacy requirements of the other entities. Although this is an expected and reasonable trade-off, careful consideration has to be given to the balance between usability and privacy.

6.2.3 Anonymity

Requirement 5 states:

R5 The stored data must preserve anonymity for the users. Users should not be able to have knowledge of which entities were involved in a transaction they are not authorized to access

In HLF it is possible to query the HLF network for a transaction by its ID without having to invoke a chaincode. This means that any entity with access to the network can query for those transactions. When retrieving a transaction in this manner it is possible to see the certificate of the entity who invoked it and thus see the username of this user. Therefore any user who has access to the system – regardless of its organiza-

tional affiliation and other attributes pinned to her certificate – will be able to see the certificates of three actor types:

- The OEM who executes the type 1 transaction for the asset
- The printshop who prints the part and who executes type 2-4 transactions for this asset
- The shipping agency which transfers the part to the customer

Naturally, it must be pointed out that the only useful information that can be retrieved from these certificates are restricted to the commonName field, which by itself cannot reveal any critical information. This pseudonym is not visibly linked to an actual organization and the real identities (i.e. the names) of the entities involved in a transaction are adequately secured under the ABE scheme. Therefore, while it is possible to know the pseudonyms of the three organizations or actors involved in a transaction involving an asset with a given serial number, no other information can be retrieved by anyone other than those three actors or an auditing authority.

In the absolute worst case scenario that the link between the ID of an entity and the actual name of the organization is revealed through external channels, it is possible to create new ID for that entity. This can be done by registering this entity with a new username. For past transactions it would still be possible to know the identities of the actors whose information have been revealed or leaked in some way, but the data contained in the transaction would still be protected by the ABE scheme. However, future transactions would be secure.

There is of course an approach that would make even the aforementioned worst case scenario infeasibly hard to occur. To elaborate, it would be technically possible to execute each transaction with a new user – and a new pseudonym – which would elevate the anonymity even further. However, processing the constant new registrations of users would require a considerably higher complex system with an enhanced user management scheme. This scheme would have to be able to reliably map each of the new pseudonyms to one single person or organization, while maintaining these mapping data anonymous. Nonetheless, this is not a reasonable trade-off given our basic level of anonymity requirements and we are satisfied with the current balance of anonymity and system complexity [1].

[1] There exists a new feature in HLF termed *Identity Mixer* which has a similar concept and a purpose of leveraging unlinkability, however at the time of writing of this thesis it was still under development and not available for usage

Our conclusion is that the system is designed and implemented in such a way that *R4* is sufficiently satisfied, although not to an optimal level.

6.2.4 Platform Evaluation

We begin with our considerations regarding the choice of HLF over other potential candidates. During the implementation and evaluation process, we found that a number of its aspects such as the Software Development Kits and certain deployment-related procedures are unfit for production purposes at the time of writing. This is understandable as HLF is a new platform and this is something that has been improved a lot upon over the past few months. It is reasonble to believe that Ethereum Quorum would potentially present less issues concerning setting up, configuring and interacting with the system. Nevertheless, HLF proved to be an adequate system on which to implement the design of TPTS, with very few workarounds needed.

Requirement 2 states:

R2 The chosen platform on which the system will be based should provide Byzantine Fault Tolerance as well as highly available access to the data

High availability to the data is provided to us by design through the distributed nature of the network. In the event of node failure the other peers can accommodate any queries or invocations. However, as already stated in section 5.4 the network is setup using a Kafka ordering service. This is a superior option compared to using a solo ordering service with a single orderer where orderer downtime would mean no data could be added to the network. However even this Kafka-based approach does not provide Byzantine Fault Tolerance as explained in subsection 6.2.1.

One of the reasons HLF was the platform of choice for this thesis is its pluggable ordering consensus mechanism. When a BFT implementation is eventually released it should be easy to replace the Kafka ordering service with it. In that case the system would thus remain largely untouched and only the HLF deployment configuration would have to be adapted accordingly with the BFT ordering service. Based on the above we conclude that *R2* is only partially satisfied at the time of writing.

6.2.5 Storage Requirements and Longevity of System

Part Serial No.:	sn6800b727fc53097d
Status:	Shipped
Model:	Model#70302
File hash:	35d91262b3c3ec88
OEM ID:	OEM#1
OEM Name:	Boeing
Printshop ID:	Printshop#78459
Printshop Name:	Printland LTD.
Customer ID:	Customer#40536
Customer Name:	Airbus FR
Customer Address:	Northington Street 46, 88888 Somewhere
Price quoted:	€2695
Printer Serial:	YOJCOFSUSY
Print Parameters:	Laser power: 65 °C oxygen: 0.727% Print duration: 941mins
Post-Processing:	Paint Color: Ultramarine
Tracking No.:	JJD1437064736260

Figure 6.1: Asset data sample

Requirement 9 states:

R9 The system should be tuned in a way which allows it to operate over a number of decades without overwhelming its resources or exhibiting performance loss, while in parallel maximizing the amount of valuable data it can hold.

The storage volume per part asset is expected to account for most of the total data stored on the ledger. These storage requirements are measured by creating part assets with random data, a sample of which can be seen in Fig. 6.1. After creating 50 such assets, findings show that each part asset has a size of 41KB on average, assuming all four transactions have been executed for this part and all fields have been filled. This total size per asset includes all the signatures and all other metadata of the transactions.

Presuming a highly demanding scenario where 1000 part assets are added on a daily basis, this would give us approximately 63 years of storage by using just a 1TB drive on network peers. This is higher than the 50 years which was set as a lower threshold, therefore with assets of this size it is perfectly feasible to store all data except the model file on the ledger itself.

Should an organization desire to store a few more KB's of data on an asset, the total storage capacity of each peer can be extended. If however considerably more data have to be tracked per asset, then different approaches can be implemented where only proof of these data are stored on the Blockchain. In that scenario, the real data would be kept on external databases and only references to these data – such as checksum hashes – would be stored on the ledger. Essentially this is the approach that TPTS follows for the model files because of their prohibitive size, where only the hash of the models is included in the transaction.

Lastly it is worth mentioning the pruning function which will be available in the future for HLF [1]. Similarly to Bitcoin's prune functionality, through the pruning process invalid transactions are discarded as they do not contribute in any way to the ledger state. This should help, to a small degree, to maintain the storage requirements to a reasonable level.

6.3 Risks and Limitations

This section documents potential risks or limitations and discusses how those can be dealt with. To begin with, let us take the scenario where a particular user acts maliciously against the Blockchain network. For example this could be a person working for an OEM organization who has access to the certificate which a benign application would normally use to create transactions for parts being ordered. This person abuses this certificate to create assets for invalid orders and floods the system with inaccurate transactions. Given the distributed nature of the network peers it would take a considerable amount of resources to turn this attack into a Denial of Service, so it's expected that the network's nodes would still respond to other valid requests. Nevertheless, the issue is the accumulation of false transactions on nodes. This asset will never manage to be updated past the "Ordered" state, therefore this fabricated order will never be validated by a printshop. However the aggregation the massive forged data would burden the storage of network peers and reduce the system's longevity.

A system operator would therefore want to prevent the malicious user from further harming the system. In this case, Fabric CA allows us to revoke an identity, generate a CRL that contains all certificates of the revoked user and subsequently send a configuration update to the channel so that Member Service Providers are aware of the

[1] http://hyperledgerdocs.readthedocs.io/en/latest/arch-deep-dive.html#post-v1-validated-ledger-and-peerledger-checkpointing-pruning

revoked certificates. Understandably that user would also not be allowed to renew his ABE key, although revoking his certificate would anyhow prevent him from interacting with the Blockchain network in any manner. Naturally a new certificate would have to be created for the benign application to allow it to resume operations.

A very crucial limitation of the system is essentially the human factor. For example, not all 3D printers offer a rich API through which to get appropriate measurements during the printing process. This means that in some cases it would still be required for a human operator to document the printing parameters manually, which in turn could allow for error prone or even malicious behaviour. In turn, this hinders the claim that TPTS *accurately* reflects the real world events. The primary solution to this issue is to integrate the provided client-side code in automated systems with minimal and if possible no human intervention whatsoever.

Chapter 7

Related Work

The previous chapters investigate how to design and implement a system in which data can be shared among different stakeholders in an efficient, reliable, tamper-resistant manner while also maintaining a high degree of privacy in order to prevent unauthorized actors from getting access to that data. In this chapter the contribution of this thesis is compared and contrasted with a number of different works to demonstrate the novelty of TPTS and explain why the approach proposed in this thesis has valuable benefits.

In their work "Towards an Ontology-Driven Blockchain Design for Supply Chain Provenance" [12], the authors Henry M. Kim and Marek Laskowski recognize the value of leveraging the Blockchain technology to track products throughout the supply chain. Their approach is based on the Ethereum platform and ontology-based modeling, where ontology is defined as an explicit specification of a conceptualization. While the asset-based approach is similar to our work, our Trustworthy Process-Tracing System (TPTS) also incorporates privacy for the data that form assets.

Data accountability and provenance tracking is also investigated by Neisse et al. [17]. The authors examine the utility of publicly auditable contracts deployed on a Blockchain, which declare the access and usage of personal data. They implement their work on the Ethereum Platform and propose different models that dictate distinct data usage policies that map to specific data controllers, data subjects or the data itself. They additionally include a provenance tracking model where a list of references is updated every time data is accessed. While in this effort the Blockchain is used solely as a means to control access to data stored off-chain, in TPTS the bulk of data (except the 3D models themselves due to their prohibitive file size of multiple megabytes) are stored on the Blockchain. TPTS also incorporates an alternative form of access policies. This enables

users to dictate a more fine grained control over the permutation of other entities which are authorized to obtain a set of data.

In "Decentralizing Privacy: Using Blockchain to Protect Personal Data" [21], the authors present a system where the Blockchain network acts as an automated access-control manager. Users are allowed to preserve and share encrypted data while controlling the sharing policies as well as being able to instantly completely opt-out of data sharing. However, the system only allows all-or-nothing access policies to be defined. This means that users are not permitted to choose what subset of their data a service is allowed to access. Secondly, the proposed system is designed to store data off the Blockchain on a separate distributed database in order not to overwhelm the Blockchain peers with mass data. Lastly, the retrieval of data from the Blockchain relies on centralized services that process queries.

On the contrary, in TPTS the data is not only stored directly on the Blockchain but the system also provides a mechanism where one can fine tune who has access to a specific subset of transactional data. Additionally, the design presented in this thesis provides a scalable and decentralized method to interact directly with the Blockchain network itself to store and retrieve data. Furtheremore, one additional key feature of the privacy mechanism in TPTS is offering a key escrow arrangement that allows selected auditing authorities to have full access to all transactions without the user having to explicitly permit access to these data.

Healthcare resource sharing and management of Electronic Health Records (EHR) has also been the focus of recent research. Two approaches which exhibit close similarities to our work are hereby examined and compared to TPTS.

First, Barua et. al [4] propose the ESPAC (Efficient and Secure Patient-centric Access Control) scheme for implementing patient-centric access control for Personal Health Information. In ESPAC the data are stored encrypted with CP-ABE on cloud services. There exists a single Trusted Authority (TA) which distributes ABE keys to entities of the system and messages are encrypted and authenticated according to a specified access policy. In this solution patients encrypt data which they submit to health care providers. These providers subsequently store the data on cloud services, where entities can access them upon request.

In the second work in the healthcare area [16] Narayan S. et al describe a similar system where the focus lies on the secure management and storage of EHRs on a cloud service. Similarly to [4], CP-ABE is utilized to associate users with a unique identifier as well as specific attributes and encrypt data with desired access policies. A single TA is responsible for distributing keys, although the authors would also like to explore

alternatives based on multi-authority ABE. Patients add encrypted data on the cloud and subsequently utilize a client application to securely notify a doctor or any other medical service provider about these records. The cloud provider is authorized to view and copy this encrypted information but does not have access to the plain text.

Similar to the mechanism of TPTS for allowing auditors to inspect data, both works support key escrow schemes for permitting record access to selected entities in case of emergencies. Both solutions also provide forward secrecy through an access revocation scheme. This is achievable by manually re-encrypting the target data record with a new access policy which excludes the attributes of the revoked entity.

Despite noticeable similarities concerning how ABE is leveraged to provide an access permission layer, two key differences can be identified between the two aforementioned works and TPTS. Firstly, in TPTS data are stored on a decentralized system which offers complete traceability of all transactions. This means that every time data is altered or processed by the Blockchain nodes in any manner, an immutable record of this activity is documented. One of the main incentives for employing the Blockchain technology in this thesis is the added benefit of tamper-resistant storage, something which is not available in either of the two other solutions described. On the other hand, one shortcoming of TPTS is that having immutable data prevents the implementation of a data access revocation mechanism, while such a scheme does exist in both the above works. Nevertheless, the effects from the lack of access revocation have been mitigated to a certain extent by integrating key issuance dates and allowing actors to define their own requirements for how often they would like private keys to be renewed.

Secondly, TPTS supports multiple coexisting ABE authorities. This decision was taken in order to prevent one central authority to govern all key distribution within the system, as this would likely create unnecessary trust issues from the system's actors towards the central ABE authority. The reason for doing so is that distributing this trust among a number of organizations aligns better with the defined goals for creating a system relying on low-trust.

In conclusion and to the best of our knowledge, our novel contribution is unique in the manner TPTS combines principles of reliable and tamper-resistant storage as well as privacy control for the purpose of supply chain traceability. These offerings are all provided over a decentralized and near-trustless network in which entities can operate. In particular, the ABE privacy mechanism proposed in this system enables fine-grained authorization through precise access policies which additionally permit entities to decrypt the encrypted transactions regardless of the point in time they joined our system.

Chapter 8

Conclusions

In this thesis we have presented the current situation in the industrial supply chains as well as why transparent traceability of items throughout this chain has compelling benefits for the companies, their customers and potential auditors. Additionally, organizations can obtain significant competitive advantages in the area of supply chain management by exploiting technological advances.

For that purpose the Trustworthy Process-Tracing System (TPTS) is presented. It provides reliable and tamper-resistant traceability of items throughout the supply chain while maintaining certain privacy guarantees for the data referencing those items. TPTS allows stakeholders to reflect real world transactions and steps of the supply chain on a distributed ledger. CP-ABE is an essential part of the privacy mechanism used by the system, providing an access layer for authorized stakeholders and auditing authorities to be able to view transactional data. The implementation of this system is particularly focused in the area of 3D printing, but its design could easily be adapted to other industries while its core features would remain untouched.

Evaluation shows that TPTS supports the majority of desired functional requirements. Regrettably the implementation of Hyperledger Fabric currently uses a Kafka-based approach for its ordering service which is lacking BFT consensus mechanism. Nevertheless, a PBFT approach for the ordering of transactions is expected to be developed in the near future which should effortlessly replace the current one.

It is important to note that this thesis simply provides the infrastructure on which to reflect real world transactions. While some information can be reliably retrieved directly from hardware, there are still human actors involved in gathering and storing

data for certain steps. This means that records stored on the ledger could possibly be a misrepresentation of the actual data which were mishandled by such actors.

It is valuable at this point to discuss about the Blockchain technology and whether it actually has significant contributions in practice. Through this work we demonstrate that a Blockchain system which implements the aforementioned requirements is not only feasible, but it indeed exhibits noteable advantages over a traditional database system. However, the Blockchain is often presented as a panacea which can provide a solution to any problem and any use case. This is far from the truth, since systems of this type have their own obstacles to overcome. To elaborate, Blockchain platforms have to balance three factors through their architecture in general:

1. Transaction throughput
2. Adequate security to prevent actors from gaining majority control over the network
3. Decentralized block creation

HLF takes care of the first through its permissioned nature and distinct private channels to reach a throughput of thousands of transactions per second. As for the second factor, HLF prevents entities from controlling the network through the endorsing policies. Regarding the third factor however, the relative centrality of the ordering peers can be an issue since the ordering service controls which transactions go into blocks. One cannot claim that a network controlled by a limited number of these ordering peers is trustless, regardless of which stakeholders are in control of them. On the other hand, the network cannot scale to more than a few dozens of orderers before throughput is impacted negatively. Therefore there is a major trilemma to be considered among these three factors without a clear answer. For now we consider this work to be an acceptable middle ground between complete isolation of data and a full trustless system.

A valuable point of extension for TPTS is the integration of zero-knowledge proofs for the encrypted data. A thorough investigation would have to be carried out to identify which fields could possibly be standardized to make it possible for such methods to perform data validation on encrypted data. Such an addition would be beneficial for the system not only for the purposes of data verification, but it would also allow stakeholders to perform data analytics to a certain degree on these confidential data.

Chapter A

Appendix

A.1 Deployment Configuration

Below we include a shortened version of a deployment configuration file (orgs.json):

```
{
    "domain": "example.com",
    "TLS": true,
    "docker": {
        "fabricTag": "1.1.0",
        "network": "trustNetwork",
        "orderer": {
            "type": "kafka",
            "genesis_block": {
                "file": "trust.block",
                "profile": "trustGenesis"
            },
            "kafka": {
                "zookeepers": {
                    "zookeeper0": {
                        "MY_ID": 0
                    },
                    "zookeeper1": {
                        "MY_ID": 1
                    }
```

```
            },
            "kafkas": {
                "kafka0": {
                    "BROKER_ID": 0
                },
                "kafka1": {
                    "BROKER_ID": 1
                }
            },
            "orderers": {
                "orderer0": {
                    "portMap": {
                        "7050": 7050
                    }
                },
                "orderer1": {
                    "portMap": {
                        "7050": 8050
                    }
                }
            }
        },
        "MSP": {
            "name": "OrdererMSPName",
            "id":   "OrdererMSP"
        }
    },
    "channels": {
        "trustedchannel": {
            "file": "trust.tx",
            "eventWaitTime": 30000,
        }
    },
    "orgs": {
        "org1": {
            "peers": [
                {
                    "container_name": "peer0.org1.example.com",
```

```
            "portMap": [
                {
                    "host": 7051,
                    "container": 7051
                },
                {
                    "host": 7053,
                    "container": 7053
                }
            ]
        },
        {
            "container_name": "peer1.org1.example.com",
            "portMap": [
                {
                    "host": 7061,
                    "container": 7051
                },
                {
                    "host": 7063,
                    "container": 7053
                }
            ]
        }
    ],
    "userCount": 0,
    "ca": {
        "enable": true,
        "admin": {
            "name": "admin",
            "pass": "3sdafuyqghn3io34"
        },
        "portHost": 7054,
        "tlsca": {
            "portHost": 7055
        }
    },
    "MSP": {
```

```
                    "name": "org1",
                    "id": "Org1MSP"
                }
            }
        }
    }
}
```

A.2 Client-Side Configuration

An example of configuration file for the client side:

```
{
    "user": "user2",
    "password": "fw4ac335t",
    "Peers":
        [
            {
                "name": "peer0",
                "org": "org1",
                "address": "example.com:7051",
                "event_address": "example.com:7053"
            },
            {
                "name": "peer1",
                "org": "org1",
                "address": "example.com:7061",
                "event_address": "example.com:7063"
            },
            {
                "name" : "peer0",
                "org" : "org3",
                "address" : "example.com:9051",
                "event_address": "example.com:9053"
            }
        ],
    "Orderers":
        [
```

```
            {
                "name" : "orderer",
                "address" : "example.com:7050"
            }
        ],
    "ca_org": "org1",
    "mspid": "Org1MSP",
    "Fabric_CA": "example.com:7054"
}
```

Chapter B

List of acronyms

ABE	Attribute-Based Encryption.
AM	Additive Manufacturing.
BFT	Byzantine Fault Tolerance.
CP-ABE	Ciphertext-Policy Attribute-Based Encryption.
FCA	Fabric Certificate Authority.
HLF	Hyperledger Fabric.
OEM	Original Equipment Manufacturer.
PL	Packet Loss.

Bibliography

[1] S.A. ABEYRATNE and R.P. MONFARED. "Blockchain ready manufacturing supply chain using distributed ledger." In: *International Journal of Research in Engineering and Technology* (2016), pp. 1–10. ISSN: 2321-7308. DOI: 10.15623/ijret.2016.0509001. URL: https://dspace.lboro.ac.uk/2134/22625.

[2] Joseph A. Akinyele et al. "Charm: a framework for rapidly prototyping cryptosystems". In: *Journal of Cryptographic Engineering* 3.2 (2013), pp. 111–128. ISSN: 2190-8516. DOI: 10.1007/s13389-013-0057-3. URL: https://doi.org/10.1007/s13389-013-0057-3.

[3] Elli Androulaki et al. *Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains*. 2018. eprint: arXiv:1801.10228.

[4] Mrinmoy Barua et al. "ESPAC: Enabling Security and Patient-centric Access Control for eHealth in Cloud Computing". In: *Int. J. Secur. Netw.* 6.2/3 (Nov. 2011), pp. 67–76. ISSN: 1747-8405. DOI: 10.1504/IJSN.2011.043666. URL: http://dx.doi.org/10.1504/IJSN.2011.043666.

[5] Alysson Bessani, João Sousa, and Marko Vukolić. "A Byzantine Fault-tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform". In: *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. SERIAL '17. Las Vegas, Nevada: ACM, 2017, 6:1–6:2. ISBN: 978-1-4503-5173-7. DOI: 10.1145/3152824.3152830. URL: http://doi.acm.org/10.1145/3152824.3152830.

[6] J. Bethencourt, A. Sahai, and B. Waters. "Ciphertext-Policy Attribute-Based Encryption". In: *2007 IEEE Symposium on Security and Privacy (SP '07)*. 2007, pp. 321–334. DOI: 10.1109/SP.2007.11.

[7] Kyle Croman et al. "On Scaling Decentralized Blockchains". In: *Financial Cryptography and Data Security*. Ed. by Jeremy Clark et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 106–125. ISBN: 978-3-662-53357-4.

[8] Primavera De Filippi. “The Interplay between Decentralization and Privacy: The Case of Blockchain Technologies”. In: *Journal of Peer Production* (7: Alternative Internets 2016), pp. 1–10. ISSN: 2321-7308. URL: https://ssrn.com/abstract=2852689.

[9] Vipul Goyal et al. “Attribute-based Encryption for Fine-grained Access Control of Encrypted Data”. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS ’06. Alexandria, Virginia, USA: ACM, 2006, pp. 89–98. ISBN: 1-59593-518-5. DOI: 10.1145/1180405.1180418. URL: http://doi.acm.org/10.1145/1180405.1180418.

[10] Stuart Haber and W. Scott Stornetta. “How to time-stamp a digital document”. In: *Journal of Cryptology* 3.2 (1991), pp. 99–111. ISSN: 1432-1378. DOI: 10.1007/BF00196791. URL: https://doi.org/10.1007/BF00196791.

[11] Don Johnson, Alfred Menezes, and Scott Vanstone. “The Elliptic Curve Digital Signature Algorithm (ECDSA)”. In: *International Journal of Information Security* 1.1 (2001), pp. 36–63. ISSN: 1615-5262. DOI: 10.1007/s102070100002. URL: https://doi.org/10.1007/s102070100002.

[12] Henry M. Kim and Marek Laskowski. “Towards an Ontology-Driven Blockchain Design for Supply Chain Provenance”. In: *CoRR* abs/1610.02922 (2016). arXiv: 1610.02922. URL: http://arxiv.org/abs/1610.02922.

[13] A. Kosba et al. “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts”. In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, pp. 839–858. DOI: 10.1109/SP.2016.55.

[14] Douglas M. Lambert, Martha C. Cooper, and Janus D. Pagh. “Supply Chain Management: Implementation Issues and Research Opportunities”. In: *The International Journal of Logistics Management* 9.2 (1998), pp. 1–20. DOI: 10.1108/09574099810805807. eprint: https://doi.org/10.1108/09574099810805807. URL: https://doi.org/10.1108/09574099810805807.

[15] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine Generals Problem”. In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401. ISSN: 0164-0925. DOI: 10.1145/357172.357176. URL: http://doi.acm.org/10.1145/357172.357176.

[16] Shivaramakrishnan Narayan, Martin Gagné, and Reihaneh Safavi-Naini. “Privacy Preserving EHR System Using Attribute-based Infrastructure”. In: *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*. CCSW ’10. Chicago, Illinois, USA: ACM, 2010, pp. 47–52. ISBN: 978-1-4503-0089-6. DOI: 10.1145/1866835.1866845. URL: http://doi.acm.org/10.1145/1866835.1866845.

[17] Ricardo Neisse, Gary Steri, and Igor Nai-Fovino. “A Blockchain-based Approach for Data Accountability and Provenance Tracking”. In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ARES ’17. Reggio Calabria, Italy: ACM, 2017, 14:1–14:10. ISBN: 978-1-4503-5257-4. DOI: 10.1145/3098954.3098958. URL: http://doi.acm.org/10.1145/3098954.3098958.

[18] Phillip Rogaway and Thomas Shrimpton. “Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance”. In: *Fast Software Encryption*. Ed. by Bimal Roy and Willi Meier. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 371–388. ISBN: 978-3-540-25937-4.

[19] Feng Tian. “An agri-food supply chain traceability system for China based on RFID blockchain technology”. In: *2016 13th International Conference on Service Systems and Service Management (ICSSSM)*. 2016, pp. 1–6. DOI: 10.1109/ICSSSM.2016.7538424.

[20] Marko Vukolić. “The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication”. In: *Open Problems in Network Security*. Ed. by Jan Camenisch and Doğan Kesdoğan. Cham: Springer International Publishing, 2016, pp. 112–125. ISBN: 978-3-319-39028-4.

[21] Guy Zyskind, Oz Nathan, and Alex ’Sandy’ Pentland. “Decentralizing Privacy: Using Blockchain to Protect Personal Data”. In: *Proceedings of the 2015 IEEE Security and Privacy Workshops*. SPW ’15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 180–184. ISBN: 978-1-4799-9933-0. DOI: 10.1109/SPW.2015.27. URL: http://dx.doi.org/10.1109/SPW.2015.27.