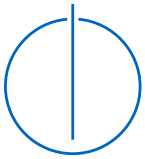
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

# Analysis of Business Applications Integrating Distributed Ledger Technologies

Christian Ziegler



DEPARTMENT OF INFORMATICS

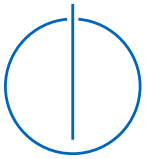
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

# Analysis of Business Applications Integrating Distributed Ledger Technologies

# Analyse von Geschäftsanwendungen mit Distributed Ledger Technologies

| | |
|---|---|
| Author: | Christian Ziegler |
| Supervisor: | Professor Dr. Florian Matthes |
| Advisor: | Ulrich Gallersdörfer, M.Sc. |
| Submission Date: | February 15th, 2021 |



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, February 15th, 2021 Christian Ziegler

# Acknowledgments

Primarily, I want to thank Ulrich Gallerdsörfer for supporting and advising me during this thesis and in the semesters before. His Blockchain-based Systems Engineering exercises gave me the fundamentals I was missing even though I thought I knew everything to know about Bitcoin and Ethereum. The Lab Course "Web Applications" where we created a student wallet that had to be non-custodial taught me that decentralization sometimes is the better way, even if it is the harder one. Finally, the work as his student associate has provided me with new experiences that I would not have received anywhere else. All of this has been invaluable to me. *Thank you, Uli*.

Secondarily, I want to thank all the great interview partners that took the time to discuss their use-cases with me. Namely: Steven Pannell, Christoph Langewisch, Marco Barulli, Andreas Zeiselmair, Felix Gerbig, Maximilian Forster, Marijo Radman, Dennis Zimmer, Erdi Dogruel, and Jeroen Breteler.

And lastly, ML for - until today - almost two not easy, but wonderful years in Munich.

# Abstract

Since the introduction of Bitcoin through *Bitcoin: A Peer-to-Peer Electronic Cash System in 2008* [24] there has been ongoing development around distributed ledger technology. Five years later, in 2013, Vitalik Buterin [6] published the Ethereum Whitepaper, which in its core describes a next-generation smart contract and decentralized application platform. Four years later, in 2017, the Linux Foundation released Hyperledger Fabric 1.0 [20] the first major platform utilizing distributed ledger technology aimed at private companies.

In 2020 there are more than 7000 active Cryptocurrency projects active [9]. Millions of smart contracts have been deployed on the Ethereum Blockchain [12], and about 250 companies collaborate on the Hyperledger project [19]. From these statistics, a huge interest in the technology world can be assumed for distributed ledger technology.

This work looks at features and functionality that are frequently used within distributed ledger applications and provides them summarized in a structured form. This thesis aims to structure and bundle research and development efforts made to distributed ledger technology allowing developers to follow them when they create new distributed ledger applications instead of re-inventing already existing features and workflows. Furthermore, structured use-case studies are presented that show decisions made in the development of distributed ledger applications and explain why they were made to help guide new developers in their process of developing. The use case studies are also used to verify the presented feature summary.

This thesis introduces the fundamentals of distributed ledger technologies on the example of Bitcoin and Ethereum representing public permissionless ledgers, and Hyperledger representing private permissioned ledgers. Following an overview of distributed ledger features in general and business applications in this context are given. Furthermore, related work is presented, and differences to this thesis are made clear. However, the core is the *eight* structured use-case studies that each describe a current distributed ledger project, its decisions, and features. Additionally, a feature summary divided into 12 categories that each have their related features is given. Each feature is also validated by the interviews that are conducted for the use-case studies.

# Contents

# 1. Introduction

Computing has started centralized with massive monoliths such as the IBM 701 dating back to 1952 [11]. However, centralized systems have significant downsides, such as being a single point of failure and not being able to scale horizontally. These two downsides are especially harmful to databases as they need to persist millions or even billions of records. Therefore, the first ideas for distributed databases were published in 1980 [32] that tried to eliminate monoliths' downsides. Distributed Systems, in general, have the advantage of having no single point of failure. So when one node crashes, the network continues without the failed node. Additionally, distributed systems can scale horizontal and vertical. However, centralized systems have their advantages, too, such as being easier to coordinate and implement. These benefits, however, can be disregarded in a system that must scale.

A Ledger describes a database in which no record can be edited after it has been written in. It can be seen as a family register or a land register where a new member of the family is added, or new owners to land are added while the old records remain there. This approach ensures immutability, one of the key properties of distributed ledger technology.

The term distributed ledger technology is often interchangeably used with the term Blockchain. Distributed ledger technology is the term that is more used in science, and Blockchain is more commonly used in common speech. Technically speaking, there is a difference between distributed ledger technology and Blockchain. A distributed ledger keeps the records immutable in many different places, and so does a Blockchain. Withal, a Blockchain also requires data to be put inside containers called blocks that are connected in a chain with the hash of the previous block that needs to be contained in the following block. Various proofing algorithms then confirm these blocks explained in the Fundamentals chapter of this work.

Distributed ledger technology is critical in business applications that need to store data immutable and in business applications used inside a consortium. For example, in a company network where many suppliers and manufacturers collaborate with many different contracts, the company that owns the network has a knowledge advantage over all other participants. In a time where data security has become a central topic in daily life, such a system is no longer sustainable unless there is lots of trust between the participants or the bigger companies push such a system upon smaller companies that can not fight back. Therefore many companies are now pushing towards distributed instead of centralized systems that are made in collaboration inside or outside of the consortium.

Companies seemingly have a very high interest in these kinds of technologies as more

than 250 companies are part of the Hyperledger Foundation to date [19]. However, for most developers, this is the first time they are working with distributed ledger technologies. Pilot projects are started, and developers tend to either use Hyperledger Fabric resulting in lots of similar but different projects or create new platforms such as Hyperledger Besu, Borrow, Indy, Iroha, and sawtooth. Since both approaches are far from ideal, this work proposes feature summaries intended to help developers pick their needed features and hold their hands making architecture decisions for their new application.

The summaries proposed in this work are feature-based and meant to be used in a modular fashion. Every category of currently in a distributed ledger or Blockchain applications implemented features is represented and structured to achieve the goal of this work: Ease the introduction of distributed ledger-based applications to the enterprise sector and improve the collaboration between companies to increase the overall productivity of the economy.

## 1.1. Motivation

Distributed ledger technology and Blockchain technology commonly used interchangeably as described in the introduction is mostly seen very different in real-world usage. Where Blockchain Technology is almost exclusively used in applications that utilize a permissionless ledger, a ledger in which unknown and distrusted parties can collaborate on a trusted basis. Distributed ledger technology uses permissioned ledgers instead where all participants are known.

Distributed Ledger Technology (DLT) started to receive academic attention in 2016 with 18 papers available on Scopus, 3 on ACM, and 5 publications on Web of Science. 958 papers with the publication year of 2019 are available on Scopus, 71 on ACM, and 394 on Web of Science. This shows a large interest of the academic community in this topic.

In 1.1 we display the combined Blockchain and Distributed Ledger Technology related publications from the mayor platform Scopus, Web of Science, and ACM. The chart displays a much higher interest in the term Blockchain than Distributed Ledger Technology.

Blockchain technology experienced an immense increase in popularity since the rise of Bitcoin in 2009 [24]. The combined market cap of all Cryptocurrencies is $339,86B with a Bitcoin dominance of 58,3% (06.10.2020 [10]). Additionally, 5818 papers about Blockchain with a publication date of 2019 are available on Scopus, 822 on ACM, and 2867 on Web of Science. This shows that the general interest in the particular specification of Blockchain within the distributed ledger technologies is more interesting to researchers and common people than the broader topic of distributed ledger technologies.

In this work, the characteristic of distributed ledger technology is neglected, and all projects that contain any specification of distributed ledger technologies are considered for their features and distilled in feature summaries. Those play an important part in

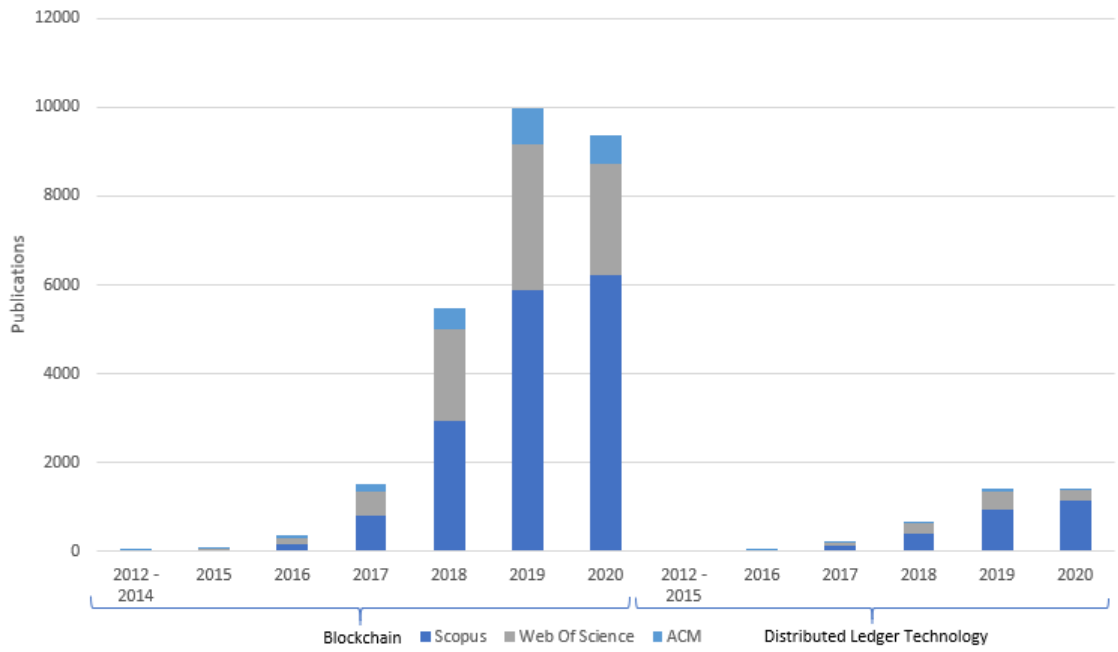



Figure 1.1.: Publications of Blockchain and DLT related work

crafting complex architecture for computing systems, in the specific case of this work, crafting complex distributed ledger architecture with numerous features and use-cases. This allows developer teams that tend to "re-invent the wheel" to build on proven and widespread features for the specific use-case.

Development processes at companies are most of the time closed source, and additionally, not many companies are currently developing new distributed ledger technology due to the immaturity of the technology. Therefore structured use-case studies are very sparse. Thus the core reason for presenting them.

## 1.2. Research Questions

This work aims to find common features in distributed ledger projects and structure them into feature summaries. The desired outcome for this work is the finished feature summary catalog divided into their respective categories. To achieve this goal, five research questions are given as a guideline:

**Q1 - What are common features that distributed ledger applications share?**

a) Which applications that use distributed ledger technology are being used and created at the moment?

b) Which distributed ledger technology do these applications use?

c) Is there related work regarding patterns for distributed ledger technology?

**Q2 - How can these features be categorized and formalized?**

a) Which technologies are used most and how are they implemented?

b) Which different categories do the features fit in?

c) How can these features be summarized to understand for no DLT savvy engineer?

**Q3 - Which decisions are made in the current DLT developments?**

a) Which problems are the projects that are currently in development solving?

b) What are re-occurring questions in the architecture of enterprise DLT applications?

**Q4 - How can a feature summary be verified?**

a) Which technology features of the catalog are implemented in business applications?

b) Are there features used in nearly every application and rare features that are only used in a limited number of applications?

c) Can the relevance of the features be derived from the use-case studies?

**Q5 - What are characteristics for proper DLT applications?**

a) Why is DLT needed for solving the specific issues?

b) Could DLT Projects also be realized without DLT and still solve the same issues?

c) Would it be possible that projects are easier to implement and execute without DLT while still solving the same issues?

## 1.3. Approach

The presented work uses the following approach to answer the research questions:

- **Q1 - Literature review and interviews**
  **Q1.a** - The goal of the first research question Q1.a in 1.2 is to find the most popular and active (open source) projects in the distributed ledger and Blockchain technology space. This is achieved by looking at the market and following the trends. In a first step, Google Scholar is searched for *DLT, Distributed ledger technology, blockchain* and the result is sorted by new. The papers are then summarized. Additionally the top 10 Cryptocurrency projects are extracted from CoinMarketCap[1] and also summarized. Lastly, Everest.link[2] and DEFI PULSE[3].

[1] https://coinmarketcap.com/ - CoinMarketCap
[2] https://everest.link/ - Universally shared projects registry
[3] https://defipulse.com/ - DEFI PULSE

**Q1.b** - The goal of this research question is to identify technologies and features that are frequently integrated within distributed ledger applications. This goal is achieved by extracting the features and their use-cases from the summaries written in Q1.a.
**Q1.c** - This research question aims to find related work, so nothing duplicated is presented as new work in the final research result. A secondary goal is to find existing patterns. This achieved by searching common databases for the search terms *DLT, Distributed ledger technology, blockchain*. The databases used are Scopus[4], ACM[5] and Web of Science[6].

- **Q2 - Categorization and formalization of features**
  **Q2.a** - The goal of this research question is to create a list of the most used distributed ledger technology features and, if possible, implementation details for each feature. This is achieved by counting the features that were worked out in Q1.b.
  **Q2.b** - This research question aims to provide feature categories that fit the features. This is achieved by literature review and feedback from industry experts, which is acquired through interviews.
  **Q2.c** - This research question aims to create easy to understand summaries of all categories and features. This is achieved with a literature review.

- **Q3 - Decisions made in DLT developments**
  **Q3.a** - This research question aims to find enterprise DLT projects that are currently in development and find out which problems they are solving and which decisions are made while solving the problems. This is achieved by conducting interviews with leading members of current DLT projects. The results are presented in a structured use-case study form.
  **Q3.b** - This research question aims to find re-occurring structures and decisions in the use-case studies. This is achieved by generalizing the use-cases and finding common ground.

- **Q4 - Verification of the feature summary**
  **Q4.a** - This research question aims to find out which features of the summary are implemented in the applications of the use-case studies. This goal is achieved by counting the features of the use-case studies.
  **Q4.b** - This research question aims to find accumulations of features by counting the features of the use-case study using the feature summary.
  **Q4.c** - This research question aims to derive the relevance of each feature from the feature summary.

- **Q5 - Characteristics of proper DLT applications**

[4]https://www.scopus.com/ - Scopus
[5]https://dl.acm.org/ - ACM Digital Library
[6]https://webofknowledge.com/ - Web of Science

**Q5.a** - The goal of this research question is to find the essential answer to the questions "Why is DLT needed?" for every use-case study. This is achieved by asking this question in the interview process.
**Q5.b** - The goal of this research question is to find out if projects of the use-case study could be realized without DLT. This goal is achieved with literature review and asking this question in the interviews.
**Q5.c** - The goal of this research question is to find out if projects would have been easier implemented without DLT. This is achieved by discussing this question with the interviewees.

## 1.4. Outline

This work starts by explaining the fundamentals of public and private ledgers on the example of Bitcoin and Hyperledger Fabric, respectively, and then goes over to defining Business Applications in the context of this work.

Next, we present related work that classifies Blockchain-based applications and present an existing pattern catalog where we summarize the most important pattern.

After that, we present feature summaries for 12 functional and three non-functional feature categories, which each have their own sub-features that each have brief explanations.

Then we provide eight use-case studies that were created by interviewing industry experts about their current projects.

In the end, we draw a summary and evaluate this work by answering the research question. After that, we describe what future work can be done and conclude this work.

# 2. Fundamentals

This chapter is designed to introduce distributed ledger technologies and use-cases for distributed ledger technologies of business applications to the technology-savvy user unfamiliar with DLT. Each section starts with defining essential terms that are commonly used with the section's topic and then goes over to explaining their uses on examples.

## 2.1. Distributed Ledger Technologies

Generally speaking, distributed ledgers can be divided into the two main categories *public* or *permissionless ledger* and *private* or *permissioned ledger*.

However, some authors make a distinction between *permissionless* and *public* or *permissioned* and *private*. For example PETERS and PANAYI [25] define it as follows:

- **Permissionless**: Anyone can participate in the verification process without authorization.
- **Permissioned**: A central authority or consortium preselects verification nodes.
- **Public**: Anyone can read and submit transactions.
- **Private**: Only users or groups with the correct permission can read and submit transactions.

This work tries to weaken the boundaries between the hard categorizations and focuses more on the feature side of Blockchains and where those features are more commonly used. Therefore we do not make a hard distinction between the two different categories. Nonetheless, we acknowledge the possibility of a *permissionless ledger* being *private* although, from a technical and business perspective, this combination does not yield practical results.

In a *public permissionless ledger* everyone can join the network and interact with it without revealing his real identity. Popular examples for this ledger type are Bitcoin[1], Ethereum[2] and all Cryptocurrencies that build on top of those. Public permissionless ledgers often have a native Cryptocurrency and most of the time use consensus mechanism such as *Proof-Of-Work* or *Proof-Of-Stake* which are explained in more detail in this chapter [2].

[1] https://bitcoin.org - Bitcoin
[2] https://ethereum.org - Ethereum

*Public permissionless ledgers* are designed for verifiable interactions between unknown entities and distrusted entities. *permissioned ledgers* require every participant of the network to be identifiable. Without authentication, the user can not join the network nor interact with it. Private permissioned ledgers are designed for trusted interactions between known distrusted participants [2]. Commonly known examples for private permissioned ledger software are Hyperledger[3] and Corda[4].

In this context, Blockchain Technology is also important because it is common and, most of the time, used in an invalid way for distributed ledger technology. Blockchain Technology is almost always used in public permissionless ledgers and is a more narrow implementation. In a Blockchain-based system, transactions are bundled in blocks and confirmed with a consensus algorithm.

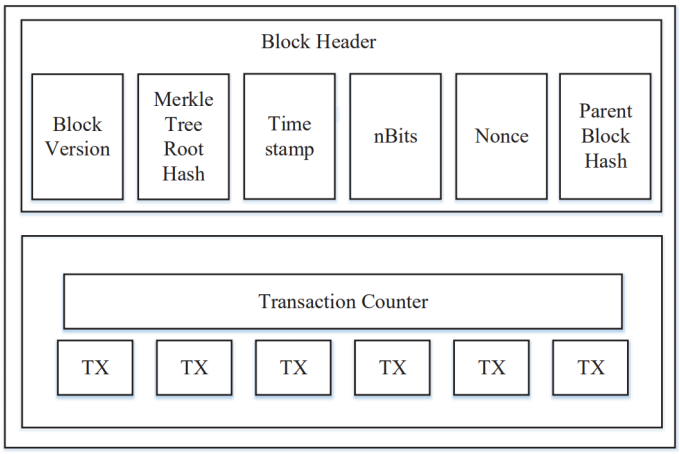



Figure 2.1.: Block structure [39]

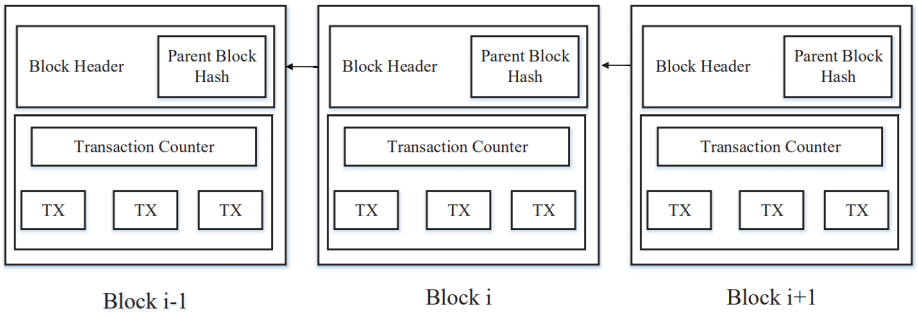



Figure 2.2.: Blockchain Architecture [39]

Figure 2.1 depicts an exemplary block structure, and figure 2.2 shows how a sequence of blocks is connected through each previous block hash.

[3]https://hyperledger.org/use/fabric - Hyperledger Fabric
[4]https://corda.net - Corda

In the following subsection, public permissionless ledgers are explained in great detail on the example of the Blockchain Technology used by Bitcoin.

### 2.1.1. Public Ledgers on the Example of Bitcoin

This whole section uses ANTONOPOULOS [3] as a knowledge basis to explain how Bitcoin works and should be consulted for deeper understanding. The main point of the section is to introduce public permissionless ledgers and their features. ANTONOPOULOS describes Bitcoin as

> a collection of concepts and technologies that form the basis of a digital money ecosystem. Units of currency called bitcoins are used to store and transmit value among participants in the bitcoin network. Bitcoin users communicate with each other using the bitcoin protocol primarily via the Internet, although other transport networks can also be used. The bitcoin protocol stack, available as open-source software, can be run on a wide range of computing devices, including laptops and smartphones, making the technology easily accessible [3]

Bitcoin is the most commonly used example for public permissionless ledgers because the field emerged starting with the publication *Bitcoin: A Peer-to-Peer Electronic Cash System* [24]. Furthermore, Bitcoin implements all common features of a permissionless ledger. More specifically, Bitcoin is a Blockchain as it utilizes blocks to bundle transactions and uses a Proof-Of-Work algorithm.

In this section, Bitcoin is the name of the permissionless ledger that is explained, and bitcoins refer to the Cryptocurrency that can be used for payments.

Bitcoin has four main features: *Identity Management*, *Cryptocurrency Storing*, *Cryptocurrency Transfer* and *Minting Cryptocurrency*. These four features are shared among all Cryptocurrencies. The following chapters will also break down its technologies in their core features and take a closer look at the features themselves and not the product they are bundled as.

In the Bitcoin Blockchain Network, all participants are unknown by default, and the participants have a strong distrust of all other participants. Bitcoin uses asymmetric key pairs for **Identity Management**. The general functionality of a key pair is the same as for an RSA key pair. There are a public-key and a private key, which are derived from the public-key. In bitcoin, the public-key is used as a hash as public address - the bitcoin address - where other participants of the network can send bitcoin to. The private key is used to identify the owner of the bitcoins and can be compared to something (less complex) like a pin code. All key pairs together owned by a participant are bundled in his wallet.

To receive bitcoins in the Bitcoin Blockchain Network, a participant can either try to solve a mining puzzle to receive the mining reward or get a transfer from another participant. Through mining, new Blocks are created for the Bitcoin Blockchain. It can be compared to the central bank introducing new money to the system. To establish

trust, Bitcoin uses mining, ensuring that the network only accepts transactions if enough computational power is devoted to the blocks containing them. This concept is more commonly known as Proof-Of-Work - the computational power is the work - and can be categorized under **Minting Cryptocurrency**.

Commonly, monetary assets such as Euros or Dollars are stored in bank accounts. Each bank account has an owner and address to transfer to it. In Europe, IBAN is used as a uniquely identifiable address. Ownership in Bitcoin, however, is handled via the before described Identity Management. Before we can understand how bitcoins are actually stored, we need to look at how transactions work in Bitcoin. When bitcoins are mined, the miner receives bitcoins in the form of an unspent transaction output (UTXO). The miner is then allowed to spend this UTXO, meaning that he could now create a transaction to send bitcoins to participants of the network. For example, he can send half of this UTXO to Bob and half of it to Alice. Now two UTXO exist that Bob and Alice can spend. In Bitcoin, the balance available on a wallet is the sum of all UTXOs that can be spent using the private key, which is in the collection of a network participant's wallet. Bitcoin, therefore, uses a UTXO pattern for **Cryptocurrency storing**.

Bitcoins can be transferred between participants in the Bitcoin Network. To send bitcoins, the sender needs to draft a transaction containing a reference to a UTXO owned by the sender. This transaction then needs to be signed by the sender with its private key, creating the signature. The whole transaction is then verified by the network, which checks if the signature matches the input values. Bitcoin also offers the possibility to use multiple signatures for a transaction. For example, a participant can define that at least two out of three signatures are required to spend the bitcoins. Summarizing, Bitcoin offers the possibility of sending bitcoins using a single signature and multiple signatures. Both features can be combined under the more general category **Cryptocurrency transfer**.

This section has explained Bitcoin's core features and showed that breaking down Bitcoin in its core features is very much possible. The following subsection will do the same with private ledgers on the example of Hyperledger Fabric.

### 2.1.2. Private Ledgers on the Example of Hyperledger Fabric

Hyperledger Fabric is a platform for distributed private permissioned ledger applications that are supported by its modular architecture. Furthermore, Hyperledger Fabric offers a high degree of reliability, flexibility, and scalability. It is one of the most well-known platforms for this ledger type and receives continuous updates and security patches, and general support from developers. Fabric supports modular consensus protocols and chain code. The Hyperledger Fabric platform is part of the Hyperledger ecosystem managed by the Linux Foundation and is supported by many well-known industry partners such as Accenture, Airbus, IBM, Daimler, and J.P.Morgan [13][2].

In a private permissioned ledger, all participants are known and have to register to participate in the network. However, there is still a general distrust between the participants.

The features of Hyperledger Fabric can be broken down in five categories: *Identity Management*, *Information Broadcasting*, *Unique Asset Transfer*, *Unique Asset Management* and *Code Execution on the Blockchain*.

**Identity Management** in Fabric is managed by the Membership Service Provider (MSP). The MSP provides a digital identity for every actor of the network in the form of a X.509 digital certificate. Each actor has its own unique identity with its own exact permissions over resources and access to information in the blockchain network [16].

Another feature of Fabric is **Information Broadcasting** which is implemented through Events and Event Listeners. A short example of how this can be used is the following: A vendor (vendor1) is in a network with other vendors and has 1000 Items replicated on the chain. The vendor now changes the price of one of the items. Another vendor (vendor2) waited for the price of the item to drop to buy it. So vendor1 emits a price change event, and vendor2 gets the event because he has set an event listener on this change [16].

To replicate an item on the network, the network needs to be capable of representing assets. Fabric is capable of **Unique Asset Management** meaning that any item can be represented and managed in Fabric. For example, a diamond can be uniquely described with the properties clarity, karat, weight, size, and serial. All these properties get written into an object and then stored on the ledger. Additionally, the object's owner can be stored using the owner's MSP Id to re-create ownership on the chain [16].

Furthermore, Fabric supports **Unique Asset Transfering**. Meaning that the ownership of any asset can be changed at any time - or - the ownership of an item can be changed automatically following rules set and agreed on by the parties beforehand.

To change the ownership of an item, a chain code is required. Chaincode is basically a program that runs on-chain and whose source code is available to all parties to look at. For example, chain code can be set up to exchange X of item A for Y of item B. The advantage of chain code is that no intermediary is needed to supervise a contract between two parties. In simple terms, all parties of the permissioned distributed ledger execute the same transaction, and only if enough come to the same result, it is written to the ledger. Speaking more generally, this is known as **Code Execution on the Blockchain**.

### 2.1.3. Comparison between public and private Ledgers

In the two subsections before, public and private ledgers were introduced on the example of Bitcoin and Fabric. In this section, this work looks at their similarities and differences.

Before this work goes into the comparison, it is important to note that this section looks at public and private ledgers' typical implementation. In theory, it is possible that a private ledger implements a Proof-Of-Work approach for creating new blocks in the Blockchain or that a public ledger does not implement a mining reward for creating blocks. However, speaking in practical terms, this has very little use and is not used in any use-case this work has looked at.

The incentive for joining a distributed network is always monetary gain. This gain can

be achieved by simplifying an existing workflow or direct in the form of Cryptocurrency. Public Ledgers commonly offer compensation in the form of Cryptocurrency, whereas private Ledgers improve workflows between participants.

Most public ledger implementations use the Proof-Of-Work consensus algorithms such as Bitcoin, Litecoin, and Ethereum. Whereas private ledgers mostly use consensus algorithms that do not require a massive amount of hashing power. Examples of consensus algorithms used in private ledgers are Crash Fault Tolerance (CFT) in Hyperledger Fabric or Proof-Of-Authority in Hyperledger BESU.

Generally speaking, public ledgers implement Cryptocurrency features such as storing and transfer. On the other hand, private ledgers rarely implement Cryptocurrency features by default and concentrate more on Asset features such as management and transfer and code execution on the Blockchain.

## 2.2. Business Applications

In this work, we propose three categories to divide distributed ledger applications in. With this categorization, we justify the focus in the case-studies on one of the categories instead of having a more broad selection of case-studies.

- **Cryptocurrencies** are distributed ledger applications that serve the main purpose of creating a digital currency with most of the prominent Cryptocurrencies all implementing **Cryptocurrency Storing**, **Cryptocurrency transfer** and **Minting Cryptocurrency**. Examples for Cryptocurrencies that fit this description are Bitcoin[5] and Tether[6]. The popularity of the Cryptocurrencies was assessed using their marketcap[7]. Cryptocurrencies are almost always Blockchains in their core. Furthermore, DeFi Pulse[8] categorizes so-called decentralized finance projects (DeFi) into five categories. Following, each category is named, and the top project for each category is named and briefly described using the project description on DeFi Pulse and the respective website of the project. The top project is assessed by marketcap as of 12.09.2020 – 16:47. These categories are listed because of each projects unique feature in addition to the base features mentioned before. In general, the specific projects are not important, but their ideas are.
  - **Lending**: *Aave*[9] allows for decentralized lending and borrowing. Aave offers flash loans: this means that borrowing and repayment must occur in the same transaction. [27]
  - **DEXES**: *Curve Finance*[10] is a decentralized exchange (DEX) for extremely

[5]https://bitcoin.org/de/
[6]https://tether.to/
[7]https://coinmarketcap.com/
[8]https://defipulse.com/
[9]https://aave.com/
[10]https://www.curve.fi/

efficient stablecoin trading. Allow trading between stablecoins with low fees. [28]

- **Derivates**: *Synthetix*[11] allows for the creation of Synths: on-chain synthetic assets that track the value of real-world goods. The platform supports Synths representing fiat currencies, commodities (e.g. gold), and cryptoassets. [30]
- **Payments**: *Flexa*[12] is an instant payments network for digital assets. It allows users to spend selected Cryptocurrencies instantly without fees. The underlying Flexcoin collateral token allows for the underlying digital asset transaction to confirm and settle on chain. [29]
- **Assets**: *yearn.finance*[13] "is a decentralized ecosystem of aggregators that utilize lending services such as Aave, Compound, etc. to optimize token lending. When tokens are deposited to yearn.finance they are converted to yTokens, which are perodically rebalanced to choose the most profitable lending services. Curve.fi is a prominent integrator of yTokens – creating and AMM (Automated Market Maker) between yDAI, yUSDC, etc. that not only earns the lending fees back but also the trading fees on Curve.fi. YFI, yearn.finance's governance token is distributed to users who provide liquidity with certain yTokens" [31].

- **DLT-based Business Problem Solvers** are designed with solving real-world use-cases in mind. Most DLT-based Business Problem Solvers either bring together lots of actors that interact with each other, store assets or data on the ledger that many actors need to access, or enable a digital process that required an intermediary to work on the ledger without that intermediary. These DLT-based Business Problem Solvers are almost always built on top of an existing distributed ledger platform and therefore utilize DLT.

- **Ecosystem Software** has an architecture that improves existing Cryptocurrencies or DLT-based Business Problem Solvers by, for example, improving the accessibility of Wallets, automate some of their functions or combine different DLT-based Business Problem Solvers to allow for **Interoperability**. The list for Ecosystem Software is not finite as any piece of software that utilizes either Cryptocurrencies or DLT-based Business Problem Solvers to create value for the customer is categorized as Ecosystem Software.

In this work, we only look at the category **DLT-based Business Problem Solvers** as those are most relevant for economic use-cases. Ecosystem Software in its core is not DLT; therefore, it is not looked at yet. Cryptocurrencies have been deemed as out of scope. Additionally, use-cases for Cryptocurrency are very similar.

[11] https://www.synthetix.io/
[12] https://app.flexa.network/
[13] https://yearn.finance/

# 3. Related Work

Since this thesis's core is to analyze business applications integrating distributed ledger technology, the presented related work primarily discusses Blockchain and DLT use-cases and classification; However, since the final goal of this research is to find generally applicable pattern for DLT, some related work, that already achieved universal patterns for DLT is also included.

## 3.1. A systematic literature review of blockchain-based applications: Current status, classification and open issues

CASINO [8] performs a systematic literature review on blockchain-based applications. In the first table, he describes the main characteristics and classifies Blockchain networks. Originally, he included the Blockchain category "Federated", but it is not presented as this category is not part of this thesis. The presented table includes the two Blockchain categories discussed in the previous chapters of this work: "Public" and "Private".

| Property | Public | Private |
|---|---|---|
| Consensus Mechanism | • Costly PoW<br>• All miners | • Light PoW<br>• Centralised organisation |
| Identity Anonymity | • (Pseudo) Anonymous<br>• Malicious? | • Identified users<br>• Trusted |
| Protocol Efficiency & Consumption | • Low efficiency<br>• High energy | • High efficiency<br>• Low energy |
| Immutability | • Almost impossible | • Collusion attacks |
| Ownership & Management | • Public<br>• Permissionless | • Centralised<br>• Permissioned whitelist |
| Transaction Approval | • Order of minutes | • Order of milliseconds |

Table 3.1.: Classification and main characteristics of blockchain networks [8]

Furthermore, CASINO [8] depicts a mind-map of different types of Blockchain-Applications A.1 (appendix). The primary categories are privacy and security, business and industry, data management, financial, integrity verification, governance, the internet of things, health, and education. These categories will be used to classify the use-cases presented later in this work.

Moreover, CASINO [8] gives an analysis of attributes and prerequisites of Blockchain

versus traditional databases A.1 (appendix). He compares the Architectures Permissionless, Permissioned, and Database. The most important takeaway is that Permissionless Blockchain architectures offer reduced privacy but higher latency and transaction speed, and better Scalability, compared to Permissionless systems. Database neither offers trust nor transparency, security, privacy, or consensus but is better in scalability and latency, and transaction speed.

## 3.2. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends

ZHENG [39] presents a comparison similar to 3.1 and A.1. Representing Public, Private, and Consortium Blockchains with overall similar results.

| Property | Public blockchain | Consortium blockchain | Private blockchain |
|---|---|---|---|
| **Consensus determination** | All miners | Selected set of nodes | One organization |
| **Read permission** | Public | Could be public or restricted | Could be public or restricted |
| **Immutability** | Nearly impossible to tamper | Could be tampered | Could be tampered |
| **Efficiency** | Low | High | High |
| **Centralized** | No | Partial | Yes |
| **Consensus process** | Permissionless | Permissioned | Permissioned |

Table 3.2.: Comparisons among public blockchain, consortium blockchain and private blockchain [39]

## 3.3. A Pattern Collection for Blockchain-based Applications

While CASINO and ZHENG performed a more general review of Distributed Ledger Systems, XU [38] presents pattern on a more technical and feature heavy basis. He presents four categories, each with its own pattern: Interactions with external world patterns, Data Management, Security, and Structural Patterns of Contract.

XU's work is interesting for this thesis since all patterns require a specific feature set of the Blockchain so that they can work. In this thesis, we present a feature catalog that is intended to become a pattern catalog ultimately, and the work of XU is somewhere in-between those two.

He presents the Oracle, Reverse Oracle, and Legal and smart contract pair pattern for the Interactions with external world patterns. Where the Oracle pattern brings external data into the Blockchain system, and the Reverse Oracle feeds external systems with Blockchain data. The Legal and smart contract pair connects a Smart Contract to a legal contract.

Within the Data Management Category, four patterns are presented: Encrypting on-chain data where the data on the blockchain is encrypted, Tokenisation that uses tokens to represent assets on the Blockchain, Off-chain data storage that utilizes the

hashes of files to store the file on an external database and the hash on the blockchain to prevent bloating of the chain while maintaining immutability and lastly the State channel pattern which utilizes side-chains to process small transactions and ultimately settles them in a condensed transaction on the main chain.

Three patterns are explained in the Security category: Multiple authorizations, Off-chain secret enabled dynamic authorization and X-confirmation. The first is basically a Multi-Signature send in which at least X out of Y participants have to authorize a transaction; the second uses a hash that is created off-chain to binding authority for a transaction dynamically. The last pattern is about waiting for enough confirmations until a transaction output can be used repeatedly until the transaction is viewed as confirmed by the receiving system.

The last category that Xu presents is the Structural Patterns of Contracts category. In this category, he presents five patterns:

1. The Contract registry, which looks up the address of the latest version of the smart contract on a contract registry by name.
2. The Embedded permission pattern is used to restrict the access to certain functions defined in the smart contract.
3. A Data contract defines that the data normally stored in the same contract is stored differently.
4. The Factory contract represents an on-chain template contract used as a factory to generate contract instances from itself.
5. Incentive execution rewards the caller of a contract function.

# 4. Feature Summaries

In this chapter, a feature summary for all functional and non-functional features identified by this work is given. This is designed for DLT newcomers and junior developers that want to grasp the possibilities of features that could be implemented in their companies new DLT project or DLT platform. Even though functional and non-functional features go hand in hand, they are separated in this chapter. When functional features are important for implementing non-functional features, the differences of the implementations and their consequences are shown so that developers can make an educated choice regarding functional features looking at non-functional features.

## 4.1. Functional Features

In this section, all functional features found in DLT projects are categorized into twelve main categories. Each category contains its responding features with a description of it. Additionally, a brief description is given for every category, and a common use-case is described. The catalog was verified with industry experts named in the next chapter as part of their respective use-case studies.

### 4.1.1. Identity Management

Identity Management refers to a framework of policies and technologies for ensuring that only authorized individuals can access the associated resources in an organization [23].

Identity Management is the most important feature of any DLT system. When a participant of the network wants to take action in it, e.g., transfer Cryptocurrency, create an asset, etc., the participant needs to be verifiable to, for example, check if the participant has enough Cryptocurrency to send if he has the correct permissions to create an asset, etc. In a DLT system, members do not participate with their real identity, email address, or contact details. Cryptographic algorithms and keys identify participants. The most common ones are explained in the subsections.

#### Asymmetric Cryptography

In most Cryptocurrencies such as Bitcoin (BTC), Litecoin (LTC), Ethereum (ETH), asymmetric cryptography is used for identity management. The general functionality of a key pair is the same as for an RSA key pair. There is a public-key and a private-key, which is derived from the public-key. The private key is used to identify the identity

owner and can be compared to something (less complex) like a pin code. In Bitcoin, for example, the hashed public-key is used as the address participants can send bitcoin to, and the private key is used to sign transactions to confirm their validity.

#### Certificate based Authentication

Certificate-based Authentication encapsulates a digital identity for every actor in the network that can consume services with an X.509 digital certificate. The main difference between certificate-based authentication and asymmetric keys is that certificates allow defining exact permissions over resources and access to information that actors have in the network. Additionally, certificates may embed principals like groupID, giving the actors access to an organization's resources and their respective permissions. The certificates, however, must be signed by a trusted authority. Outside of the Blockchain sector, this would be the Root-Certificate-Authorities. The specific example of Hyperledger Fabric implements this trusted authority using a membership service provider (MSRP) which defines the rules that govern valid identities for organizations [17].

#### Self-Sovereign Identities (SSI)

Self-Sovereign Identities try to solve the issue that nowadays, there are an increasing amount of digital identities that are associated with the workplace, personal life, and professional activities. Self-Sovereign Identities give the owner of an identity full control over it, which strengthens data-privacy. A Blockchain-based system enables users to shift their sensitive data from servers of third parties to a distributed ledger. In a Blockchain-based System designed for distributed identity management, a user can act as a node in the network and store its own identity on the ledger. This can minimize the risk of, for example, user identity abuse [23]. Three prominent Blockchain-based identity management systems are Sovrin, uPort and ShoCard [23].

### 4.1.2. Unique Asset Management

Unique Asset Management describes the creation and storage of uniquely identifiable objects on the ledger. Examples of such assets are ERC721 Tokens (Non-Fungible Tokens) on the Ethereum Blockchain that are used to describe, for example, physical property such as houses, unique artwork, virtual collectibles such as unique pictures of kittens[1], collectible cards or represent "Negative value" assets such as loans or burdens [36]. Another example of Assets are objects on Hyperledger Fabric, which are represented with a key and a set of key-value pairs where arbitrary data can be created, and stored [18]. Special characteristics of this feature are Digital Twins and Ownership.

[1] https://www.cryptokitties.co/

#### Digital Twin

A *Digital Twin* describes an - at best - exact representation of a physical item in a digital form [35]. The goal of a Digital Twin on the Ledger is to create a secure and permanent record of an asset's characteristics. For example, this can be used to describe a unique painting on the Ledger, and when the ownership of the physical painting changes, the ownership of the digital painting changes too.

#### Matching Unique Assets with Digital Identities

Representing ownership of a unique asset can be done by connecting digital identities such as asymmetric key pairs, certificates, or SSI with a unique asset. To link the digital identity, a reference to it is written into a unique asset. Using asymmetric key pairs, either the public-key or the hashed public-key is written into the unique asset. Certificates allow for more fine control over ownership. For example, a unique asset can be assigned to a single actor or to an organization where every actor in the organization has control over it.

### 4.1.3. Unique Asset Transfer

The easiest way of transferring a unique asset is to change the owner by, for example, issuing a transaction or changing the properties of the unique asset through a smart contract/chain code interaction. However, an asset can also be exchanged against other assets or Cryptocurrency. Code Execution on the Blockchain makes trades with rules, but without intermediaries, possible.

### 4.1.4. Code Execution on the Blockchain

Code Execution on the Blockchain describes the execution of transactions without intermediary on the chain using smart contracts. Depending on the used software - in this example, Hyperledger -, smart contracts may also be called chain code [16]. The idea of smart contracts was formulated by Nick Szabo [34] way before the publication of the Bitcoin Whitepaper by Satoshi Nakamoto. In Szabo's original definition, he describes smart contracts "as a computerized transaction protocol that executes the terms of a contract". He also defines their general design objectives "to satisfy common contractual conditions [...], minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries."

In the case of smart contracts on the Blockchain, the goal is not only to minimize the need for trusted intermediaries but eliminate it. The following two subsections describe specific smart contract and chain code implementations on Hyperledger Fabric and Ethereum.

#### Execute-Order-Validate Architecture Pattern

In Hyperledger, the state of the Blockchain is maintained by a versioned key-value store. Each entry in this store is a tuple (key, version, value) [33]. The execute-order-validate architecture represents a way of modifying the Blockchain state and is divided into three execution steps: Execution, Ordering, and Validation.

**Execution**: a client proposes a transaction consisting of interactions with smart contracts to endorsing peers selected by an endorsement policy. The endorsing peers then all execute the transaction simulation in parallel and send the result, which contains the readset (all keys read by the simulation) and writeset (all keys modified with their new values), together with their endorsement signature back to the proposing client [33].

**Ordering**: here, transactions are ordered in a total order to create a block. There can be many different orderers from different organizations, but they have to follow the same consensus protocol. A block is created when the number of pending transactions reaches a certain threshold, or a timeout is triggered [33].

**Validation**: validation happens when a block has been received from the ordering peers. The receiving client then sequentially validates each transaction based on the corresponding endorsement policy and transaction serializability, which is tested by inspecting the staleness of its readset. In the event that the transaction does not match the endorsement policy or the transaction serializability, it is marked as invalid [33].

#### EVM Architecture Pattern

The full computational state of the Ethereum Virtual Machine (EVM) can be defined by the tuple (block_state, transaction, message, code, memory, stack, pc, gas). Where *block_state* is the global state of all accounts and includes balances and storage [6].

Code can be executed in the EVM using *Ethereum virtual machine code*. The code consists of a series of operations, with each operation having its own program counter. Code execution is an infinite loop that executes these operations at the order of the program counter until a STOP or RETURN instruction is reached. The smart contract code can only access the value, sender, and data of the incoming message and the block header data [6].

### 4.1.5. Information Broadcasting

Information Broadcasting over the Blockchain describes the usage of the underlying peer-to-peer network of connected nodes to send information. The information can, for example, be a raw transaction that is propagated to the network and then - if valid - incorporated into the Blockchain, or it can be an arbitrary message directed at a specific node in the network. Another possible piece of information could be the broadcast of a new node joining the network with the request to also open a connection to the joining node. Lastly, events such as contract modifications can be broadcasted and listened to by event listeners.

### Message broadcasting over the Network

Message broadcasting over the Network is used for building second-layer solutions such as messengers on top of DLT applications. A second layer solution uses existing infrastructure to create a new use-case on top of it. In the specific example of a messenger on top of DLT applications, a message that is encrypted with the receiver's public key is broadcasted over the network, which can only be read by the receiver. The downside of this approach is that everyone has to relay every message sent in the network. This approach's upside is the use of existing infrastructure, complete anonymity through peer-to-peer use, and the security of the existing public-private key encryption scheme. An example for an implementation is TrezarMessage[2].

### Event Emits and Event Listeners

Event listeners on the Blockchain work the same way as any other event listeners. An event listener is bound to a specific event and triggers once this event is emitted. Following, the specific implementation of Hyperledger Fabric 1.4 is presented for contract events:
With this listener, the block number, transaction id and status is printed every time the

```
/**
* @param {String} listenerName the name of the event listener
* @param {String} eventName the name of the event being listened to
* @param {Function} callback the callback function with signature
(error, event, blockNumber, transactionId, status)
* @param {module:fabric-network.Network~EventListenerOptions} options
**/
const listener = await contract.
addContractListener(
'my-contract-listener',
'sale',
(err, event, blockNumber, transactionId, status) => {
       if (err) {
              console.error(err);
              return;
       }
       console.log(`Block Number: ${blockNumber} Transaction ID:
       ${transactionId} Status: ${status}`);
})
```

Figure 4.1.: Registering a contract listener [21]

[2]https://github.com/TrezarCoin/TrezarCoin/blob/master/src/qt/trezarmessage.cpp

event is emitted.

### 4.1.6. Storing verifiable Data Off-Chain: Hash Anchoring Pattern

In a distributed ledger system, it is best practice to store as little information as on-chain since all participants of the system have to verify and synchronize all content that is stored on-chain to ensure full replication and verifiability. Additionally, Blockchains like Ethereum and Bitcoin limit the amount of data that can be put on the chain in one transaction.

XU [38] describes the forces of this problem:

- *Scalability*: limited, full replication and immutability.
- *Cost*: embedding data into transactions is cheapest, storing data in a contract is more efficient but less flexible. Transaction costs vary between Blockchains.
- *Size*: Block sizes are limited.

Rather than storing data directly on the chain, the distributed ledger system only stores an immutable reference in the form of a hash.

However, this approach is mostly only useful for private permissioned ledgers as a complementary database is needed to store the additional data. This database would also be distributed in the best case—an example for this is IPFS[3].

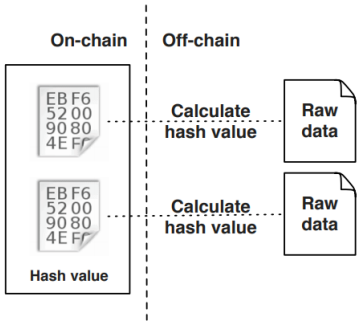



Figure 4.2.: Off-chain Data Storage Pattern [38]

This behavior is depicted in 4.2. In addition to the reference, metadata about the data can be stored on the distributed ledger system to, for example, make it easier to find or index it.

### 4.1.7. Storing Data On-Chain and its Usage

When the hash anchoring approach is not applicable, for example, in public permissionless Blockchains, it has to be stored on-chain. This category shows the most effective ways of doing so.

[3]https://ipfs.io/

### Encrypting Data on-chain Pattern

Encrypting data on the chain is necessary when the information written to the chain has to stay private, for example, the trade secrets of a company. XU [38] suggests using symmetric or on the data.

XU [38] describes the forces to balance this problem as follows:

- *Transparency*: every transaction on the blockchain is visible for every participant on the internet or even every user of the internet using a Blockexporer for common chains such as a Bitcoin explorer[4] [5].
- *Lack of confidentiality*: since all data is publicly visible, no sensitive data should be stored on the Blockchain in plain text.

Furthermore, he describes the drawbacks of this approach, with the main concern being that even though the data is encrypted, it will be on the ledger forever, and there may come a time where the encryption can be broken.

Another drawback of this approach is that the data will not be searchable.

### Oracles Pattern

The EVM as described in 4.1.4 is a closed execution environment that can only access data of the system. This is the case with all systems that do code execution on the Blockchain. This has the background that every transaction must always yield the same result at any point in time given the Blockchain's same state, meaning that there can never be a dependency on outside sources that could change their content rendering the Blockchain unverifiable.

Oracles solve this problem by introducing outside resources to the Blockchain through, for example, the invocation of a smart contract or adding the data in transactions.

Xu [38] describes the drawbacks of this approach being *trust* and *validity*. Participants of the network have to trust the oracle only to insert valid data into the system.

### Reverse Oracles Pattern

As opposed to oracles, Reverse oracles are used to integrate data of the Ledger into external systems. The reverse oracle should be designed to be minimally intrusive to the external system, for example, with a REST-API Endpoint that is universal. The base functionality of a reverse oracle is depicted in 4.3.

XU [38] suggests *Connectivity* and *Simplicity* as forces that need to be balanced. On the one hand, the ledger should be integrated into an existing system the best way possible, and on the other hand, it should introduce only minimal changes to the existing external system.

[4] https://www.blockchain.com/de/explorer
[5] https://btc.com/

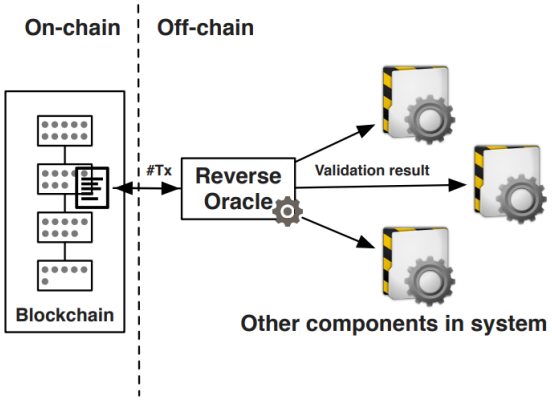



**Figure 4: Reverse Oracle Pattern**

Figure 4.3.: Reverse Oracle [38]

### Findable Data

On large and small distributed ledger systems, it can be necessary to find specific data on the ledger like assets. Let's look at a business example: Company A stores products on the blockchain with specific properties like category, name, and location. One workflow of the company is to get a list of all assets currently in warehouse XY.

To achieve functionality like this, we have to look at the different chains. In Blockchains that do not feature a world state like Hyperledger does[6], the whole blockchain needs to be iterated in order to find the matching items, which is very tedious. Here it is useful to build an index in each client that can query the blockchain where the current state is represented as a snapshot, which can then be queried. Each client should calculate the current state based on the Blockchain data to maintain trust.

In extensive Blockchain-based systems like the Ethereum Mainnet, an indexing approach might be infeasible. Here a registry service where any asset or contract that needs to be findable is registered with the identifiers needed for querying.

An example for such a registry is thegraph[7]. It is an indexing protocol for querying networks like Ethereum and IPFS with open APIs called subgraphs. Thegraph also aims to solve the issue of encrypted data (4.1.7) on the Blockchain not being findable but has solved it as of 16.12.2020.

## 4.1.8. Interoperability

Interoperability is concerned with the interaction between two or more ledgers. The most prominent example for a framework that enables interoperability between ledgers is Polkadot[8]. For example, interoperability can be used to tackle scalability issues, verify

[6]https://hyperledger-fabric.readthedocs.io/en/release-2.2/ledger/ledger.html
[7]https://thegraph.com/
[8]https://polkadot.network/

information across networks, or exchange information or assets between distributed ledgers.

#### Atomic Cross-Chain Swaps

"An *atomic cross-chain swap* is a distributed coordination task where multiple parties exchange assets across multiple blockchains, for example, trading bitcoin for ether" [15]

HERLIHY [15] lists the following guarantees for the atomic swap protocol:

1. The swap only takes place if all parties agree
2. No party ends up worse off when one coalition deviates from the protocol
3. There is no incentive for the party to deviate from the protocol

HERLIHY [15] also gives a practical example for the use of atomic cross-chain swaps: Three participants in a transaction, namely Carol, Alice, and Bob. Alice wants to buy an asset from Carol, but Carol only accepts Cryptocurrency that Alice does not have. However, Bob can exchange the Cryptocurrency that Alice has in the Cryptocurrency that Carol wants. The transaction, therefore, goes as follows: Alice transfers her Cryptocurrency to Bob, Bob transfers his Cryptocurrency to Carol, and Carol transfers the asset to Alice. When any party deviates from the intended behavior, refunds are issued, and no party is worse of than before. For the technical implementation of this case, HERLIHY [15] should be consulted, as it is too complex for this thesis.

#### Flash swaps, Liquidity pools

ADAMS [1] introduces *flash swaps* as a feature for traders that allows them to receive assets and use them somewhere else before paying for them later in the same atomic transaction. The traders pay a minimal fee for this.

For enabling *flash swaps*, a liquidity pool is required. A liquidity pool is a smart contract wallet in which participants lock their funds for a specific period of time to provide liquidity for the smart contract application. The participants get rewarded for locking their funds with interest either in the token they locked or in a token issued by the liquidity pool provider. The rewards are paid for by the trader fees.

The most prominent example for this is Uniswap v2[9].

### 4.1.9. Governance

All distributed ledger technologies are first and foremost *distributed*. Governance tries to enable participants of a network to not only have distributed consensus for verifying data in the distributed ledger system but also in the decision making of further developments that upgrade the network.

[9] https://uniswap.org/

This is not as important in *private permissioned* system as there most decisions are made outside of the system in a more traditional environment like conferences between the participating companies or parties.

However, in a public permissionless system where every participant wants to remain anonymous, on-chain governance tools are essential for further development.

### Superblocks

Superblocks are blocks in a Blockchain containing a large number of coins or tokens that are mined periodically. This Cryptocurrency is used for funding projects that help the Blockchain to improve its technology, increase awareness or help the community in general.

Such a system is built of three components: Proposals, Votes, and Superblocks.

Users who are skilled in their respective traits and want to improve the Blockchain or its ecosystem can open a proposal on the Blockchain where they describe what they want to develop or do and put this proposal on the Blockchain. PIVX[10] also offers a forum where proposals are pulled from the Blockchain and can be discussed.

Each participant of the network can then vote on the proposals. To be eligible for voting, criteria such as "node has to be active for at least one month", "node has to own at least X Cryptocurrency", or "Y Cryptocurrency has to be locked with timelocks" can be defined.

The voting is purely done on the Blockchain. PIVX implementation of the voting is as follows *mnbudgetvote many <hash> yes* to vote in favour, *mnbudgetvote many <hash> no* to vote against.

Depending on the *yes* and *no* votes, the next Superblock pays out to the proposal creators sorting by in favor descending until no more Cryptocurrency is in the Superblock.

The specific implementation of PIVX is PIVXCentral[11] where all current, archived, and coming proposals are listed. However, once the provider of PIVXCentral has no financial interest in running the page anymore, all additional information to the proposals are permanently deleted. Only the raw information that is actually stored on the Blockchain will last until the last participant of the network shuts down its node and deletes the Blockchain data.

### Signalling and Voting

Contradictory to Superblocks, signaling and voting is not about funding projects but rather accepting modifications to the Blockchain, which change the consensus mechanism and cause a hard fork or a soft fork. Miners then signal their preference.

In Bitcoin for example, each miner can signal support for a fork by setting a bit in the version field of the block that he mined. The specification of BIP9[12] defines that

[10] https://pivx.org/
[11] https://pivxcentral.org
[12] https://github.com/bitcoin/bips/blob/master/bip-0009.mediawiki

each soft fork needs to define a *name*, *bit*, *starttime* and *timeout*. Where the *bit* has to be chosen from the set {0,1,2,...,28}. Once enough miners signal for their preferred fork, it transitions to state *LOCKED_IN* and is activated on the next block that satisfies the condition *block.height % 2016 == 0*.

This, however, is a specification that should be used to keep the network stable. However, everyone with enough hashing power to create a new block with a different consensus mechanism can fork the Blockchain.

### 4.1.10. Cryptocurrency Minting

This subsection discusses how Cryptocurrency can be created. Five categories will be described which each have the goal to create or destroy Cryptocurrency. Two of them will introduce Cryptocurrency as a reward for validation and work done on the Blockchain, while three merely describe how Cryptocurrency can be created or destroyed.

#### Pre-Mined Cryptocurrency

Pre-Mined Cryptocurrency defines a way of obtaining Cryptocurrency before any consensus mechanism has been executed. On creating a Blockchain, the genesis block assigns the desired amount of Cryptocurrency to an address controlled by the Blockchain's creator. When acting in good faith, the development team behind the Blockchain will use the pre-mined coins to pay for development efforts.

#### Mining Pattern: Proof-Of-Work

Using SHA256, a hexadecimal hash from an arbitrary value can be created.

ANTONOPOULOS [3] gives the simple example of *"I am Satoshi Nakamoto"* hashing to
*5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e*
using the SHA256 algorithm.

The Proof-Of-Work algorithm rewards miners for finding a block with a block header of less value than a set amount. The minimal difficulty in Bitcoin starts with 1 and is currently at 18670168558399,59 (Block height 661782) with the block hash of that height being
00000000000000000090f2d2575fe8c7b97b266ec2b9721f3d340f1ad034e20

This difficulty can be translated to the bits *0x170f1372* and the target value of less than resulting in the target value of less than
00000000000000000170f13720000000000000000000000000000000000000 when compared to the hash of the block at that height, it can be seen that it is less than the target value. The conversion was done using the formula $nbits * 256^{(0x18-3)}$ so for our specific block height $0x170f1372 * 256^{(0x18-3)}$.

The reward that miners received is called the mining reward. Finding a correct block allows miners to issue a so-called *coinbase* transaction to themselves with the of the

network agreed amount in it. The correctness of the block is determined by the correct block structure as defined in the protocol.

The Bitcoin network aims to create one block every ten minutes for the network. To achieve this goal, the difficulty for the Proof-Of-Work algorithm is adjusted every 2016th block. The difficulty directly corresponds to the hash-value that the block header has to be smaller than.

The main goal of Proof-Of-Work, however, is that the network is kept stable and that there is always a longest chain[13] and not many chains with the same height that result in orphaned blocks.

The Proof-Of-Work algorithm was introduced by Satoshi Nakamoto in the 2008 Bitcoin white paper [24].

#### Staking Pattern: Proof-Of-Stake

Proof-Of-Stake offers an alternative to the Proof-Of-Work algorithm, with its main advantage being that it does not consume an excessive amount of energy.

In the Proof-Of-Stake algorithm, the user's coins and the time they have not been moved define how much staking power a user has. The calculation proposed by [22] which took the concept of *coindays* from NAKAMOTO [24] is *coins * days*. When Alice has 10 coins, and they have not been moved for 20 days, Alice has a staking power of 200 coindays. When Alice now moves the coins to Bob, the coindays are at 0 again. KING [22] introduces the Proof-Of-Stake as follows:

> The proof-of-stake in the new type of blocks is a special transaction called coinstake (named after Bitcoin's special transaction coinbase). In the coinstake transaction block owner pays himself thereby consuming his coin age while gaining the privilege of generating a block for the network and minting for proof-of-stake. The first input of coinstake is called kernel and is required to meet certain hash target protocol, thus making the generation of proof-of-stake blocks a stochastic process similar to Proof-of-Work blocks. However an important difference is that the hashing operation is done over a limited search space (more specifically one hash per unspent wallet-output per second) instead of an unlimited search space as in proof-of-work, thus no significant consumption of energy is involved. [22]

#### Minting

Minting is mostly applied in Stablecoins such as Tether[14] which are controlled by a central authority. Stablecoins have the property that they are always worth the same. For example, one USD-Tether (USDT) is always worth one US-Dollar (USD). The pattern has two functions: First, the minting. Here the central authority can create an arbitrary

[13] longest chain refers to the chain with the most work done
[14] https://tether.to/

amount of Cryptocurrency to an arbitrary address. Second, the removal of coins from the system.

In general, this pattern makes the most sense in a permissioned environment or in a system where an authority has the power to force the system upon its users.

#### Elastic Supply

The idea of elastic supply is to connect the supply of a Cryptocurrency to a fiat currency. The connection triggers when a certain threshold of price change between the Cryptocurrency and fiat currency pair happens and balances the supply accordingly.

Ampleforth[15] is the first Blockchain to implement this concept. Ampleforth has three states: Expansion, Contraction, and Equilibrium. When the price of one AMPL (Cryptocurrency unit of Ampleforth), for example, raises by 10%, then the expansion state is triggered where the amount of AMPL every participant owns increases by 10% so instead of the price rising, the amount of AMPL in the participants wallets rises. Contraction is triggered respectively when the price lowers. Equilibrium is the default state when the price is stable.

### 4.1.11. Cryptocurrency Storing

This category is about storing Cryptocurrency. It shows different ways to programmatically create ways to store the data needed to check the balance of a participant in a Blockchain-based system.

#### UTXO Pattern

UTXO refers to *unspent transaction output*. UTXOs are indivisible chunks of Cryptocurrency that can be combined to create new transactions. Each UTXO can only be spent fully [3]. The combination of all UTXOs in the wallet of a user is his balance. For better understanding, an example is given in the following paragraph.

Alice has 50 bitcoin in a single UTXO and wants to send 20 bitcoin to Carol and 15 bitcoin to Bob. Alice now creates a transaction with a single input - her UTXO - and three outputs for Carol, Bob, and her return. Alice now has a UTXO worth 15 bitcoin, Bob one that allows him to spend 15 bitcoin, and Carol 20. Alice now wants to send another 20 bitcoin to Carol. Alice has to combine her 15 bitcoin UTXO with another input to create another two outputs. If Alice would only create one output for Carol and none for herself, the miners would receive the remaining bitcoin. The thought experiment is depicted in 4.4.

Summarising, the following statements about UTXOs can be made:

- UTXOs can only be spent fully
- One UTXO can create many other UTXOs in a transaction

[15] https://www.ampleforth.org/

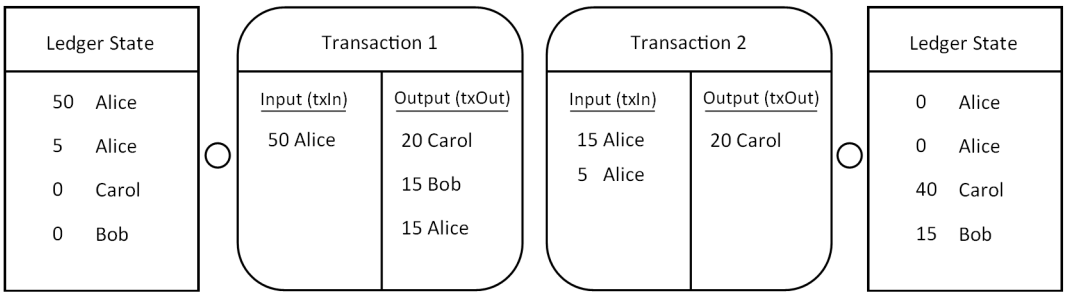



Figure 4.4.: Example UTXO transfers

- Multiple UTXOs can be combined to create a single output
- Everything above the combined value of the input UTXOs minus the combined outputs is given to the miner

Besides, the UTXO Pattern offers a much higher degree of privacy than the accounting system since linking transactions to users is much easier than linking transactions to UTXOs linked to a user.

### Account Pattern

The Account Pattern takes a simpler approach to storing Cryptocurrency when compared to the UTXO Pattern. Here, every user has an account, hence, the name for the pattern. With every transaction, balance can be added or subtracted from the account given the transaction's nature.

Ethereum, for example, takes this approach to accounting. Every user has an account that is created by generating a key-pair.

### Hybrid Approaches

Hybrid approaches are used when the simple functionality of an Account Pattern is needed for usability, but the underlying system is a UTXO-based Blockchain.

Beforehand, it is important to note that the Bitcoin-Core software deprecated its accounting system, which implemented this hybrid approach in version 0.16.0[16] replacing it with a less automatic system.

This chapter briefly describes the hybrid approach and then goes into details about why this approach did not yield the results the Bitcoin-Core developers expected it to.

The hybrid approach builds a virtual layer on top of the Bitcoin-Wallet, summing up all UTXOs to a single balance for every account to display it in the wallet. The underlying system to create transactions remains the same. The only real difference lies in the handling of the specific accounts inside a wallet. The Bitcoin-Wallet allows users to create an arbitrary amount of named accounts inside a single wallet, enabling,

[16]https://bitcoincore.org/en/doc/0.16.0/rpc/wallet/getbalance/

for example, exchanges or other service providers the possibility to directly send and receive from a single wallet using the account as a parameter.

However, especially the exchange wallets accumulate hundreds of thousands or even millions of UTXOs. Since the layer on top of the Bitcoin-Wallet is only virtual and not persistent, all UTXOs have to be iterated to fetch the balance of a single account or to create a transaction resulting in extremely heavy executions of a single balance call.

This resulted in the change of the Bitcoin-Core RPC API to not allow this behavior anymore and developers needing to create an external database when they want to manage multiple accounts in a single wallet efficiently.

### 4.1.12. Cryptocurrency Transfer

This chapter goes into detail about how Cryptocurrency can be moved from one participant of the network to another participant of the *same* network and which security measures should be implemented by developers when they design systems accepting Cryptocurrency.

#### Confirmation Pattern

The Confirmation Pattern is a security pattern that aims to improve the security of systems accepting Cryptocurrency. Even though the Blockchain is immutable in theory, there exists the possibility that a few blocks are replaced by a competing chain fork [38].

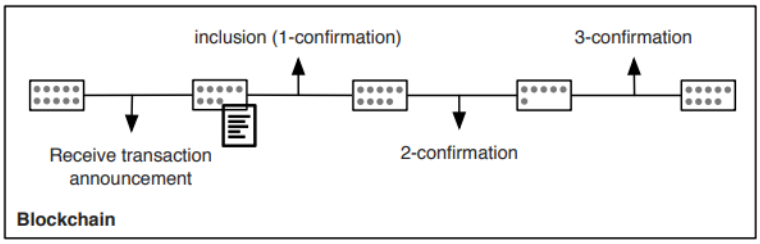



Figure 4.5.: Confirmation Pattern [38]

In figure 4.5 confirmations are depicted. After the transaction has been broadcast to the network, it has 0-confirmations. Once it is included in a block, it has 1-confirmation. Each following block adds another confirmation.

XU [38] names the two forces for this pattern. *Chain fork* and *Frequency of chain fork*. Where *Chain fork* is the possibility of a chain fork and *Frequency of chain fork* the frequency of forks. He also lists the major drawback as being *Latency*. When for example, a Bitcoin transaction is issued and is included in the very next block, a single confirmation takes 10 minutes. When the application now has to wait for 6 confirmations, this security pattern adds a latency of 60 minutes to the application.

Here, the forces and drawbacks should be balanced. When implementing this pattern, it is important to know how frequent chain forks happen and to what extent they

happen. In Bitcoin, for example, the chain is very stable due to the high difficulty needed to find new blocks using the Proof-Of-Work algorithm, and therefore a lower number of confirmations for Bitcoin might be acceptable such as 1 to 3 confirmations. In smaller networks, e.g., Bitcoin clones, 6 to 12 confirmations might be needed as the work was done that is needed to publish a new longest chain could be afforded by a single malicious actor.

#### Single Signature Send

The Single Signature Send represents the default way of sending Cryptocurrency. It defines that users create a transaction that is syntactically and semantically in the bounds of the consensus protocol and sign this transaction with their private key, using funds that are only locked by a single signature requirement.

#### Multiple Signature Pattern

The Multiple Signature Pattern describes a way of sending Cryptocurrency which requires at least M out of N signatures to create a valid transaction to spend funds and is also known as the M-of-N scheme [3]. For example, two out of three possible signatures are needed to spend funds.

An example of the need for this pattern is exchange wallets where millions or even billions of USD worth of Cryptocurrencies are stored. Here, the signatures of multiple high ranking employees of the exchange are required to send large amounts of Cryptocurrency so that in the case that a single or multiple employees have their private keys stolen, the funds are still secure.

A locking script in Bitcoin for setting up such as M-of-N multi-signature condition is: *M <Public Key 1> <Public Key 2> ... <Public Key N> N OP_CHECKMULTISIG* [3], where *M* is the corresponding amount of signatures needed to use the funds and *N* the total amount of signatures in the lock.

## 4.2. Non-Functional Features

Non-Functional Features are as important as functional features, if not more important. Functional features in DLT make systems compelling to solve DLT-based Business Problem Solverss. However, non-functional features make the system useable in production through scalability, privacy, and security.

Therefore, this chapter summarises the forces that need to be balanced when designing a distributed ledger system and describes the ups and downsides.

### 4.2.1. Scalability

Scalability in DLT is a double-edged sword with two main components: Keeping the size of the Blockchain low, decentralization high, stability high, and executing transactions

fast. The goal of every distributed ledger system is to prioritize the right attributes for the right use-case. Therefore, a general recommendation can not be given, and every developer that creates a new distributed ledger system has to make his own decisions. The following points give examples of which adjusting screws can be used and describes the benefits and drawbacks of each.

#### Stability, Decentralisation, and Size VS Transaction Speeds

The simplest way to increase the speed of a Blockchain-based system is to increase the Blocktime. For example, Bitcoin is hard to justify for daily use because of the 10-minute Blocktime combined with the relatively small Blocksize. Those two factors increase the cost for each transaction. One could now reduce the Blocktime to 10 seconds and increase the Blocksize by tenfold. This would result in Bitcoin being perfectly usable for daily use and make it, in theory, the perfect payment solution. One could even decrease the Blocktime even further down to one second to make the transactions even faster.

However, the downsides to this are predominantly. First, Bitcoin would see huge inflation since the Block reward has not been lowered.

Second, the chain would produce lots of orphan blocks since the work needed to create a block is significantly reduced, resulting in lots of miners finding the next block simultaneously again, resulting in a volatile network.

Third, the Bitcoin Blockchain size would increase heavily since everyone can now spam transactions to the chain and therefore bloat it. This results in a Blockchain size that is no longer usable for the average user running a full-node. Consequently, only a few large participants would still have their nodes running, resulting in a more centralized network.

Summarizing, faster transaction speeds result in a more unstable, centralized network with a more data-heavy ledger.

#### Sidechains

"A Network of Micropayment Channels Can Solve Scalability" [26] - Joseph Poon, creator of the Bitcoin Lightning Network.

The Bitcoin Lightning Network is, as of today, the best example for a sidechain that solves the scalability issues of a large distributed ledger.

The idea of the network is that a channel is created between two parties with a funding transaction that locks the parties' funds on the real Bitcoin Blockchain. After this, the parties can make transactions to each other while the channel tracks the parties' balance. Once one party needs the funds on the real Bitcoin Blockchain, the channel is closed, and one multi-send transaction is issued to the network to settle the balance of the channels on-chain to the parties. For a deeper understanding, POON [26] should be consulted.

The only downside to this approach is currently the immaturity of the software, which does not always behave correctly from an administrator's perspective. However, the

cryptographic functions have not been found to be incorrect yet, so that the lightning network - which is still in beta - is in use in production for many large players in the Blockchain space.

Popular implementations of the Lightning Network Protocol are c-lightning[17] and LND[18]. A more technical overview of the protocol and its implementations can be found on the *awesome-lightning-network* repository on GitHub[19].

### 4.2.2. Privacy

Generally speaking, one can divide privacy in DLT and Blockchain into the different categories of user privacy and data privacy. Furthermore, those two have to be looked at from the permissioned and permissionless side. Permissionless systems tend to value user privacy higher than permissioned systems—Permissionless systems, on the other hand, value data privacy less.

This observation is made from looking at the different features of common permissionless and permissioned systems. For example, Bitcoin implements methods that make it harder to track a user's identity but none out of the box for user privacy. Hyperledger Fabric implements private data collections and channels to make the ledger's data only available to the ones that should have access to it. However, this assumption is not always correct as BUTERIN [7] presented privacy-preserving smart contracts that encrypt data inside a smart contract for privacy back in 2016 on Ethereum, which is a permissionless platform.

We conclude that user privacy on permissionless system is only more important on systems whose main feature is Transferring and Storing Cryptocurrency. Whereas in permissionless systems that implement Code Execution on the Blockchain, Asset Management, Asset Transfer and Data Privacy is as important as, or more important as User Privacy.

Different functionalities in DLT and Blockchain can be implemented to make user and data privacy better. The ones we describe in this work are pseudo-anonymity, which describes participants interacting on the network using pseudonyms, private ledgers as described in subsection 2.1.2, which restrict access to a few selected participants, encryption which was already explained in subsection 4.1.7 and Zero-Knowledge Proofs which allow confirmation of a transaction without knowing its content. However, each feature that adds privacy decreases either security, scalability or usability.

#### Pseudo-anonymity

Pseudo-anonymity is only relevant in permissionless system since all participants in permissioned systems are known. In the section Identity management 4.1.1 possible ways of identification were described. These ways always provide a cryptographic

[17] https://github.com/ElementsProject/lightning
[18] https://github.com/lightningnetwork/lnd
[19] https://github.com/bcongdon/awesome-lightning-network

pseudonym for a real participant whose physical address and details are unknown. However, if used carelessly, the traces of the cryptographic pseudonym can be tracked back through looking up transactions, and assumptions can be made about the real identity of a participant when he makes a transaction to a service where Know your Customer (KYC) is required.

To combat the linking of a cryptographic identity with a real identity, we recommend strictly disallow address re-use, meaning that each address the user has can only be used for receiving and sending once. However, this is only possible in UTXO-based Ledgers as described in 4.1.11. Account-based Ledgers have to solve this issue differently by, for example, implementing a mixer by default. A mixer mixes coins of many different users by consolidating the balances on a single address and splitting it again multiple times, and then sending it to the users again in many transactions to different addresses.

#### Zero-Knowledge Proofs

Cryptographic proofs with strong privacy and efficiency properties are known as zkSNARKs (zero-knowledge Succinct Non-interactive ARgument of Knowledge) and are, for example, the core of the privacy-preserving Cryptocurrency Zcash [37]. zkSNARKS allow participants of the network to broadcast encrypted transaction details that can be proved without validators knowing the transaction's plain text contents.

> More formally, zkSNARKs allow a prover (e.g., a Zcash user making a payment) to convince a verifier (e.g., any other Zcash user) of a statement of the form "given a function F and input x, there is a secret w such that F(x,w) = true". In the cryptocurrency example, w is the private payment details, x is the encryption of the payment details, and F is a predicate that checks that x is an encryption of w and w is a valid payment. [37]

This allows participants to interact in the Ledger privately. However, this comes at the cost of a significant computational overhead on the prover side.

### 4.2.3. Security

Security in DLT has many aspects, such as guaranteed immutability, sufficiently distributed trust, attack resistance, and the correctness of implementing a protocol.

Since immutability is the core feature of a distributed ledger system, it must always be guaranteed.

Sufficiently distributed trust requires many different and, in the best case, equally computationally mighty nodes with higher block times, increasing the system's security and stability.

Attack resistance requires implementing the specific ledger to prevent all known attacks, or at least, strong mitigations against them. Examples of known attacks are 51%, Dos, Quantum, Double spending, Sybil, Eclipse, and Selfish mining. Since each attack is a topic of itself, this work will not further discuss the single attacks.

The implementation's correctness describes that the underlying code can not be exploited for personal or financial gain and follows the underlying mathematical guidelines properly. The most prominent example of an incorrect implementation is the smart contract of "The DAO" that had a flaw that allowed malicious parties to drain the ether stored on the contract to gain more than 3.6 million ether[20]. This attack shows that open-source is not a guarantee for the correctness of an implementation.

[20]https://www.coindesk.com/understanding-dao-hack-journalists

# 5. Case Studies

In this work, we present case studies that were created by interviewing industry experts. The interviews were scheduled to take about 30 minutes and were conducted over video chat. The questions used for the interview are split into five main categories: Meta Information, Overview of the Project, Technology Choices, Questions to the Catalogue, and Development Process. Each of the question categories is designed to gather a specific set of information. The full questionnaire can be found in A.4 (appendix).

The first part Meta Information collects personal information about the interviewee and his position, the company he works for, and his DLT background. The second category is designed to get an overview of the specific DLT project the interviewee is working on by asking for the problem the project is solving, the justification for DLT, and the perceived value for the customer.

The third category asks for specific technology choices by asking for the selected platform and the reasoning for choosing it and frameworks and design patterns used in the project.

The fourth category is only in part important for the case studies as it asks questions about the feature summary catalog. Here, the goal was to determine if the catalog is complete from their perspective, if any feature is misplaced, which features of the catalog are used in their project and if such a catalog would benefit their development process and a general understanding of DLT on a technical level.

The last category is about the development process and gathers information about the general development process, tools used to develop the project, and the deployment process in place. In the following sections, eight structured use-case studies will be presented, each using the same template to allow for a better comparison and summarization in the next chapter.

## 5.1. IBO - Product Tracing

### Meta Information

| | |
|---|---|
| Interviewee | Steven Pannell |
| Position | Chief Technology Officer |
| Company | IBO GmbH |
| Company Size | Medium |
| Company founded | 1991 |

## The Project

### Summary of the Project in one Sentence

Recording Quality-Documentation and making it available

### Project State

Minimal viable product (Development)

### Problem Statement

IBO is a manufacturer that creates parts for the aerospace industry and has a very complex supply chain. In the supply chain, a quality document is issued for every step in an item's lifecycle. Every material or component needs a quality document for every production and shipping step. The quality documents are important because materials and components have to meet ISO standards and processes are regularly audited by authorities, and customers want to verify materials and components.

Currently, these quality documents are only available in paper and are shipped with the materials and components. However, the documents get lost a lot of the time, and if the customer needs them, IBO has to find them and send them to the customer again. This is a very time-consuming process as there are many quality documents. Furthermore, the quality document is only included for the current step in the process and has no history.

Since there are many parties involved in the supply chain of IBO and all of them have different interests, it would not be possible for IBO to have a single database for all quality documents since then IBO would have a business advantage over the suppliers and their suppliers in the form of a knowledge advantage. That is why only an open system can be used in this case. Additionally, the gathered data must be in a data storage that can not be manipulated.

This project aims to create an easily adoptable cross-organizational documentation platform for small- and medium-sized manufacturers and supplies to track the complete supply chain, verify that correct quality documents exist and trace the origins and components of a product.

### How the Problem is solved

The main goal for IBO is to onboard every supplier in their supply chain on the new quality document tracing platform. The team tries to achieve this by giving the suppliers free access to the platform and tools to adopt it. The plan is that adopting the system is seen as a free technology push for small to medium-sized suppliers.

Their system is split into two main parts. The first part is responsible for storing the encrypted files in a file-storage, and the second part is responsible for saving the

decryption key to the quality document on the ledger and keep meta-information about the quality document on the chain.

The quality documents that are currently only available as paper need to be scanned and converted to an export format such as PDF. This document is then uploaded into the system through a frontend-application that passes that file to a backend-application, which then encrypts the document, creates an object on the ledger, and saves the access keys with the correct decryption permissions on the ledger.

The system only saves the hash of the encrypted file on the ledger. With this approach, the team prevents bloating of the blockchain and ensures that only the minimum necessary data is saved on the immutable ledger. This approach also allows the system to be GDPR compliant since documents can be deleted from the file storage when such a request is made. The system, therefore, uses the DLT Design Pattern **Storing verifiable Data Off-Chain: Hash Anchoring Pattern**.

Internally, the decryption keys and quality document hashes are stored in each organization's private data collection (PDC). However, the identity management is not yet implemented with the membership service provider.

#### Availability of the Project

The project is closed source. Making it open-source in the future might be possible, but it is not being discussed right now. The code is not very well documented in its current state, and it would take extra effort to make it useable for outsiders.

#### Justification for DLT

According to Mr. Pannell, DLT is needed because of the following reasons: The documents have to be stored on immutable file storage, and comparable systems that can store data immutable are niche and therefore very expensive.

None of the participants can gain more advantage from the system than another. A few years ago, a centralized system could have been possible, but after the recent debate about data-privacy and data-security, not many suppliers would accept a centralized system.

The before mentioned argument about the knowledge benefits also includes that everyone needs to have access to the same data, and therefore, a distributed ledger is required for this use-case.

#### What the Customer says

More recent feedback that the team received was not as expected. The team had the idea to use DLT for this use-case because of the aforementioned reasons, but the value of the technology choice is not that high for the customer. For the customer, the chosen technology is not that important. It is only important to them where they can save money

and where they can save time. Monetary the system is not justifiable, and because of this, the development of the system has slowed down by a lot.

## Technology Choices

### The Platform

The project uses Hyperledger Fabric (and AWS File storage)

### Used Frameworks and Design Pattern

According to Mr. Pannell, the team has built the application – more or less – from scratch.

### Public or Private

The team has looked into the matter of public versus private chain but decided on a private chain because they needed a permissioned base because the system cannot be completely open. The access to the data needs to be tightly controlled on an organization-granular level, and according to the team, this would not be possible with a public chain.

### Justification for the Platform Choice

The team did not have a long decision process regarding the platform of choice. Because of the immaturity of the technology, the team wanted to have a mature and actively developed platform that they can rely on. However, Mr. Pannell noted, "As it stands today, Hyperledger is not exactly the perfect fit, but it is the best that they have got at the moment," but it meets a certain feature saturation that allows for developing such a complex system. Another important factor for the decision was the support that Hyperledger Fabric receives, which is - according to the team - unseen in most other DLT-Platforms. Its modularity also allows each network participant to tailor their peers to their specific needs if necessary.

### Used Frameworks and Design Pattern

The Project of IBO can be represented using the Feature catalog:
**Storing data off-chain and its usage**: Strong verifiable data off-chain
**Unique Asset Transfer**
**Unique Asset Management**
**Code Execution on the Blockchain**: Execute-order-validate Architecture Pattern

### Development Process

#### General Approach

The team uses an agile approach for development with Kanban. The team runs on weekly sprints and does a delivery every week. The team also has a planning session where it defines the goals for the next week and sums up features that need to be implemented.

#### Used Tools

IBO uses the following tools for their development process:

- Jira for project management
- GitHub for storing the Sourcecode
- Visual Studio Code for development
- Azure for Continuous Integration and Continuous Development

#### Deployment Process

The Team deploys weekly where the code is manually deployed. There is no immutable data on the blockchain nor chaincode that would need to be endorsed.

## 5.2. Cash on Ledger - Payment processing

### Meta Information

| | |
|---|---|
| Interviewee | Maximilian Forster |
| Position | Co-Founder |
| Company | CashOnLedger |
| Company Size | Small |
| DLT Experience | Created Blockchain Offerings for KPMG, Accenture. |
| Company founded | January 2020 |

### The Project

#### Summary of the Project in one Sentence

Orchestration of payments using Blockchain-Technology

#### Project State

Live Case (Production)

### Problem Statement

CashOnLedger wants to establish a pay-per-use system for industrial machinery. CashOnLedger brings together the payment processing of the parties involved in the system so that customers only receive one invoice, and the individual payments are returned to the parties involved, such as manufacturers and insurers. However, this business model is only profitable if all steps - including a settlement with the bank - are fully automated.

### How the Problem is solved

For this purpose, CashOnLedger works together with *Lindner Traktoren Werke* from Austria. They build tractors that are equipped with telemetry data. This data is extracted from the vehicle and forwarded to CashOnLedger's platform. CashOnLedger brings this data into a settlement logic to present the settlements in a bill-compliant manner. For this purpose, CashOnLedger offers integration with ERP systems. Furthermore, a PDF file is created for the invoice and sent to the customer. The accounting logic is also connected to an interface in the bank to balance the amounts there automatically.

The core of the project is to bring data into accounting logic. If, for example, someone decides to use a tractor or any other machine, the machine's data is transferred to CashOnLedger and evaluated so that the data can be settled usage-oriented. The following is an example of how this concept is already being applied in the B2C sector: "Drive Now" offers its customers the option of renting cars by the minute so that only the actual use of the customer is paid for. CashOnLedger wants to implement this principle for all types of industrial machines and additionally allow user-defined orders through the collected data. For example, in the case of tractors, there are attachments such as spreaders and plows, which offer the customer additional benefits. However, there is also more wear and use of the machine, so that a higher price is charged if a tractor with a plow is used. CashOnLedger's business model is the billing of flexible and user-defined industrial machinery.

CashOnLedger represents the Pay-per-Use process fully automatically. All steps in the process have been fully automated, so no human intervention is necessary. The automation makes it possible to transfer and use the model for car-sharing to industrial goods. For the settlement of payments, the euro is, in fact, shifted and not a Cryptocurrency. For this purpose, ERC721 (Non-Fungible Tokens) are used because of the regulations, representing clear payment demands. A Blockchain listener has been implemented but is currently not used productively because customers are not yet up to date with the technology to be able to use it. Non-Fungible tokens are unique, unlike Bitcoin or ERC20 tokens, and can not be replicated or equally exchanged.

### Availability of the Project

The Project is currently closed source, and there are no plans of making it publicly available with open source.

#### Justification for DLT

The problem can easily be solved without DLT. However, DLT allows for a higher degree of automation. In the system, many parties are under one roof. There is the customer who uses the tractor and the manufacturer of the tractor. There are also insurance companies that cover the system's use in case of damage to the industrial machinery or unexpected breakdowns. A multi-party system is needed, which consolidates the payments because the end customer does not want to pay money to the manufacturer and the insurance companies but only wants to have one invoice, including all services. CashOnLedger then has to distribute this payment back to the different parties. Blockchain continues to offer the Single Source of Truth (SSoT) for the different parties to recalculate everything themselves. Finally, a bank is involved, with which the payments can be settled programmatically. The programmed payment eliminates an accountant who would otherwise have to execute the payment manually.

In summary, process efficiencies are achieved with DLT that were previously unattained. Rolls Royce, for example, has tried a similar system but failed because of the administration. The turnover for such a system has a linearly increasing turnover but exponentially increasing costs. The whole system can, therefore, only be profitable if all activities are automated. This automation reduces the costs to a linear instead of exponential growth.

#### What the Customer says

Customers do not care how the system is implemented internally. They only care about the degree of automation that can be achieved with the technology.

### Technology Choices

#### The Platform

Ethereum based on Hyperledger BESU is used. Hyperledger BESU is an enterprise Ethereum client that allows running public and private networks with either Proof-Of-Work (PoW) or Proof-Of-Authority (PoA) and is specifically designed for consortium environments. Furthermore, it supports test networks such as Rinkeby, Ropsten, and Görli.

The Ethereum client is connected with an interlink to Corda so that both platforms can be used simultaneously. Corda is a permissioned DLT Platform that supports Chaincode.

#### Used Frameworks and Design Pattern

CashOnLedger uses the following frameworks and design patterns:

- IoT Reference Architecture
- Swagger for the REST interfaces

- Microservice Architecture
- Standards developed with banking partners, such as network adapters that connect customers to network structures.

#### Public or Private

The CashOnLedger team decided to use a private blockchain because it has low transaction costs. Moreover, data protection was a crucial point.

#### Justification for the Platform Choice

In general, CashOnLedger requires two main points of the DLT architecture: Smart Contracts and Asset tokenization.

In the system, a virtual container is needed, representing money in the form of a token. Moreover, an automated logic is needed to use the money programmatically.

The team has chosen Ethereum on a private instance, realized by Hyperledger BESU because Ethereum supports token standards like ERC20 and ERC721 by default. Additionally, Mr. Forster claims that Ethereum has the largest developer community with 400-600 developers. This strengthens the trust in the chain and increases reliability.

On the other hand, the team has chosen Corda because the Corda Token SDK allows easy asset tokenization. Another reason for the decision was that it is one of the most widely used systems in the banking sector. In Corda, trust and reliability are also given by continuous development. This is especially important for CashOnLedger to comply with regulations and SLA's.

The team decided against Hyperledger Fabric because, at the start of the system's development, there was no easy way to tokenize to Hyperledger Fabric. Meanwhile, Hyperledger Token exists for this purpose.

#### What would happen if a new technology emerges?

Rapid technology changes in the DLT world do not pose a high risk for CashOnLedger since adapters can be exchanged quickly to adapt to a new DLT Platform.

#### Used Frameworks and Design Pattern

The Project of CashOnLedger can be represented using the Feature catalog:
**Information Broadcasting**: Event Emits/Event Listeners
**Unique Asset Transfer**
**Unique Asset Management**
**Code Execution on the Blockchain**: EVM Architecture Pattern
**Storing verifiable Data Off-Chain: Hash Anchoring Pattern**
**Interoperability**: Atomic Cross-Chain Swaps

### Development Process

#### General Approach

CashOnLedger uses an agile development process but does not use a Scrum master. Kanban boards are mainly used for the tasks.

#### Used Tools

CashOnLedger uses the following Tools for its Development Process:

- GitHub and GitLab for Sourcecode administration and task distribution
- Confluence with links to GitHub and GitLab
- Kanban boards on GitLab

#### Deployment Process

There is no special Deployment process in place. Since immutability does not a big role in the current system and most of the system is off-chain, deploys are just made to the network.

## 5.3. CodeNotary - Create Trust in Digital Objects

### Meta Information

| | |
|---|---|
| Interviewee | Dennis Zimmer |
| Position | Chief Technology Officer |
| Company | CodeNotary Inc. |
| Company Size | Medium |
| DLT Experience | Started in 2016 focusing on smart contracts |
| Company founded date | 2018 |

### The Project

#### Summary of the Project in one Sentence

Create trust in digital objects

#### Project State

Shipped (Production)

### Problem Statement

Software producers nowadays have their extensive Continuous Integration (CI) and Continuous Delivery (CD) pipelines. An example of such a pipeline would be the following: Every build of a project starts with the Sourcecode. In the Sourcecode, there is the configuration file for the CI and CD systems. After that, the quality assurance signs off on the Sourcecode and the CI and CD configuration. The project's dependencies get pulled from the internet or an artifact repository, and they are checked by anti-virus software for malware or known security vulnerabilities. When the initial build process has concluded, a binary file is produced, and the completion is verified by CI and CD Software like Travis, Bamboo, or CircleCi. Now that a finished build exists, the project is deployed encapsulated as a container, and this container is then put into a runtime like Docker or Kubernetes.

Programs running in the runtime of the software companies must be verified. The default way of doing this is by creating a digital certificate that assures the program's integrity. These certificates have the downside that they cannot be easily revoked, can only be issued once for every object, cost a non-trivial amount of money, can be issued in a malicious way that the user cannot distinguish between a real and a forged certificate, and additionally, not the workflow is certified by the certificate but only the result is.

A possible attack scenario could be the following: An attacker manages to inject malicious code into a repository or an artifactory because of a security issue within the Sourcecode repository or the artifactory provider. The injection stays unnoticed and is active in the next deployment of the product. A real-world example of this is the SolarWinds-Hack.

There may also be a situation where some parts of the build system have not been working properly, e.g., there are five virus-scanners in production, and one of them is outdated, and therefore the executables cannot be fully trusted anymore. Naturally, the software company now wants to set the executables to untrusted but cannot do so, as either all certificates of an issuer must be revoked or none. There is a big incentive for software manufacturers to not revoke all their certificates because of a few possibly malicious builds, so they are more likely to ignore those builds.

### How the Problem is solved

Initially, the project ran on Parity Ethereum but later moved to an own custom platform. During this process of moving the platform, the main business case changed from replacing digital certificates to certifying CI and CD pipelines of software manufacturers.

The first system's goal was to store digital assets like documents, docker containers, or executables, which are created in a software build process. All checksums of the build artifacts mentioned before are written to the Blockchain. These artifacts can now be trusted, and others can verify that.

Every participant of the system has a Wallet with which the participant can sign a step in the deployment process. The participant can call a smart contract with the object

which results from the step in the deployment process and can mark it as "I trust it", "I do not trust it", or "It is outdated".

All single items in the pipeline are converted to a checksum and notarized on the Ledger so that for every step in the CI and CD process, the integrity of the data and process is notarized by digital signatures that are stored in a smart contract.

At the runtime stage, an agent periodically checks if any program is running, which cannot be traced back completely to its origin by looking up the state of every executable on the Ledger to see whether it is trusted untrusted or outdated.

The original plan of CodeNotary was to replace digital certificates with their system, which was impossible since they would have to persuade Microsoft and Apple also to accept their system instead of only digital certificates. Therefore, they went ahead and marketed their solution to mainly big customers that needed to secure their internal build process.

Their second system's goal is to store as many checksums of artifacts on the Ledger as possible while maintaining immutability and being auditable.

It builds upon immudb, which turns around the concept of DLT. In immudb, there is only a single central database for the data storage, and only the clients perform the cryptographic proof. This concept allows CodeNotary to achieve ten million transactions per second. A transaction is a single write of a larger hash into the database; this is necessary because customers produce many artifacts that need to be verifiable.

Immudb carries along a Merkle tree, which is statically connected to the data, so the construct cannot be unraveled.

> The Merkle tree acts as a representative role in the blockchain and contains all transactions in a block. Such a container leaves all transaction details in the body, and the relatively light block header can only hold a Merkle root of these transactions and other configured attributes. [23]

The Merkle tree enables nodes to verify the integrity of the chain without storing the complete Blockchain [23].

In immudb the client trusts the storage because he can proof and recalculate it. Every client, which can be an application or an auditor, saves the interim report of the Merkle-tree regularly. Basically, every client has a Checkpoint-Blockchain with a rolling checkpoint-window that is verified by the client to prevent any tampering with the data and ensure immutability. CodeNotary offers auditor programs that run the verification on a second basis.

#### Availability of the Project

Their own platform immudb is open source[1].

[1] https://github.com/codenotary/immudb

#### Justification for DLT

The main reason for using DLT in this project is that everything needs to be publicly visible. It was crucial for the project that many different companies can get together in the Consortium Blockchain, and all have the same data and the same knowledge to verify the data.

The second reason for DLT is that many actors are in the system, which need a way to interact with one another securely.

#### What the Customer says

CodeNotary Inc. emerged from vChain Inc. When the vChain team initially started with the project, there was a huge distrust against any Blockchain-Technology, and any customer that heard the word Blockchain immediately walled. So, the team changed their approach to telling the customers that they have a distributed, trustable platform protected by multiple companies and directly offered them a test of the product.

The customer base, especially the small ones, accepted the system relatively quickly. Securing the internal build process of big customers has been a huge success, according to Mr. Zimmer.

### Technology Choices

#### The Platform

First Parity Ethereum on a private chain then immudb. Parity Ethereum is one of the most prominent interoperable implementations of the Ethereum client software and is written in Rust and is an implementation of a full-node Ethereum client and DApp browser [4].

#### Used Frameworks and Design Pattern

The CodeNotary team used the Design Pattern provided by the Parity team. For example, this led to a configuration that excluded Cryptocurrency from the project in the Parity Ethereum chain.

#### Public or Private

The decision for a private platform was very clear from the start since the team needed a platform that could compute a high number of transactions at extremely low or no cost, which is unfeasible with a public Ledger, according to Mr. Zimmer.

#### Justification for the Platform Choice

CodeNotary used to run the system on Parity Ethereum with fees set to zero but switched to their own platform called immudb.

Initially, the team decided on Parity Ethereum instead of other Platforms because the technology was much further than comparable platforms back when they started developing Mr. Zimmer claims. Furthermore, it was much easier for CodeNotary to find staff that was able to work with Ethereum tools. Lastly, it is still much more complex to set up a project on Hyperledger Sawtooth than on Parity Ethereum, Mr. Zimmer claims.

The Team also discussed Corda, Quorum, and Hyperledger Sawtooth. Sawtooth did not seem appropriate from the teams' perspective for the use-case and was too complex for it. Corda was not investigated in detail and was kicked out of the decision process together with Sawtooth. Finally, there was only the decision between Quorum and Parity Ethereum, and the team decided on Parity because they discussed with the Parity Team in Berlin and the support and tool base of the platform persuaded the team to use it. Additionally, Parity Ethereum was the fastest Platform back then when the team ran tests on it.

The main reason for CodeNotary to build their own platform is that they have encountered customers during the development who had millions of objects that needed to be stored on the Ledger. However, DLT is too slow for this specific use-case because of the distributed database's latency. CodeNotary could only achieve about sixty thousand transactions per minute with Parity Ethereum but needed around ten million transactions per minute for their largest customer. Since they could not satisfy the large customers' needs, they had to switch away from Parity Ethereum.

#### Used Frameworks and Design Pattern

The Project of CodeNotary can be represented using the Feature catalog:
**Identity Management**: Asymmetric Cryptography
**Unique Asset Management**
**Code Execution on the Blockchain**: EVM Architecture Pattern

### Development Process

#### General Approach

The CodeNotary Team uses an agile approach running a modified version of scrum.

#### Used Tools

The Team uses the following tools for their development process:

- Truffle suite
- GitHub Project

## 5.4. Lakoma - Insurance of Sustainability

### Meta Information

| | |
|---|---|
| Interviewee | Christoph Langewisch |
| Position | Founder |
| Company | Lakoma |
| Company Size | Small |
| DLT Experience | Four Years, several projects |
| Company founded | 2017 |

### The Project

#### Summary of the Project in one Sentence

Proof of sustainable production across company borders

#### Project State

Pilot - 8 Month in - about to go into production (Development)

#### Problem Statement

Lakoma wants to solve the sustainability issue in today's supply chains. The UN has defined 17 criteria on how companies should produce and deliver sustainable products. Such as no child labor, climate protection, etc. It is becoming increasingly important for companies responsible to know whether the compliance rules are actually being enforced. To be able to present the data credibly, immutability, trust, and transparency need to be enforced at every step of the supply chain. The supply chain brings many stakeholders together who work, for example, as supplies. Therefore, a system is needed that combines immutability, trust, and transparency across company borders within the supply chain.

In a concrete example, a farmer has produced and delivered products. For the subsequent partners in the supply chain, it is important that the farmer is certified and has worked, e.g., according to the organic certificate; That his people have been paid, where each step of the production conducted done until the product arrived in Europe, etc.

#### How the Problem is solved

In the supply chain, the crop is produced by a first entity; Then processed by a second entity. After that, it is passed to a third entity for logistics and shipping. The goal is to track the asset until the final customer consumes it. Assets are created on the public Ethereum Blockchain when they are first entered into the system for the first entity. E.g., at the harvest, where the picker and collector confirm the validity of the entered

assets. At each step and each transfer from one entity to another, further information about the process is added to the asset. From a technical perspective, Lakoma creates a smart contract where the data is stored, and workers can be identified with their digital identities. The fees that need to be paid are paid by the customer but are layered by Lakoma, so the customer does not have to deal with DLT and only pays Lakoma in Fiat money.

Therefore, the main issues are solved through a public permissionless ledger that offers immutability, trust, and transparency by default. New companies are easily onboarded and can see the history because of the transparent nature of a permissionless ledger. Specific workflow problems are solved with assets that are managed by smart contracts.

Like the picker, collection partner, or operation partner, all relevant stakeholders that participate in the system can register by themselves and are activated by the higher instance. For example, a collection partner registers a picker, and this digital identity is secured by saving it on the Blockchain. To interact with the system, entities are identified by the underlying system with their digital identities that have unique IDs and digital signatures assigned to them.

The specific assets like crops, products, etc., are represented using the ERC-1155 protocol on the Ethereum Blockchain. This is because the team claims that they can combine fungible and non-fungible tokens that help to secure the entire supply chain for each product. These assets are transferred between entities at checkpoints of a real process like logistics, which are monitored and execute the transfer on the Blockchain. When information needs to be added to assets, it is either stored on a cloud service or the Blockchain-service. Cloud focuses on normal data like personal data and the Blockchain service on compliance data that needs to be digitally secured.

#### Availability of the Project

Parts of the project are open source, but it depends on the customer. More parts may be open-sourced when contracts with the customer allow it.

#### Justification for DLT

According to Mr. Langewisch, DLT was chosen in this use-case because it would not be possible to realize it without DLT.

- Information is propagated peer-to-peer without central instances
- Digital Identities are created for every collector
- Pickers verify that collectors give correct data and approve it with a transaction

#### What the Customer says

The customer has given positive feedback so far and approved going from pilot to production. Once a whole harvest cycle is completed, a full review can be given.

### Technology Choices

#### The Platform

Lakoma uses the Ethereum Mainnet

#### Used Frameworks and Design Pattern

Lakoma uses a few frameworks and design pattern to bootstrap their project more quickly:

- Solidity toolchain for Smart Contracts
- ERC20 for Tokens
- Open-Zeppelin for security in Smart Contracts
- Truffle for development

#### Public or Private

Lakoma decided on a public permissioned platform due to the following points:

- They wanted to have an open standard that has a huge community
- From their perspective, the continuous development in Ethereum is more extensive than in Hyperledger Fabric
- Ethereum has a higher brand awareness
- The Know-how of the team is mainly Ethereum
- It was more of a gut feeling than scientific consideration

#### Justification for the Platform Choice

The Ethereum mainnet was chosen as the platform of the product because:

- A permissionless platform can be used without investing in infrastructure
- The time to market is shorter with an existing DLT network
- An own DLT network for a pilot is more expensive than transaction costs in the Ethereum Mainnet
- The team can focus on the use-case and not the platform

#### Used Frameworks and Design Pattern

The Project of Lakoma can be represented using the Feature catalog:
**Cryptocurrency Minting**: Pre-Mined Cryptocurrency
**Cryptocurrency Storing**: Account Pattern
**Cryptocurrency Transfer**: Single Signature Send
**Identity Management**: Asymmetric Cryptography
**Unique Asset Transfer**
**Unique Asset Management**: Digital Twin
**Code Execution on the Blockchain**: EVM Architecture Pattern
**Storing verifiable Data Off-Chain: Hash Anchoring Pattern**

### Development Process

#### General Approach

Lakoma uses an iterative and agile development process. All of their problem statements are split into smaller chunks and solved use-case by use-case. For the customer, this development process is represented by multiple workshops that are being held to develop further requirements and specifications with the customer.

#### Used Tools

GitLab is used to manage the Sourcecode, tasks, and requirements, etc. When enough features have been developed, quality assurance is performed.

## 5.5. Chaincentive - Digital Incentives

### Meta Information

| | |
|---|---|
| Interviewee | Marijo Radman |
| Position | Founder |
| Company | Chaincentive |
| Company Size | Small |
| DLT Experience | Four Years |

### The Project

#### Summary of the Project in one Sentence

Incentives for the positive change in behavior

#### Project State

The Project has a finished prototype (Development)

### Problem Statement

In companies, social behavior develops between employees. These behavioral patterns arise through their default attitude. As this social behavior is not optimal most of the time, the project aims to influence and optimize the employees' behavioral patterns. As an example: There is a normal company with employees and systems. Now a new system is introduced, and employees are skeptical by default because they know the old system and not the new system and are very careful to adapt it. Especially old employees or employees who are about to retire do not tend to use new systems. Since companies have invested time and money into the new system, they want employees to use it.

### How the Problem is solved

Chaincentive develops a system that encourages employees to use the new system. Tokens are generated and then given to employees for good behavior and can be used for rewards such as vouchers. There are three levels of psychological influence the system offers:

1. A small economy within the company is created where employees can exchange tokens with each other
2. Employees are rewarded for good behavior with the token
3. Employees know that they are rewarded for good behavior

An example of a reward that has already been implemented is a voucher for Jochen Schweizer. Chaincentive is paid by the customer for the system integration and pays their reward partners such as Jochen Schweizer, that exchange vouchers for Services, fiat to "buy back the tokens".

### Availability of the Project

The Project is not open source since it contains sophisticated software to connect to the customer's systems, and the customers do not want this to be public.

### Justification for DLT

The project could theoretically be implemented without DLT; However, a few downsides occur without DLT.

- Blockchain-Technology provides pseudo-anonymity which would not be possible without it
- DLT makes the data tamper-proof and immutable
- Transparency is offered to the customer so that the data can be reviewed

- Once the system reaches across companies; it is harder to implement without DLT
- By using DLT, the problem is solved more elegantly
- By using DLT, a distributed platform is created which offers business benefits to all partners

#### What the Customer says

Privacy is most important to the customer. Chaincentive sells the product to companies over their Human Resources Department as they are the gatekeeper for such systems. Chaincentive does not advertise with DLT to their customers, and they mainly argue with the psychological aspect. Companies know that the offered system is cutting edge and are willing to join the experiment.

### Technology Choices

#### The Platform

Chaincentive uses the Kaleido platform with Enterprise Ethereum.

#### Used Frameworks and Design Pattern

Chaincentive uses a few Frameworks and Design Patterns to bootstrap their project quicker:

- ERC20 interface for Token
- Truffle and Drizzle for Smart Contracts
- Open Zeppelin for Smart Contract Design Pattern
- React and Web3 for Frontend Development

#### Public or Private

Chaincentive uses a private blockchain because of two main points:

- On public Blockchain transactions are visible, and their customers do not want that because of privacy aspects
- The transaction fees on the public Ethereum Network are too high

#### Justification for the Platform Choice

The platform was primarily chosen because it was the easiest and well known for the team to start with. However, additional arguments can be made:

- Hyperledger does not offer Token support by default. The developer must use libraries and re-create tokens in Chaincode[2].
- The platform is supported by all major cloud providers allowing Chaincentive to spin up nodes at different providers to allow for more decentralization.
- It supports the known frameworks such as Drizzle and Truffle.
- Developers in the company have completed courses about this platform and knew how to use it.

#### Used Frameworks and Design Pattern

The Project of Chaincentive can be represented using the Feature catalog:
**Cryptocurrency Minting**: Pre-Mined Cryptocurrency
**Cryptocurrency Storing**: Account Pattern
**Cryptocurrency Transfer**: Single Signature Send
**Identity Management**: Asymmetric Cryptography
**Code Execution on the Blockchain**: EVM Architecture Pattern
**Storing verifiable Data Off-Chain: Hash Anchoring Pattern**
**Interoperability**: Atomic Cross-Chain Swaps

### Development Process

#### General Approach

Chaincentive uses an Agile approach to development. It uses an adaption of Scrum but without Product Owner and Retrospectives.

#### Used Tools

Chaincentive uses a few tools to streamline their development process:

- Trello
- GitHub
- Microsoft Teams and Zoom

#### Deployment Process

On a new deployment, Smart Contracts are "killed" and newly deployed.

[2]Hyperledger Fabric 2.0 offers FabToken which promises to make this easier

## 5.6. FfE - Proof of Origin for Electricity

### Meta Information

| | |
|---|---|
| Interviewee | Andres Zeiselmair |
| Position | Scientific Employee |
| Company | Forschungsstelle für Energiewirtschaft (FfE) |
| Company Size | Medium |
| DLT Experience | 4-5 Years |
| Company founded | 2001 |

### The Project

#### Summary of the Project in one Sentence

Blockchain-based proof of origin for electricity in a high resolution

#### Project State

Before Pilot (Planning)

#### Problem Statement

The Project InDEED wants to enable Blockchain-based proof of origin for electricity in a high resolution to create new verifiable energy products for the consumer without relying on many intermediaries.

The widespread adoption of smart meters in Europe enables new business models that leverage data-based value creation through new possibilities for consumer-producer interactions. An increasing number of actors - both on the producer and consumer side - which interact with the Energy system combined, with the growing importance of real-time processes, lead to increased automation and scalability requirements. Therefore, there is a need to replace historically grown legacy systems with new systems capable of delivering the automation and scalability needed to meet the market requirements, offer additional features, follow regulations, and ensure compatibility [5].

Currently, the origin of electricity is very opaque, which is also the main problem. Furthermore, the process of identifying the origin of the electricity is based on many intermediaries. For example, it is possible to buy certified green electricity, but this contract is very basic and not very specific. Those contracts are very basic because the temporal resolution is very simple, and the allocation of origin energy quantities to consumption is opaque. The goal is now to prove the origin of the electricity without an intermediary such as the TÜV and to achieve a higher resolution through Blockchain-Technology.

According to Mr. Zeiselmair, the Blockchain-based System should fulfill the following technical requirements:

1. Compatibility with certified smart metering solutions and increased automation to foster scalability, security, trust, and interoperability
2. Increased time resolution and guidance for individual customer behavior adaptation
3. Scalability and micro-transactions to facilitate millions of small-scale renewable energy systems
4. Implementation of simple physical boundary conditions (e.g., cross-country grid constraints for imports) to reflect realistic cross-border energy quantities

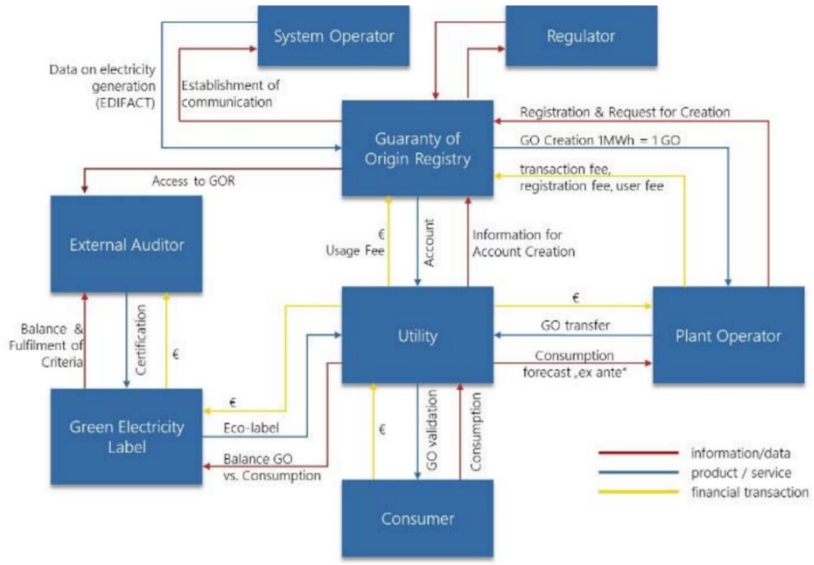



Figure 5.1.: The complex process of the current German guarantees of origin register (GOR) [5]

In Figure 1, the complex process of the German GOR with its many involved parties is displayed. From the figure, it is easy to see that many parties are involved in this process and that there is much information, financial transactions, products, and services involved.

**How the Problem is solved**

Ideally, the consumption data is collected directly from the consumer using a smart meter infrastructure. Furthermore, the feed-in data is measured directly at the generation plants using a similar infrastructure. Other data transmitted by the infrastructure are, for example, the name of the plant, geolocation of the plant, installed power of the plant, information about the owner of the plant. The system aims to achieve a more granular resolution of the electricity data than a quarter of an hour.

Now the customer has the possibility to say that he wants to get the electricity from a radius of 50km, preferably from solar plants, then wind power plants, and lastly from hydroelectric plants. Thus, the customer can be offered a new electricity product in which he has full control over the origin, eliminating the non-transparency of the current established – legacy – system.

**A concrete example**: The goal of the system is to be able to tell every quarter of an hour from which system the electricity comes. Generation and consumption of electricity are fed into a digital platform every quarter of an hour. This platform then uses a linear optimization algorithm to calculate supply and demand allocation based on a distance matrix. The optimization result is written to a Blockchain, and the zero-knowledge proof is used to prove that the calculation was performed correctly.

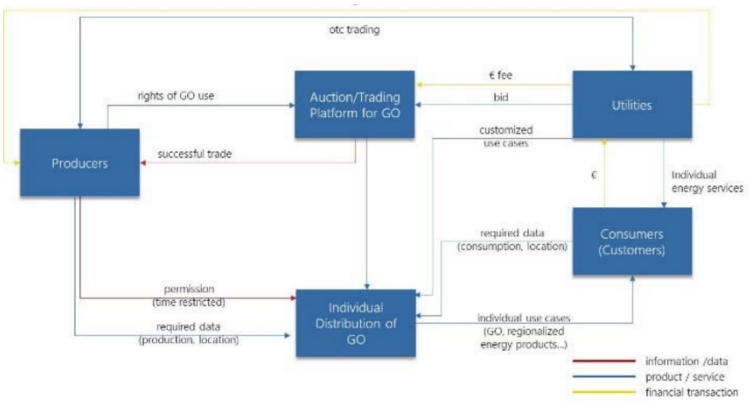



Figure 5.2.: The Architecture of the proposed Platform [5]

The platform proposed by the team is split into two components: One for trading the rights of future produced renewable energy certificates. Here the system has to estimate energy quantities for all energy resources for the future staggered in days, weeks, and months. Currently, these usage rights are traded over-the-counter or via brokers and markets. The second component is responsible for downstream linear optimization and distributing the certificates to the customers. One goal of the linear optimization could be to optimize electricity distribution to maximize revenue from regional direct marketing or the distribution according to the preferred energy mix for any customer [5].

### Availability of the Project

The project will be available open-source as soon as it is in a later state. Furthermore, a paper about this use-case has been published by Alexander Bogensperger and Andreas Zeiselmair called "Updating renewable energy certificate markets via integration of smart meter data, improved time resolution and spatial optimization" [5].

#### Justification for DLT

The project is a use case in which there are many partners and many actors. All partners and actors must be brought to a single platform that unifies all data and actions.

Furthermore, transparency is essential. In the case of the project, it is crucial that it can be proven beyond doubt that the calculation of the optimization algorithm, which combines supply and demand, has been performed correctly.

Also, privacy is a big issue. The data transferred from the smart meter to the platform is all personal and must not be seen by the actors.

As was explained, on the one hand, transparency is needed to prove that the data has been calculated correctly. On the other hand, the personal data must not be public. Therefore, DLT is used in the project, as it is one of the only technologies that can be used to prove the correctness of data without intermediaries actually seeing them.

#### What the Customer says

There is no customer feedback so far, as the project is still in a very early phase. In general, the customer base is generation plants that are no longer eligible for EEG subsidies and private and commercial end customers who can benefit from such a system.

### Technology Choices

#### The Platform

So far, no final decision has been made regarding the platform. However, the team has made a pre-selection between Hyperledger Fabric and Ethereum, with Ethereum either being on the Mainnet or a private ledger.

#### Used Frameworks and Design Pattern

The Team currently only considers using the Zero-Knowledge Proofs toolbox Zokrates[3] of the TU Berlin. Further Frameworks or Design Pattern have no been decided on yet.

#### Public or Private

No final decision has yet been made between public and private Blockchains. For the InDEED team, the following points speak for a public chain: Greater security and greater transparency. In terms of transparency, more open is better for the team. According to the InDEED team, private Blockchains generally perform better when compared to public blockchains. Transaction fees and infrastructure costs are not yet considered a major problem by the team.

[3] https://www.ise.tu-berlin.de/menue/projekte/zokrates/

#### Justification for the Platform Choice

Against the Ethereum Mainnet speak the high transaction cost.

In Hyperledger Fabric, many features can be used "out-of-the-box", which is - according to the team - not the case with Ethereum. Only these two platforms come into question since code execution on the Blockchain is urgently needed, and this is realized in Ethereum with EVM and in Hyperledger with chain code. Furthermore, it is these two platforms that make the biggest development steps from the team's perspective.

#### Used Frameworks and Design Pattern

The Project of InDEED can be represented using the Feature catalog:
**Identity Management**: Self-Sovereign identities (SSI), Asymmetric Cryptography
**Information Broadcasting**: Event Emits and Event Listeners (planned)
**Unique Assets Transfer**
**Unique Asset Management**: Matching Unique Assets with Digital Identities
**Code Execution on the Blockchain**: (No yet decided on the architecture)
**Storing verifiable Data Off-Chain: Hash Anchoring Pattern**

### Development Process

#### General Approach

The Team uses an agile development process. The team utilizes sprints and tries to run scrum without a scrum master but with a product owner.

#### Used Tools

The InDEED team uses tools for development:

- GitLab for Sourcecode and Tasks
- Asana as Kanban-Board

#### Deployment Process

There is no deployment process in place yet because the project is before the first pilot.

## 5.7. BMW - Traceability of the Supply Chain

### Meta Information

| | |
|---|---|
| Interviewee | Felix Gerbig |
| Position | Blockchain Engineer / Software Engineer |
| Company | BMW Group |
| Company Size | Large; Small Team |
| DLT Experience | Through Master thesis; 1.5 Years. Bitcoin before |
| Company founded | 1916 |

### The Project

#### Summary of the Project in one Sentence

Part Chain is about traceability in the whole supply chain of BMW

#### Project State

Extended Pilot Phase: Post PoC (Development)

#### Problem Statement

Part Chain is a B2B system in which BMW and its suppliers can share and track vehicle and component information between the participants. In case of a recall, BMW needs to know exactly which components have been installed. This knowledge is not completely available yet, but only partially. For example, BMW knows which airbags and tires are installed, but it is very difficult or even impossible to find out which sub-components are installed in the airbag because only the T1 supplier has this knowledge. In case of a recall, the whole history of a part has to be traced back, and there are much information and too many inconsistencies, which in turn leads to delays in this time-critical process.

#### How the Problem is solved

Part Chain links all components on a ledger, and in case of a recall, the complete supply chain can be fully tracked. For example, BMW now knows that these three light modules were installed in a headlamp and then in the vehicle.

On the one hand, component information such as production time, production location, supplier and component ID, and manufacturer company is stored in the form of a hash and on the Ledger. When the data is called, it can be verified that the data has not been changed.

On the other hand, the Ledger is used as a public access control, in which the business relationships are mapped, so that it can be ensured that information is only passed on to companies that should have it. For example, a supplier is not in a business relationship

with BMW, and this information is stored in the Ledger. Now, this supplier tries to write data to BMW but cannot do so because there is no business relationship.

The Ledger is also used as a kind of global lookup. It is stored in which supplier has built which component so that the recipient of the component can look up who built it. BMW offers two options for joining the network. Either a new supplier hosts a fabric node themselves, including all additional software needed for Part Chain or they use the offering "trust as a service," which offers an API of BMW and a link to a corresponding frontend.

**Asset Production Workflow**: A Supplier produces an asset. The asset is then registered in Part Chain in two steps: Asset details are stored in an Off-Chain database, and the Private Data Collection (PDC) of the supplier and the Proof-Of-Ownership (Hash of the asset details) is stored together with the Membership Service Provider (MSP) Id of that supplier on the Ledger. The Asset is then shipped and received.

**Asset Lookup Workflow**: The process of looking up a part on the ledger must be connected to the physical process of scanning an item. For this, every part has its own unique 30-digit Id, which is printed by the supplier on the part. The scan of the part allows everyone to look up this unique id and the corresponding part on the network using the functions of the Chaincode. The unique id is only stored as a hash on the Ledger itself, so the only supplier that knows the unique id can look it up.

**Child Part Data Storing/Asset Exchange**: Once BMW receives an asset and scans it, the corresponding vehicle information is written into the PDC of the supplier, and an event is emitted. The event prompts the supplier to write the data about the scanned part into the PDC of BMW after the vehicle information written by BMW is verified by the supplier using the Ledger. This exchange works because the PDCs of every participant is write-only.

**Data Privacy/Data Visibility Concept**: It is possible to control down to the individual component in fine granularity, which information a supplier sees and which not. BMW has the information for a vehicle off-chain, and the suppliers write the information for the individual components into the PDC of BMW. The supplier only sees in the PDC of BMW exactly for those vehicles for which he is a supplier, and he only sees the components for which he was the supplier.

#### Availability of the Project

The project will be open source starting January 2021

#### Justification for DLT

In general, the problem would be solvable with a traditional system, as there are no trust issues. BMW and its suppliers have known and trusted each other for decades.

For BMW, DLT is a framework to initiate a new way of thinking. So that data can be shared across company boundaries. A paradigm shift from "Yours and Mine" to "Our" should take place.

The supplier does not necessarily want BMW to have all its data, so equal rights should be established so that the supplier also has an advantage from the system because BMW instructed that the supplier also passes on the data. So, in the end, everyone gets a piece of the cake, and there is no central authority that governs the data.

The data of the T2 and T3 suppliers are out of BMW's reach, but this data is also crucial for tracking the whole supply chain, and this new approach would give those suppliers an incentive to put their data in the system.

#### What the Customer says

The supplier "Automotive Lighting" liked the system that much that they are now also supporting development. At the time of the interview, no other suppliers have been onboarded yet.

### Technology Choices

#### The Platform

Part Chain uses Hyperledger Fabric 2.1

#### Used Frameworks and Design Pattern

Part Chain utilizes the following Frameworks and Design Pattern:

- Microservice architecture
- One channel concept
- One Chaincode

#### Public or Private

The Part Chain team decided on a private Ledger because business-critical data is stored on the Ledger, which cannot be public. Access control and privacy are major points in Part Chain, which is not possible with public Ledgers. The team decided against a public Ledger with encrypted data because encrypted data is not easily searchable, and therefore, the main feature of this project – the tracking of an item throughout the supply chain – would be much harder to implement.

#### Justification for the Platform Choice

For the Team, Hyperledger Fabric is the only permissioned blockchain to be taken seriously for enterprise software because of its big community, maturity, and its continuous development. Furthermore, it contains all the required features for BMW, such as access control and private data collections. Lastly, Hyperledger Fabric is open-source, which is a major point because BMW aims to have the full stack open source.

#### Used Frameworks and Design Pattern

The Part Chain project can be represented using the Feature catalog:
**Identity Management**: Certificate based Authentication
**Information Broadcasting**: Event Emits and Event Listeners
**Unique Asset Transfer**
**Unique Asset Management**
**Code execution on the Blockchain**: Execute-order-validate Architecture Pattern
**Storing verifiable Data Off-Chain: Hash Anchoring Pattern**

### Development Process

#### General Approach

Part Chain is developed with an agile Scrumban approach. However, there are no planned sprints. There are internal reviews and planning and a single merge to production every week.

#### Used Tools

In Part Chain, the following tools are used:

- Jira, Bitbucket and Confluence (Atlassian Suite)
- CI/CD with AWS native

#### Deployment Process

Part Chain follows monthly release cycles for deployment on production. If there are changes to the Chaincode, it will be manually re-deployed.

## 5.8. Bernstein - Secure the Ownership of Intellectual Property

### Meta Information

| | |
|---|---|
| Interviewee | Marco Barulli |
| Position | Founder |
| Company | Bernstein |
| Company Size | Small |
| DLT Experience | Started with Bitcoin in 2010 |
| Company founded | 2016 |

### The Project

#### Summary of the Project in one Sentence

Secure the Ownership of Intellectual Property

**Project State**

Shipped (Production)

**Problem Statement**

Registering and proofing the intellectual property rights of a project can be tedious because it contains steps such as creating a trail of evidence of the innovation processes going on in the company and creating the tail so that its integrity is beyond any doubt.

There is a lot of know-how and non-core technology that needs protection around the patentable knowledge in any company. There can be a situation where the need to proof changing things quickly and efficiently is required. For example, in fashion, it may be necessary to prove that you have been working on a new garment from the initial sketches to the final CAT-models for production, the brand and marketing plans, and everything in between. It may be necessary to prove that this specific design is yours, which is easier when the whole innovation chain is available. With this proof, you can attack the counterfeit product on an intellectual property level and ask them for statutory damages and financial damages.

**How the Problem is solved**

Bernstein is a gateway to the Bitcoin Blockchain. It allows an innovator to create a transaction to proves the existence, integrity, and control of ownership over any digital file.

Users can create a project in their innovation process, press a button, and a certificate is created and saved on the Bitcoin-Blockchain. If a user changes the project like modifying or creating a file and presses the button again, another certificate will be created linked to the previous one to create a chain of certificates. They will end up having a time machine where they can move back and forth in time, proving what the status of the knowledge in the project was.

From the user's perspective, it is pressing a button to get a certificate every time one is requested. However, this creates not only the certificate but also an official EU and Chinese digital timestamp, which is useful in legal procedures.

The certificate creation workflow functions the following: A transaction is created and propagated to the Bitcoin-Network. This transaction has three outputs. Bernstein controls one, one is derived from the rightsholder's actual data, so the digital fingerprint of the data is used to create one of the transaction outputs. The last output is derived from the credentials of the user.

That transaction includes the following information: Proof that certain data represented by the second signature was existing. Additionally, the information about when the network approved the transaction and that the actual user of the data can only spend it.

For the workflow to work, Bernstein creates a Bitcoin-Wallet for every user that signs up. They specifically create a hierarchical deterministic wallet. Within this hierarchical

deterministic wallet, a new branch is assigned to every innovation project that the user creates within the Bernstein-Interface. Bernstein initiates this wallet with funds (about 30 000 satoshi, which equals 5.40$ as of 11.12.2020). Those funds are used every time the user initiates the certification creation to create a transaction like mentioned before.

The system of Bernstein is designed to complete work in the browser. Every transaction is created and signed in the browser and then pushed to the network. So, in addition to being a hierarchical deterministic wallet, it is also a non-custodial wallet. In DLT, there is a distinction between custodial and non-custodial wallets. In custodial wallets, the company providing the wallet has control over the private keys that hold the funds, wherein in a non-custodial wallet, the user has direct access to the private keys to control the funds.

Bernstein uses a Zero-Knowledge-Architecture, which means that all the cryptographic functions for managing the encryption of the data are done right in the client's browser. With this approach, Bernstein never gets access to the client's data and the client, therefore, is the sole owner of his data and the hierarchical deterministic wallet.

This data can be verified either by using the process of Bernstein, which is centralized, or by following their guide, which is a five-step process:

1. Compute the hash of the project as sha256sum.
2. Create a Bitcoin private-key from the hash created in Step 1.
3. Derive the associated Bitcoin public-key
4. Create the Bitcoin MultiSig address with the project data public-key, owner public-key, and Bernstein's public-key
5. Validate against the Blockchain by finding an identical address in the list of outputs of the transaction

Using the decentralized and non-custodial approach, the certificates are still valid even if Bernstein went out of business.

#### Availability of the Project

Closed Source

#### Justification for DLT

Blockchain enables decentralized public registries where Bernstein can store and verify Intellectual Property (IP)-Assets without relying on any central authority. This use-case was not solvable without Blockchain-Technology. According to Mr. Barulli, "Blockchain is a publicly available registry that is uncensorable, resilient and open to everyone and that is exactly why we need the Blockchain".

#### What the Customer says

Bernstein has paying customers that are very happy with the current system. The general opinion of customers is very polarized. For example, some IP-Lawyers are willing to have a tool like this because it enables them to do great things for the client. However, others do not care about it because they file and prosecute for patents and trademarks.

### Technology Choices

#### The Platform

Bernstein uses Bitcoin

#### Used Frameworks and Design Pattern

None, besides from the ones mentioned in the Feature Catalog

#### Public or Private

For Mr. Barulli, private Blockchains were never the right choice as, in his opinion, those do not offer benefits compared to other cryptographic solutions that have been out for years. In general, the team only considers Blockchain-Technology and not the generalization of DLT.

#### Justification for the Platform Choice

According to Mr. Barulli, there was no other public platform available when they started the project in 2015, with the other possibility only being an early-stage Ethereum network. Because of "the openness for security issues," the team did not feel comfortable choosing Ethereum at that time and went with Bitcoin.

Bitcoin-like Blockchains such as Litecoin were not considered because of their immaturity. The team did not see the benefits of slightly lower fees for the project as they have never been an issue. It was more critical for Bernstein to have a sizeable and active community behind the Blockchain of choice.

#### Used Frameworks and Design Pattern

The Project of Bernstein can be represented using the Feature catalog:
**Cryptocurrency Storing**: UTXO Pattern
**Cryptocurrency Transfer**: Confirmation Pattern, Single Signature Send
**Identity Management**: Asymmetric Cryptography
**Storing verifiable Data Off-Chain: Hash Anchoring Pattern**

## Development Process

### General Approach

Bernstein uses an agile development approach with sprints. A focus is set on testing within the Bernstein team.

### Used Tools

Bernstein uses the following tools to improve their development efforts:

- Jira for the general development approach
- GitHub for storing the Sourcecode
- Notion for sharing information

# 6. Summary and Evaluation

In this chapter, a summary for the use-case studies is given, and the direct answers to the research questions are given.

## 6.1. Statistical evaluation of the Case Studies

In this work, we conducted eight use-case studies.

Of the interviewees, four had the title *Founder*, two *Chief Technology officer*, and one the title *Software Engineer* and *Scientific Employee*, respectively.

Furthermore, six companies that the interviewees are working for are small (3-12 people), and two are medium (13 - 50).

All teams that the interviewees are working on were created within the last five years. However, two companies have been founded before the year 2000. Each of the interviewees had at least one and a half years of experience with DLT at the interview time.

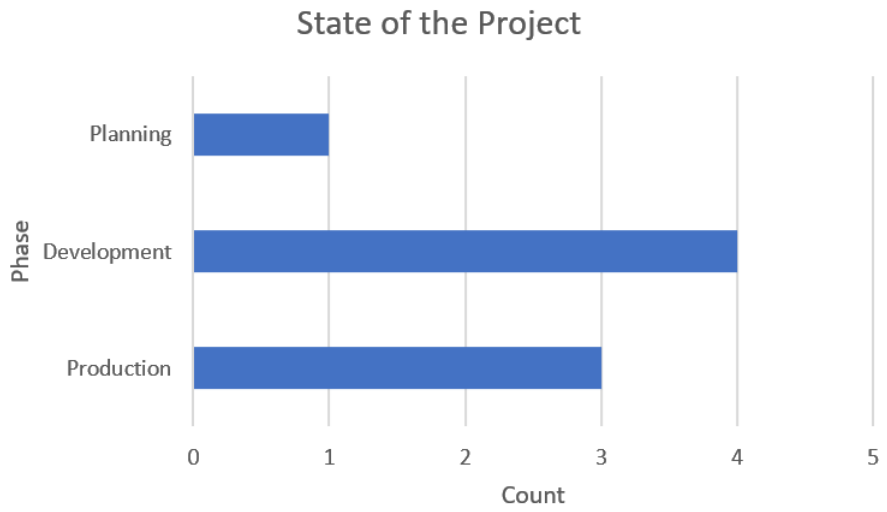



Figure 6.1.: State of the Projects

From the 8 projects, one is in the Planning phase, four in the development phase and three in the production phase.

Five projects used permissioned platforms with two Hyperledger Fabric uses and one use of each Hyperledger Besu, Kaleido, and Parity Ethereum. Two projects decided

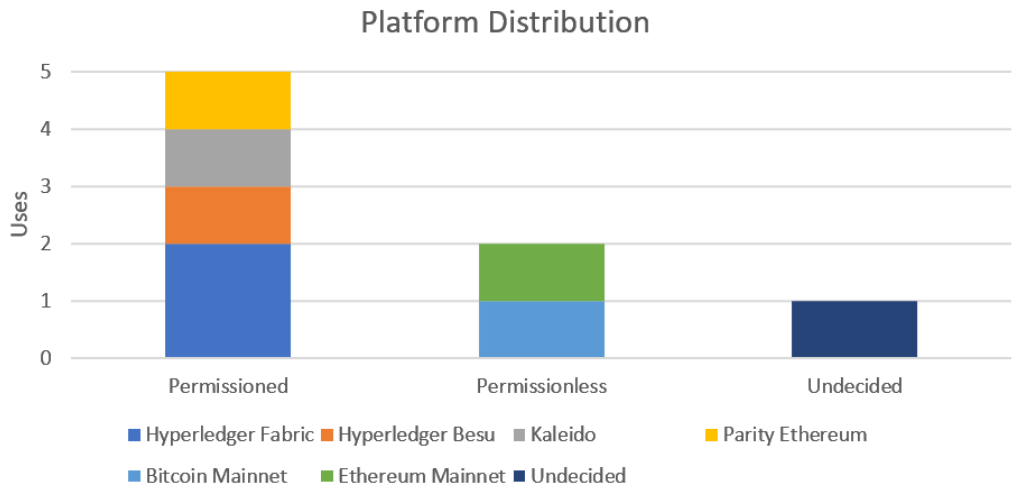



Figure 6.2.: Platform Distribution

on a permissionless platform, with the platform of choice being Bitcoin Mainnet and Ethereum Mainnet. One project in the planning phase has not decided on a platform yet (Use-case 6.1).

The feature usage of the feature catalog is depicted in A.2 (appendix). The total use of the features is given on the Y-Axis, and the single features are named on the X-Axis. The projects are indicated by coloring.

All interviewees claim to use an agile and iterative development process.

## 6.2. Summary

In this work, we have shown that the different distributed ledger technologies can be split into their respective features (section 2.1.1, section 2.1.2).

In section 2.2 we divided business applications related to DLT into the categories Cryptocurrency, DLT-based Business Problem Solvers, and Ecosystem Software and have chosen to focus on DLT-based Business Problem Solvers for this thesis because those are the most relevant for economic use-cases.

The feature summary in chapter 4 divided common features of distributed ledger system into functional and non-functional features and explained them briefly. We discovered twelve functional feature categories: Identity Management, Unique Asset Management, Unique Asset Transfer, Code Execution on the Blockchain, Information Broadcasting, Storing verifiable Data Off-Chain: Hash Anchoring Pattern, Storing Data On-Chain and its Usage, Interoperability, Governance, Cryptocurrency Minting, Cryptocurrency Storing and Cryptocurrency Transfer. Furthermore, we presented the forces and drawbacks of the non-functional features Scalability, Privacy, and Security.

Additionally, we presented eight use case studies that were created by interviewing

industry experts. The main goal for these is to present meta information, an overview of the project, technology choices, and the development process of the use-case. This is to raise awareness for the decision that teams are currently creating new distributed ledger systems and to give justifications why DLT is needed in certain use-cases.

## 6.3. Evaluation

This section reviews the research questions that were asked at the beginning of this work and boil an answer down to its essence and present it.

**Q1 - What are common features that distributed ledger applications share?**

a) Which applications that use distributed ledger technology are being used and created at the moment?
We can only partly answer this question as it is pretty broad. In this work, we presented eight projects that are currently either being created or used. IBO - Product Tracing (5.1), CashOnLedger - Payment processing (5.2), CodeNotary - Create Trust in Digital Objects (5.3), Lakoma - Insurance of Sustainability (5.4), Chaincentive - Digital Incentives (5.5), FfE - Proof of Origin for Electricity (5.6), BMW - Traceability of the Supply Chain (5.7) and Bernstein - Secure the Ownership of Intellectual Property (5.8). Contradictory to the approach presented in section 1.3 to only use the results of the literature review to create the feature catalog and not to describe projects in this work.

b) Which distributed ledger technology do these applications use?
As presented in section 6.1 and section 6.2, the projects use Hyperledger Fabric (2), Hyperledger Besu (1), Kaleido (1), Parity Ethereum (1), Bitcoin Mainnet (1), and Ethereum Mainnet (1).

c) Is there related work regarding patterns for distributed ledger technology?
We found a pattern catalog in section 3.3 and summarized the most important patterns.

**Q2 - How can these features be categorized and formalized?**

a) Which technologies are used most and how are they implemented?
We can not give a qualified answer to this question yet as more research is required answering this question. This will be part of the future work.

b) Which different categories do the features fit in?
We identified 15 different feature categories in chapter 4 divided into 12 functional features (section 4.1) and three non-functional features (section 4.2).

c) How can these features be summarized to understand for no DLT savvy engineer?
We presented several features for each category in section 4.1 and section 4.2

respectively, which are summarized and easy to understand for developers that are new to DLT.

**Q3 - Which decisions are made in the current DLT developments?**

a) Which problems are the projects that are currently in development solving?
We identified three comprehensive and distinct problems:

* cross-organisational data storage to enable use cases like product and supply chain tracing (5.1, 5.7)
* proof of origin, ownership or authenticity of assets (5.3, 5.4, 5.6, 5.8)
* payment processing (5.2, 5.5)

b) What are re-occurring questions in the architecture of enterprise DLT applications?
We found several decisions that are made creating enterprise DLT applications. First, the decision between a permissioned or permissionless platform. Second, the specific platform of the category permissioned, permissionless, or the decision to create an own platform. Third, frameworks and design patterns within the platform of choice.

**Q4 - How can a feature summary be verified?**

a) Which technology features of the catalog are implemented in business applications?
We evaluated our - limited - dataset of use-cases and concluded that Identity Management, Unique Asset Management, Unique Asset Transfer, Code Execution on the Blockchain, and Storing verifiable Data Off-Chain: Hash Anchoring are the most commonly used features. We provide the full statistical evaluation in figure A.2 (appendix).

b) Are there features used in nearly every application and rare features that are only used in a limited number of applications?
We singled out Storing Data On-Chain and its Usage, Interoperability, Governance, and Cryptocurrency Minting to be used less frequently than the other functional features.

c) Can the relevance of the features be derived from the use-case studies?
We can only derive a tendency from the use-cases regarding the feature catalog. Additionally, we limited this work to DLT-based Business Problem Solvers as defined in section 2.2. This limitation gives better results for the relevance of the features regarding the defined category. It especially can not derive a relevance over the whole DLT spectrum, specifically including permissionless Blockchains defined as Cryptocurrency in section 2.2.

**Q5 - What are characteristics for proper DLT applications?**

a) Why is DLT needed for solving the specific issues?
Aggregating all use-case studies we receive the following list of reasons why DLT is needed for solving specific issues:
   * Elimination of third parties for profit
   * The Blockchain is uncensorable
   * Sharing data and their business benefits across company borders
   * Many actors with different needs on a single platform can securely interact with each other
   * Privacy of the data and transparency of the correctness
   * Pseudo-anonymity and digital identities
   * Tamper-proof and immutable data storage

b) Could DLT Projects also be realized without DLT and still solve the same issues?
The use-case of CashOnLedger (5.2) could be solved without DLT. However, DLT allows for a higher degree of automation, and with the requirements of CashOnLedger, it is the best technology choice. All other interviewees claimed that their use-case would not be possible without DLT.

c) Would it be possible that projects are easier to implement and execute without DLT while still solving the same issues?
In our - again - limited dataset, we have not found a single project that would be easier to implement without DLT, rather, one that utilizes DLT to solve a non-DLT problem more efficiently.

# 7. Future Work and Conclusion

## 7.1. Future Work

In this work, we presented a feature catalog and use-case studies.

Future work's primary goal is to expand the existing catalog on both the functional and non-functional side of it. The first step would be to expand the feature set to find new functional features. The second step would be to enrich each existing feature with a more detailed description, uses, and real-world examples to finally transform the feature catalog into a pattern catalog.

GAMMA [14] gives a framework for describing patterns which we propose to use in this pattern catalog with slight modifications.

| Gamma et al. | Proposed |
|---|---|
| Pattern Name | Pattern Name |
| Intent | Intent |
| Also Known As | - |
| Motivation | Motivation |
| Applicability | Applicability |
| Structure | Structure |
| Participants | Participants |
| Collaborations | Collaborations |
| Consequences | Forces |
| Implementation | Implementation |
| Sample Code | Sample Code |
| Known Uses | Known Uses |
| Related Patterns | - |
| - | DLT Spectrum Position |
| - | Relevance |

Table 7.1.: Proposed Pattern Structure [14]

Additionally, a radar chart should be created for each pattern to describe the position of the pattern in the DLT spectrum.

Furthermore, a map where the patterns can be positioned should be created after the pattern catalog has been finished. In this map, the different patterns can be clustered and organized even better.

Also, a lot more use-case studies have to be conducted to receive more data. We also

propose to include the categories *Cryptocurrencies* and *Ecosystem Software* as defined in section 2.2 in the use-case studies to gain more insight on the full field of DLT instead of only on the enterprise level. Because at the time of writing, the Cryptocurrencies and Ecosystem Software categories are the far more relevant ones when compared by investment money.

Lastly, the use-cases should ultimately be judged according to reasonableness by DLT experts instead of just presenting them to the audience. We did not feel comfortable making a judgment for the use-cases at the time of writing.

## 7.2. Conclusion

First we conclude, that it is possible to partition DLT by features instead of just into the categories *public*, *private*, *permissionless* and *permissioned*. Furthermore, we have shown that there are no clear boundaries of the features and specific implementations between the before mentioned categories, but instead, it is possible to mix and match common permissioned and permissionless features to create new platforms.

Furthermore, we were only able to identity a small quantity of related work regarding the generalization of technology into functional and non-functional features and their refining into generally applicable patterns.

Therefore, we conclude that a lot more work has to be done on scientifically processing and categorizing use-cases in the DLT space. For a real comparison of the reasonableness of use-cases and their categorization and break down in features, a few dozens if not a hundred case studies are needed for the data not to be biased.

Additionally, we discovered, that the non-functional features in DLT are not just features that can be improved on each new iteration but that they are instead forces working against each other that have to be balanced for the specific use-case to receive the best product.

We see the feature catalog as a first step to partitioning the DLT space and have set a generalization level at which features should be turned into design patterns. However, we want to stress that the evaluation given in this work lacks generalization. With eight other use-cases, we might have come to entirely different results in the evaluation regarding the usage of the feature catalog, and therefore the evaluation about the quantity of the appearance of the features should be taken with a grain of salt.

# A. Appendix

## A.1. Blockchain Applications


Figure A.1.: Mindmap abstraction of the different types of blockchain applications. [8]

## A.2. Blockchain Attributes

| Attributes | Prerequisites & determinants | Architecture Blockchain | | |
|---|---|---|---|---|
| | | Permissionless | Permissioned | Database |
| Trust | Lack of Trusted Third Parties | High | High | Low |
| | Accountability | High | High | High |
| | Immutability | High | High | Medium |
| | Multiple non-trusting writers | High | High | Low |
| | Peer-to-peer transactions | High | High | Low |
| | Traceability of transactions | High | High | Low |
| | Verifiability of transactions | High | High | Low |
| Context | Data/transaction notarization | High | High | Low |
| | Data transparency | High | High | Low |
| | Security | High | High | Low |
| | Privacy | High | Medium | Low |
| Performance | Latency and transaction speed | Low | Medium | High |
| | Maintenance costs | High | High | Low |
| | Redundancy | High | High | Medium |
| | Scalability | Low | Medium | High |
| Consensus | Rules of engagement | High | High | Low |
| | Need for verifiers | High | High | Low |
| | Autonomous/dynamic interactions between transactions of different writers | High | High | Low |

Table A.1.: Analysis of attributes and prerequisites of blockchain versus traditional databases. [8]

## A.3. Features of the Catalog used by the projects

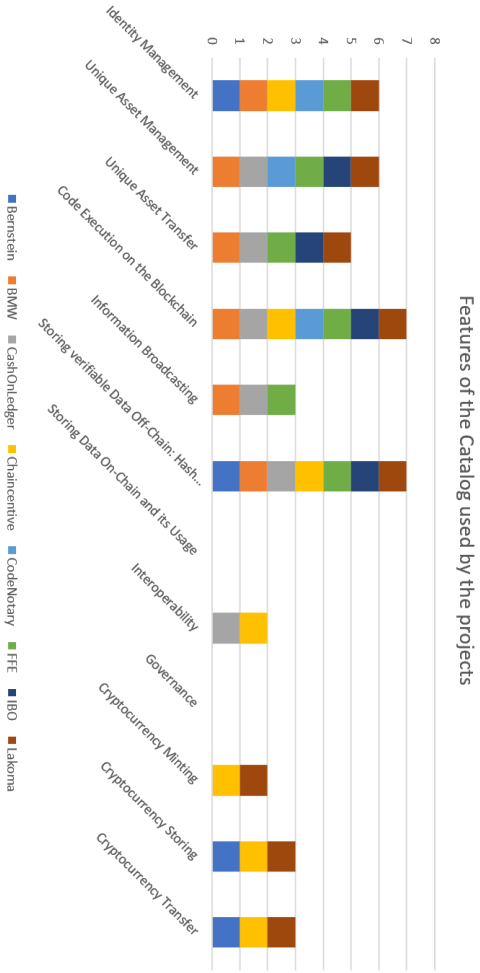



Figure A.2.: Features of the Catalog used by the projects

## A.4. Interview Questionnaire

1. Meta Information
    - Who are you?
    - What company are you working for?
    - Is it a small, medium or big company?
    - What is you DLT background?
    - How long have you been in DLT?
2. Overview of the Project
    - Which problem are you solving?
    - Could you summarize your project in one sentence?
    - Is your project open source? // Why not?
    - Why could this problem not be solved without DLT?
    - Could the problem be solved without DLT?
    - What are your customers saying?
3. Technology Choices
    - Which platform are you using // Did you create a new Platform?
    - Did you use existing frameworks/patterns designing your product?
    - Could you explain the decisions that led to choosing the platform
    - What core features of the platform do you use?
    - Did your decision process include deciding between permissioned and permissionless ledgers?
    - What would happen on a platform change?
4. Questions to the Catalog
    - From your experience do you see any missing categories or technology?
    - Is there a category or technology that you would remove?
    - Would such a catalog be beneficial for your work?
5. Development Process
    - Could you outline your general development process?
    - Could you describe the tools you are using to develop the Product?
    - Could you describe your deployment process?

# List of Figures

# List of Tables

# Bibliography


[1] H. Adams, N. Zinsmeister, and D. Robinson. "Uniswap v2 core." In: *URl: https://uniswap.org/whitepaper. pdf* (2020).

[2] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." In: *Proceedings of the thirteenth EuroSys conference*. 2018, pp. 1–15.

[3] A. M. Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014.

[4] A. M. Antonopoulos and G. Wood. *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018.

[5] A. Bogensperger and A. Zeiselmair. "Updating renewable energy certificate markets via integration of smart meter data, improved time resolution and spatial optimization." In: *2020 17th International Conference on the European Energy Market (EEM)*. IEEE. 2020, pp. 1–5.

[6] V. Buterin et al. "Ethereum white paper." In: *GitHub repository* 1 (2013), pp. 22–23.

[7] V. Buterin. *Privacy on the Blockchain*. 2016. URL: https://blog.ethereum.org/2016/01/15/privacy-on-the-blockchain/ (visited on 01/15/2016).

[8] F. Casino, T. K. Dasaklis, and C. Patsakis. "A systematic literature review of blockchain-based applications: current status, classification and open issues." In: *Telematics and Informatics* 36 (2019), pp. 55–81.

[9] Coinmarketcap. *All Cryptocurrencies*. 2020. URL: https://coinmarketcap.com/all/views/all/ (visited on 10/04/2020).

[10] Coinmarketcap. *Today's Cryptocurrency Prices by Market Cap*. 2020. URL: https://coinmarketcap.com/ (visited on 10/06/2020).

[11] J. Elliott. *IBM Mainframes–45+ Years of Evolution*. 2010.

[12] Etherscan. *Top Accounts by ETH Balance*. 2020. URL: https://etherscan.io/accounts/c/2 (visited on 10/04/2020).

[13] T. L. Foundation. *Hyperledger*. 2020. URL: https://www.hyperledger.org/ (visited on 10/29/2020).

[14] E. Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[15] M. Herlihy. "Atomic cross-chain swaps." In: *Proceedings of the 2018 ACM symposium on principles of distributed computing*. 2018, pp. 245–254.

[16] Hyperledger. *A Blockchain Platform for the Enterprise*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/latest/.

[17] Hyperledger. *A Blockchain Platform for the Enterprise*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html.

[18] Hyperledger. *A Blockchain Platform for the Enterprise*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/latest/ledger/ledger.html#ledgers-facts-and-states.

[19] Hyperledger. *Hyperledger - Advancing business blockchain adoption through global open source collaboration*. 2020. URL: https://www.hyperledger.org/ (visited on 10/04/2020).

[20] Hyperledger. *Hyperledger Fabric 1.0 is Released!* 2017. URL: https://www.hyperledger.org/blog/2017/07/11/hyperledger-fabric-1-0-is-released (visited on 11/07/2017).

[21] Hyperledger. *Listening to events with Fabric Network*. 2020. URL: https://hyperledger.github.io/fabric-sdk-node/release-1.4/tutorial-listening-to-events.html.

[22] S. King and S. Nadal. "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake." In: *self-published paper, August* 19 (2012), p. 1.

[23] Y. Liu, D. He, M. S. Obaidat, N. Kumar, M. K. Khan, K.-K. R. Choo, et al. "Blockchain-based identity management systems: A review." In: *Journal of Network and Computer Applications* (2020), p. 102731.

[24] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. http://www.bitcoin.org/bitcoin.pdf. 2009.

[25] G. W. Peters and E. Panayi. "Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money." In: *Banking beyond banks and money*. Springer, 2016, pp. 239–278.

[26] J. Poon and T. Dryja. *The bitcoin lightning network: Scalable off-chain instant payments*. 2016.

[27] D. Pulse. *Aave*. 2020. URL: https://defipulse.com/aave (visited on 09/12/2020).

[28] D. Pulse. *Curve Finance*. 2020. URL: https://defipulse.com/curve-finance (visited on 09/12/2020).

[29] D. Pulse. *Flexa*. 2020. URL: https://defipulse.com/flexa (visited on 09/12/2020).

[30] D. Pulse. *Synthetix*. 2020. URL: https://defipulse.com/synthetix (visited on 09/12/2020).

[31] D. Pulse. *yearn.finance*. 2020. URL: https://defipulse.com/yearn.finance (visited on 09/12/2020).

[32] J. B. Rothnie Jr, P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, and E. Wong. *Introduction to a system for distributed databases (SDD-1)*. 1980.

[33] P. Ruan, D. Loghin, Q.-T. Ta, M. Zhang, G. Chen, and B. C. Ooi. "A Transactional Perspective on Execute-order-validate Blockchains." In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 543–557.

[34] N. Szabo. *Smart Contracts*. 1994. URL: https://web.archive.org/web/20140413000357/http://szabo.best.vwh.net/smart.contracts.html (visited on 11/18/2020).

[35] F. Tao, Q. Qi, A. Liu, and A. Kusiak. "Data-driven smart manufacturing." In: *Journal of Manufacturing Systems* 48 (2018), pp. 157–169.

[36] J. E. William Entriken Dieter Shirley. *EIP-721*. 2020. URL: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md (visited on 11/18/2020).

[37] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica. "{DIZK}: A distributed zero knowledge proof system." In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 675–692.

[38] X. Xu, C. Pautasso, L. Zhu, Q. Lu, and I. Weber. "A pattern collection for blockchain-based applications." In: *Proceedings of the 23rd European Conference on Pattern Languages of Programs*. 2018, pp. 1–20.

[39] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang. "An overview of blockchain technology: Architecture, consensus, and future trends." In: *2017 IEEE international congress on big data (BigData congress)*. IEEE. 2017, pp. 557–564.