TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

Kristjan Ulst 152899IABM

CREATING A BLOCKCHAIN-BASED SOLUTION FOR THE MUSIC INDUSTRY TO MANAGE COPYRIGHT ROYALTIES

Master's Thesis

Supervisor: Mart Roost

MSc

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Kristjan Ulst 152899IABM

PLOKIAHELATEHNOLOOGIAL PÕHINEVA LAHENDUSE LOOMINE AUTORITASUDE HALDAMISEKS MUUSIKATÖÖSTUSES

Magistritöö

Juhendaja: Mart Roost
MSc

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Kristjan Ulst

03.01.2019

Abstract

Music consumption has changed immensely with the digital age. Completely new music services and listening opportunities have arisen that did not exist before. Music industry lies on royalties' payments, which are due when music is consumed. To this date royalties' distribution systems within the music industry are widely paper-derived, which in essence is a non-transparent, inefficient and outdated method to collect and distribute royalty payments. This results in undistributed royalties – estimated from 3.5 to 8.6 billion dollars yearly – and payment times ranging from months to a year. In the present work, the author investigates the possibility of implementing blockchain technology to distribute royalties. A novel royalty distribution system is created and Ethereum smart contracts are chosen based on the requirements of the system and for their properties. The created solution records transactions in a distributed manner with the information is accessible to all parties in real-time. The royalty distribution process is completed in minutes instead of months as is in the current system. The system described in the thesis was implemented in the start-up company called Fanvestory and used for distributing royalties for the local artist Tommy Cash.

This thesis is written in English and is 80 pages long, including 5 chapters, 23 figures, and 6 tables.

Annotatsioon

Muusika tarbimine on digitaalajastul tohutult muutunud. On tekkinud täiesti uusi muusikateenuseid ja kuulamisvõimalusi, mida varem ei eksisteerinud. Muusikatööstuses teenitakse tulu autoritasude kaudu, mis kuuluvad väljamaksmisele kui kuulatakse või mängitakse autori lugusid. Kuni tänase päevani on muusikatööstuses autoritasude kogumine ja jaotamine laialt levinud paberkandjal. See aga on oma olemuselt läbipaistmatu, ebaefektiivne ja aegunud litsentsitasude kogumise ja jaotamise meetod. Selle tulemusena jääb hinnanguliselt jaotamata 3,5–8,6 miljardit dollarit autoritasusid aastas ning maksetähtajad ulatuvad ühest kuust kuni aastani. Käesolevas töös uurib autor võimalust kasutada litsentsitasude jaotamiseks plokiahela tehnoloogiat. Töö tulemusena loodi uudne autoritasude jaotamise lahendus, mis põhineb Ethereumi tarkadel lepingutel, mille omadused võimaldavad süsteemil toimida. Loodud lahendus salvestab tehingud jagatult nii, et informatsioon on kättesaadav kõigile osapooltele reaalajas. Autoritasude jaotamise aeg uue süsteemiga on minutite pikkune, kui varem pidi ootama kuid. Töös kirjeldatud lahendust rakendati Fanvestory start-upis ning kasutati Tommy Cashi autoritasude jaotamiseks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 80 leheküljel, 5 peatükki, 23 joonist, 6 tabelit.

List of abbreviations and terms

CMO	Collective Management Organization
ISRC	International Standard Recording Code
ISWC	International Standard Work Code
IDE	An integrated development environment
VM	A virtual machine
IP	Intellectual Property
ID	Unique Identifier
PHP	Hypertext Preprocessor
JSON	JavaScript Object Notation
JS	JavaScript
ABI	Application Binary Interface
EVM	Ethereum Virtual Machine

Table of Contents

List of Figures

List of Tables

1 Introduction

This chapter presents a foundation for understanding this thesis. The background of the problem is explained and the problem statements are proposed. Lastly, the research process and the structure of the thesis are being introduced.

1.1 Background

The way that people consume music has changed immensely with the digital age, also multiple different music services have emerged that did not exist before. In the digital age, it is not feasible to license music and compensate rights holders, including composers, recording artists, record labels and publishers, **based on paper-derived processes**. However, it is still a wide-spread practice in the music industry at the moment. The processes of royalty management and rights development have been slow when comparing to the developments in music access models and the worldwide consumption scale within the past years.

It has been fundamentally **problematic to gain access to authoritative sources of data** about music copyright, which describes the owner's rights and license terms. This data is critically important for music service providers and rights administrators to **process royalty transactions** for the vast amount of music uses through their services. The absence of easily accessible and correct data causes inefficiencies, inaccuracies, ambiguities and legal risks, which result in large amounts of unclaimed royalties, erroneous transaction, and lawsuits [1].

In music licensing, cryptography and blockchain technology has emerged as a way to deliver secure, transparent and reliable distributed transaction processing. The technology has a database of transactions that are consensually shared and synchronized across the network. One of the main issues that the blockchain technology can address in this context is the digital registry of artwork, including the certificates of authenticity, condition, and ownership.

1.2 Problem Statement

The aim of this thesis was to investigate the possibilities of solving the music industry's inefficiencies and lack of transparency in royalty distribution through cryptography and blockchain technology.

The central research question which this thesis will investigate is:

- How can blockchain technology solve the music industry **transparency and efficiency problems** regarding royalty payouts in today's digital era?

1.3 Research Process

To reach the aim of the study, the music industry's royalty system will be investigated thoroughly, cryptography and blockchain technology will be studied in detail, and various possibilities will be mapped. An implementation of blockchain technology for royalty distribution system is going to be carried out, based on the needs of a start-up company called Fanvestory.

The following steps need to be taken to answer the posed central research question:

- Explain the music industry and point out the **most problematic challenges** the industry is facing;
- Find out the **possibilities** behind cryptography and blockchain technology;
- **Implement** the technology based on the needs of the startup;
- Analyze the results and describe the benefits and challenges with the solution that was presented;
    - Distributed data ownership;
    - Transparent and understandable intellectual property contracts, available to access by all related parties;
    - Improved speed of royalty distribution process.

These steps also roughly correspond to the outline of the thesis, which will be introduced in the next chapter.

1.4 Outline of the Thesis

The thesis is structured into five chapters. The first, chapter provides background and introduction to the research conducted, including the research process. The second chapter explains the theory behind the thesis. Firstly, an overview of the music industry and how does it work technically and point out topical challenges in the industry. Secondly, cryptography and blockchain technology is being presented by pointing out what are the possibilities around this technology. The third chapter explains the implementation of the research. The fourth chapter introduces the results. The final chapter summarizes the thesis by revisiting all of the most important remarks from each of the thesis chapters and proposing directions for future work.

2 Background theory

This chapter gives a detailed overview of the music industry, as well the challenges that are relevant in today’s digital world. Finally, technologies as cryptography and blockchain are being presented, together with possibilities, different use cases, and threats.

2.1 Music industry

The primary form of payments for musicians is royalties, which are generated by the licensing of copyrighted songs and recordings. Both, licensing system and the intellectual property law have gone through a number of adjustments over the past years due to the digitalization of music, but most of the music industry legal framework has been unchanged [2].

2.1.1 Music rights

Copyright is legal right that protects the use of creators work once the idea has been physically expressed. Copyright law lays out a framework of rules around how that work can be used. It sets out the rights of the owner, as well as the responsibilities of others who want to use the work. Copyright protects various forms of creative expression that are **fixed in a tangible form** (17 U.S. C. §102) [3].

There are two types of creative works that are most frequently involved: **musical composition and sound recordings** [4] – see Figure 2.1. It is important to understand that the two main copyrights in a song are separate and involve different rights and sometimes different owners.

- The copyright in songs - when a lyricist, composer, or songwriter creates a musical work, consisting of the melody and or lyrics, this is referred to as the “author rights”, **musical composition** or under civil law systems and generally referred to as "publishing rights".

- The copyright in sound recordings - when this composition is performed, recorded, mixed, and mastered, the resulting creation is referred to as the “master”, **sound recording** or under many civil law systems known as "neighboring rights".

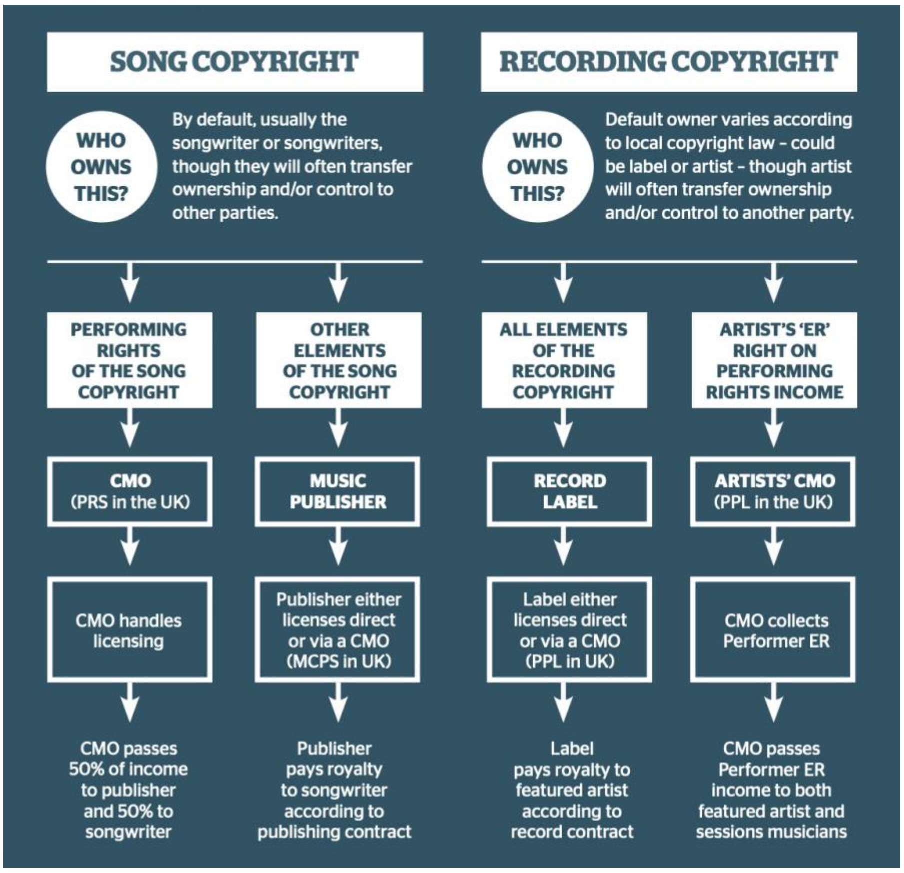
SONG COPYRIGHT
RECORDING COPYRIGHT
WHO OWNS THIS?
By default, usually the songwriter or songwriters, though they will often transfer ownership and/or control to other parties.
WHO OWNS THIS?
Default owner varies according to local copyright law - could be label or artist - though artist will often transfer ownership and/or control to another party.
PERFORMING RIGHTS OF THE SONG COPYRIGHT
OTHER ELEMENTS OF THE SONG COPYRIGHT
ALL ELEMENTS OF THE RECORDING COPYRIGHT
ARTIST'S 'ER' RIGHT ON PERFORMING RIGHTS INCOME
CMO (PRS in the UK)
MUSIC PUBLISHER
RECORD LABEL
ARTISTS' CMO (PPL in the UK)
CMO handles licensing
Publisher either licenses direct or via a CMO (MCPS in UK)
Label either licenses direct or via a CMO (PPL in UK)
CMO collects Performer ER
CMO passes 50% of income to publisher and 50% to songwriter
Publisher pays royalty to songwriter according to publishing contract
Label pays royalty to featured artist according to record contract
CMO passes Performer ER income to both featured artist and sessions musicians

Figure 2.1 Recording and publishing rights [5].

2.1.2 Ownership & Royalties

The default owners of lyrics and **musical compositions** are the lyricist and the composer, in other words ‘creator’ or ‘author’. There are also works that are co-written, therefore it’s for the creators to decide on how the copyright is split between each contributor.

With **sound recordings**, default ownership rules may vary, depending on the local copyright system. In some countries, the individual or company that funds (pays for) a recording, rather than the performers who appear on it, will be the default owner of the resulting copyright. These funders, usually record labels, are often referred to in copyright law as the ‘producer’ [5]. If there are multiple writers, publishers, artists, or record labels involved, each of those stakeholders can be further divided.

The typical stakeholders of composition and sound recording can be seen in Figure 2.2.

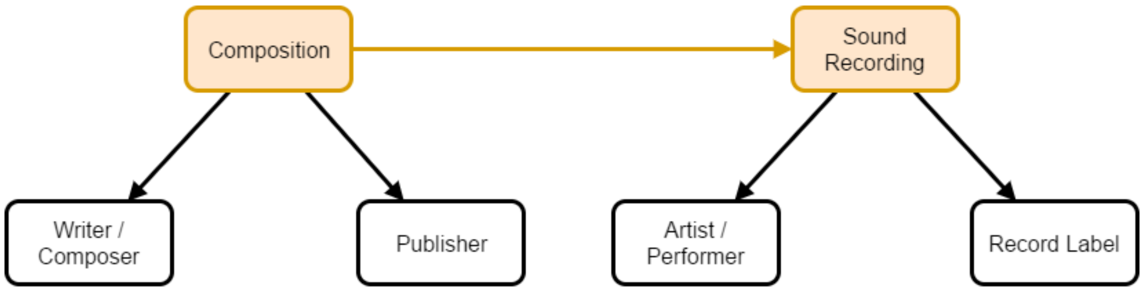

Figure 2.2 Typical stakeholders of composition and sound recording [6].

Royalties are the sums paid to rights-holders when their creations are sold, distributed, embedded in other media or monetized in any other way [7].

Music royalties are generated for various types of licensing and usage. The types of music royalties include **mechanical, public performance and synchronization** [8].

- **Mechanical royalties** are generated for the physical or digital reproduction and distribution. This applies to music formats such as CD, cassette, vinyl, digital downloads, and streaming services. For example, a record label pays a mechanical royalty to a songwriter or artist every time they press a CD of their music.

- **Public performance royalties** are generated for copyrighted works performed, recorded, played or streamed in public. This includes radio, tv, live concerts, restaurants, clubs, music streaming services and anywhere else the music plays in public.

- **Synchronization royalties** (sync) are generated for copyrighted music used in visual media. Sync licenses allow the right to use copyrighted music in films, tv, commercials, video games, online streaming, advertisements, and any other type of visual media. It is an agreement between the master recording owner such as a record label and the person or company seeking permission to use the recording.

2.2 Music industry challenges

The advent of the internet and the expansion of digital technologies have together created profound **disruptions in the music industry**. Artists of all sizes and ambitions are today more empowered and equipped to reach wider audiences, and music listeners have access to more songs on more devices than ever before. Unfortunately, the music industry is complex, and the digital transformation has made it even more opaque. Some of the key stakeholders have been hesitant to accept these new innovations, fighting to preserve an incredibly profitable, pre-digital industry, and as a result, the lack of transparency and common standards for data reporting, despite their availability. Along with it, there has been only a *limited implementation* of digital technologies in the industry, creating an enormously mismatched ecosystem in which there are superficial, profit-generating elements without the proper digital infrastructure to support it. In the modern music business, millions of micro-transactions take place daily, generating revenues from songs and albums – new technologies have the potential of making this **process transparent** to all participants [9].

The digital age has created challenges but more importantly opportunities for the music industry. Over the previous decade, the music rights sector has undergone a busy evolution in *licensing models and industry standards* are now starting to emerge. There are differences in music rights models around the world due to variations in copyright law which have been evolved in each market separately. The variation is challenging in a digital sector where many services aspire to be truly global [5].

2.2.1 Lack of transparency

The lack of transparency has been proven to exist in the value chain in recorded music. Due to this, often large funds are **paid to the wrong party**. Also, a large amount of royalty revenue is trapped in a so-called "black box" where the rightful owner of that revenue cannot be determined due to poor transparency of the industry-wide system, which would connect usage to ownership [9], [5], [10], [11].

While most of the streaming deals' specific details are hidden behind **non-disclosure agreements**, it is virtually *impossible* for artists, songwriters or managers to check if labels, publishers or collective management organizations *process royalty payments correctly and efficiently* [5].

Music creators' works are nowadays available in the digital market on multiple platforms in different models. In principle, it could be possible to have and give **real-time information** about royalties to the artist or managers. Instead, they usually receive this information on paper. Due to this, vast funds are paid to the wrong party. The music industry has attempted implementing unique identifiers like ISRC for sound recordings and ISWC for music works but has **failed to properly link them** for music releases. Furthermore, the rights owners often pursue their own standard for data reporting from digital services, which means there is still an absence of a common standard for music data reporting, resulting in services having to report in **multiple formats and losing efficiency** [9].

The modern music industry, like few other industries, involves millions of small-scale transactions a day while generating revenues from songs and albums being played. Stakeholders of the music will earn fractions of pennies on one song or album, which is spread across **millions of transactions**. It should be possible to achieve a high level of transparency with new technologies. Instead, it is seen, that music industry has applied less transparent frameworks, technology, and processes while evolving from the era of music-as-a-product into music-as-a-service. Such uneven application of technology causes friction, opacity, and frustration [9].

Doubts and mistrust are generated with the current distributions systems. More transparency between authors, management agencies and distributors has been demanded by the authors for a long period of time. With increased transparency trust and improved terms between associates would ultimately make the distribution fairer, which is the goal.

European commission had interest in **proposing legislation** about collective management of *copyright and related rights* because [11]:

1. Some collective management organizations have *raised concerns* since their transparency in collecting and handling of revenues on behalf of rightsholders have been **suspicious and nontransparent**.

2. To date many collective management organizations are unable for online licensing, they are lacking the capability to process data from service providers on streaming or downloading music. Also, they are *unable to match this data* with their repertoire. This can easily lead to **faulty invoicing**. One of the requirements

of today’s legislative proposal is that collective management organizations need to set up *proper databases* for tracking their own repertoire.

Transparency towards rightsholders as well as users would have to improve also, for example, rightsholders should have the possibility to reach information about collective management repertoire, that they represent.

2.2.2 Efficiency issues in royalty distribution

The efficiency of collaboration between collective management organizations (CMOs) can suffer from the **lack of quality standards** and services for information exchange that can result in *excessive costs* in manual labor and various system integrations needed [12]. Usually, CMOs receive money throughout the year from overseas societies, which are paid through their distributions. There are societies that **only distribute once or twice a year**, which withholds receivable royalties long after the play or performance date [13], [14].

A global database would make the licensing process *more efficient with faster and cheaper ways* to obtain a license since intermediaries’ operational and administrative costs between licensee and owner would be reduced to the minimum [9].

2.2.3 Lack of information

In the music industry today, there is *no global registry* of musical recordings with information on their creators and owners. At the moment a number of databases exist, but the information in them may vary [15].

Large amounts of revenues are in a so-called **“black box”,** in where it is *impossible to identify revenue rightful owner* since there is no worldwide system connecting usage to ownership. In many cases, mechanical royalty payments, CMO payments, and other royalties do not reach owners for that kind of reasons and there are few financial incentives for money holders to find the rightful owners.

Unattributable payments because of *poor or incorrect licensing information* and poor knowledge also end up in an obscure “black box”. CMOs and services, that are unable to distinguish a rightful royalty recipient via an ISRC or ISWC or other identifier systems will place this money into escrow accounts and eventually distribute it to labels and publishers based on market share. Such revenue is normally not shared with the artist or composers signed to those publishers or labels, since they *cannot be rightfully identified* [9]. It is estimated, that a percentage of **20-50%** music payments **do not make it to their rightful owners** [9], [16]. According to the IFPI report, global recorded music industry revenue in 2017 was US$ 17.3 billion [17]. In other words, **there is a problem worth up to 8.6 billion that needs solving**.

Blockchain technology can be used to create a single and distributed database of musical creators and their works and this database could provide correct contract information about music owners and as well as terms of use [15].

2.3 Cryptography

Cryptography is the method for **encrypting and decrypting information** through complex mathematics. It means, that information can be understood only by the intended recipient and nobody else. First unencrypted data, such as a piece of text, is selected and encrypted using a mathematical algorithm known as a *cipher*. It will create a ciphertext, a piece of information that is useless and nonsensical until it is decrypted. This method of encryption is known as *symmetric key cryptography* [17].

For an example, in the Roman times, Julius Caesar used the Caesar cipher to protect Roman military secrets. Only Caesar's generals knew that to decode the letters they would have to shift each letter to the right by three, whilst information remained safe if intercepted by Caesar's enemies. Modern cryptography working principle is the same, only with a far greater level of complexity [18].

In Caesar cipher each letter in a text was substituted by a letter 3 spaces to the left in the alphabet, only with this knowledge it was possible to decrypt Caesars' messages – see Figure 2.3.

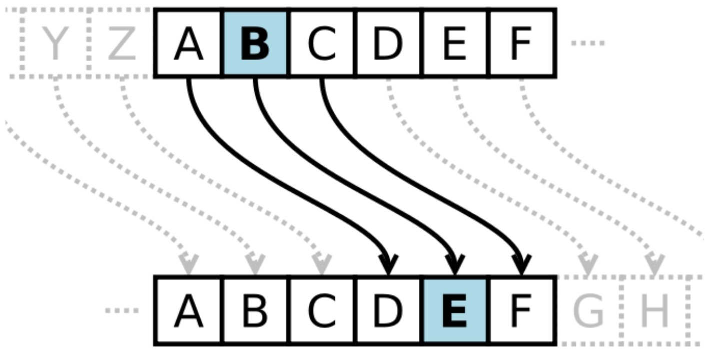

Figure 2.3 Caesar cipher's method of encoding messages [19].

2.3.1 Symmetric and public-key cryptography

Blockchain technology uses cryptography for protecting the identities of users, also it ensures, that all transactions are **done safely and all information is secured and stored**. In that sense, anyone that uses the blockchain technology can be completely confident that once something is recorded on a blockchain, it is done legitimately and securely.

The type of cryptography used in a blockchain, mostly *public-key cryptography*, is considered to be better suited for functions associated with technology than symmetric key cryptography [20], which is illustrated in Figure 2.4.

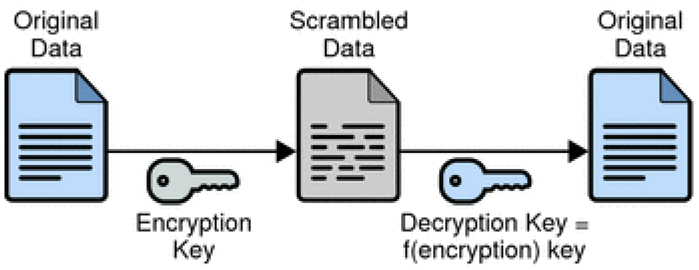

Figure 2.4 Symmetric-key encryption [21].

Asymmetric cryptography, also known as *public-key cryptography*, represents an improvement on standard symmetric key cryptography. As the public key can be shared with anyone, it also allows information to be shared with anyone if they have a *private key*, illustrated in Figure 2.5. Instead of using a single key to encrypt and decrypt information as it is the case with symmetric key cryptography, separate keys, public and private key are used [22].

A combination of a public and private key encrypts the data, while the recipients private key and the senders public key will decrypt it. It is not possible to figure out the private key based on the public key. Therefore, the sender can send the public key to absolutely anyone and nobody would gain access to the information since nobody can have its private key. The encrypted information can only be decrypted by the indented party, that also receives the private key.

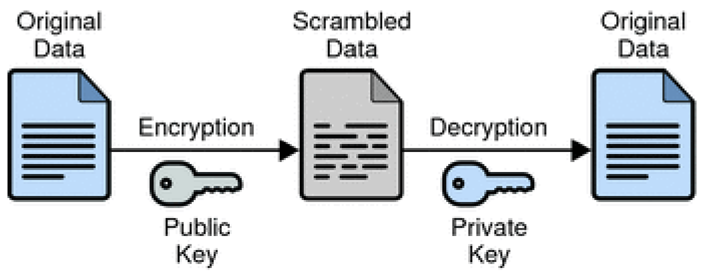

Figure 2.5 Public-key encryption [21].

Through public-key cryptography, a digital signature is produced and it secures the integrity of the data. It will be done by combining a user's private key with the data they wish to sign using a mathematical algorithm.

Since the data is part of the digital signature, the network will not accept it as valid information when any part of it is tampered with. Even editing a minor part of the data will result in reshaping the whole information and making it false and obsolete. Based on this logic, blockchain technology will guarantee that any data being recorded onto it is true, accurate and untampered with. Digital signatures give the data recorded onto blockchain its immutability.

2.3.2 Digital signature

A digital signature is the **fraction of an electronic document** that is used to *identify the person* who is sending the information. Therefore, it is possible to ascertain the nondistortion status of data in a document once signed and validate if the signature belongs to the key certificate owner [23].

As shown in Figure 2.6, the digital signature does what the name suggests, it provides *validation and authentication* in digital form.

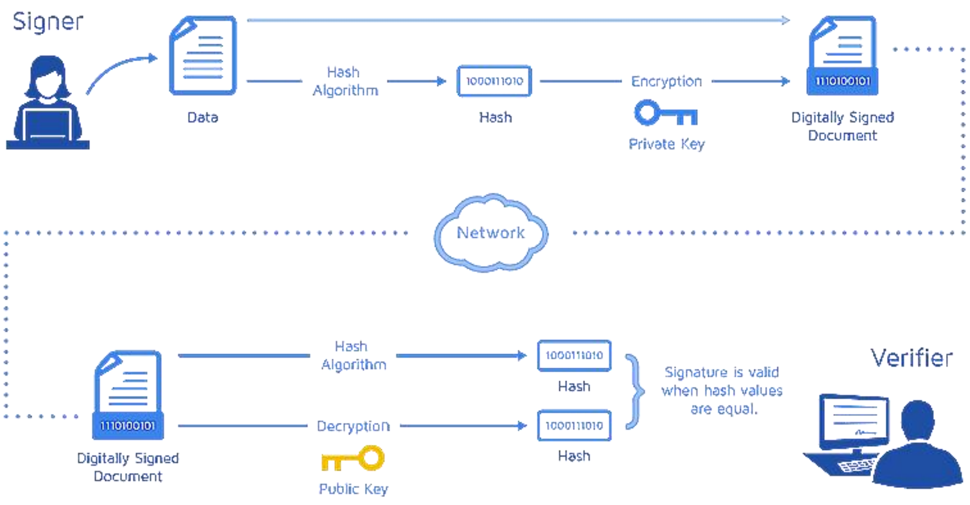

Figure 2.6 Digital signing and verification process diagram [24].

The digital signature is one of the core elements of ensuring the **security and integrity of the information** that is written to the blockchain network.

It is a standard component of many blockchain protocols, mainly used to secure transactions and blocks of transactions, manage contracts, transfer sensitive data, distribute software and any other situations where detecting and preventing tampering is necessary. The data can be *shared with anyone through a public key*, meaning digital signatures utilize **asymmetric cryptography**. Digital signatures not only protect the information but also identify the person who is sending it. The owner of a digital signature is constantly bound to a certain user and the other participants are able to know with who they are communicating with.

Multisignature, known as *multisig*, is a digital signature scheme with the condition of more than one signee to approve a transaction. A joint signature is more compact than a collection of individual digital signatures. A Large number of cryptocurrencies use multi signatures such as Bitcoin and Lisk for improving security dividing the ability to make decisions between different parties.

2.4 Blockchain technology

The technology is the future according to many people. This chapter explains what exactly it is and what it could mean for the technology industry moving forward.

2.4.1 Introduction

Blockchain technology is not a company, neither is it an application, but rather a novel way of **recording data** on the Internet. The technology can be used to develop blockchain applications, such as social networks, crowdfunding, voting systems, insurance, exchanges, intellectual property, prediction markets, online shops and much more [9], [12], [14], [15], [25], [26], [27], [28], [29], [30], [31].

The first major blockchain application **Bitcoin** was presented in a whitepaper called "Bitcoin: A Peer-to-Peer Electronic Cash System", which was created by Satoshi Nakamoto in 2008 [32]. In other words, Bitcoin is a cryptocurrency and the blockchain is the technology that underpins it. A cryptocurrency indicates to a *digital coin* that is running on a blockchain. Understanding how the technology works with Bitcoin will allow the reader to see how the blockchain can be transferred to many other real-world use cases. Bitcoin is the creation of a mysterious person or group of people known as Satoshi Nakamoto and the **identity of it is still unknown** [32], [33].

Right after the white paper was released, Bitcoin was presented to the open source community in 2009. Blockchain provided the solution to digital trust because it records the information in a public space and it is not possible to remove it. It is decentralized, transparent and timestamped [34], [35], [36].

Marc Andreessen has stated the possibilities of the new technology the following: "*The practical consequence […is…] for the first time, a way for one Internet user to transfer a unique piece of digital property to another Internet user, such that the transfer is guaranteed to be safe and secure, everyone knows that the transfer has taken place, and nobody can challenge the legitimacy of the transfer. The consequences of this breakthrough are hard to overstate.*" [36].

At its core, the technology is an open, **decentralized ledger** that documents all the transactions between different parties in a permanent way without needing additional authentication. This creates a highly *efficient process* and reduces the *cost of transactions*. Shortly after many entrepreneurs started to understand the power of blockchain, it led to a surge of investment and discovery to see how the technology could impact different industries [35].

2.4.2 Ethereum smart contracts

However, both Bitcoin and Ether are cryptocurrencies, the Ethereum blockchain is much different from the Bitcoin blockchain. Bitcoin is designed completely as a digital currency. The Ethereum blockchain has a more general approach to the technology. Both of the blockchains offer *anonymous transactions*, and none of those two are regulated or controlled by a centralized party. Still, they differ outstandingly in nature as well as functions [37], [30].

The inventor of Ethereum, Vitalik Buterin, explains the major difference between Bitcoin and Ethereum: "*The key component is this idea of a Turing-complete blockchain. ... As a data structure, it works kind of the same way that Bitcoin works, except the difference in Ethereum is, it has this built-in programming language."* [38].

By definition, a Turing-complete language is a language that can perform any computation. Especially, if there is an algorithm for something, it is possible to express it. **Smart contracts**, in other words, *Ethereum scripts* which are reusable code templates written in Solidity can therefore, run any computation. Computations are executed as part of a transaction, which means each node in the network has to execute computations [25], [27].

Because computation is expensive and it is rewarded by giving ether for the nodes that produce blocks, it is an ideal way to limit computations. Therefore, Ethereum solves the problem of denial of service attacks through everlasting malicious scripts. Every time a script is executed, the client requesting the script to execute must set a *limit for spending* Ether. The Virtual machine that runs the scripts, ensures that Ether is consumed by the script as it runs. If the script runs out of Ether before it completes, it will be halted. In Ethereum the Ether assigned to a script as a limit is called as ***gas*** [26].

Once a smart contract is agreed upon different parties, it will be stored on a decentralized, encrypted ledger. The decentralization of a blockchain hosting the ledger means that it is not processed by and as a result entrusted in a central entity who might manipulate the information. Especially, the encrypted nature of blockchain means that the data in the smart contract and its history is permanently planted into blockchain and any manipulation with the data will be *seen and corrected by other nodes* on the network. Any **tampering** would need a **huge amount of computing power** and would not be financially reasonable.

2.4.3 Different solutions

Payments

As we know, the digital revolution has significantly transformed media. It has also had an effect on the finance industry. Financial institutions use computers and have been doing that for a long time. In the 1970s and 1980 they used them for databases, in 1990 they made web pages and migrated to mobile apps in the new millennium. Although, the digital revolution in finance has not yet revolutionized cross-border transactions. Western Union has not changed their business since the beginning and is still a big name. Banks continue **using complex infrastructure for making simple transactions**, for example sending money abroad. The digitalization of banking only meant that we merely sort information and data into private databases much faster [39].

Blockchain technology makes it possible for financial institutions to create direct links between each other, which allows them to avoid correspondent banking. Financial institutions could use this database to keep track of execution, clearing, and settlement of transactions without needing any central bank database or management system. In principle, banks could formalize and secure digital relationships between themselves in a new way. It is possible to conduct transactions between two parties on a frictionless P2P basis [39]. A permissioned blockchain called Ripple is built to solve many of mentioned problems [40].

E-voting

E-voting is proven to be problematic with many difficulties. There is a number of desired e-voting properties, that have trade-offs. For example, the main requirement of voting is privacy, since votes should be anonymous to prevent coercion, but e-voting needs public verifiability or there would be a possibility that votes are changed at will. In e-voting multiple parties are involved, who usually **do not trust each other**. For these reasons, many have proposed to base e-voting on blockchain technology. Looking at the requirements for e-voting, it would be reasonable to think that blockchain technology could help to achieve some of the desired properties [30].

Blockchain technology can be used to address *voter tampering* since it generates cryptographically secure voting records, which are accurate, permanent, secure and transparent [28]. This makes it impossible to manipulate votes [29]. Moreover, anonymity is preserved while being open to public inspection. Tampering with votes is nearly impossible with blockchain technology.

At the moment the Estonian financial technology company LHV Group shareholders, if they are Estonian citizens or Estonian e-residents can use **blockchain-enabled e-voting** (BEV) for making corporate governance-related decisions [41]. Estonia’s e-residency platform authenticates e- resident shareholders and they can log in using verified national online ID and can vote LHV Group annual general meeting [42].

Estonia is planning to adopt blockchain technology in a variety of areas such as an e-residency project, which would allow foreign citizens to establish a business in Estonian jurisdiction and healthcare by storing secure health data and allowing real-time monitoring of patient conditions [43], [44].

Internet of things

There have been multiple suggestions for possible use cases for blockchain technology on the Internet of Thing (IoT) in combination with smart contracts. The aim is to provide autonomous systems, that pay for the resources that are consumed and also get paid for the provided resources. As the system is decentralized and has entities that do not trust each other, using a blockchain is a logical solution [30].

Settling on industry standards for IoT has been difficult for large hardware makers. Blockchain technology can offer a secure ownerless protocol that anyone can use, examples of IoT interactions on blockchain follows:

- **Device-to-device payment** policies: The terms in a smart contract can be customized, which would make it possible for your phone to buy things it knows you need. Customization could be done for example for the amount of money it is allowed to spend without asking permission.

- Encoding value or financial contracts onto **retail objects**: It has proven to be difficult to merchandize intellectual property such as video or music without a physical good to put on a store shelf. In a retail setting, financial products and services can be sold as gift-card-like objects by printing or encoding a contract address onto the item.
- **Hardware wallets**: Hardware wallet creates itself an address and stores the private encrypted key on the hardware. Hardware wallets are revolutionary in wealth management since they allow to keep the possessions in a media safe, in an arbitrarily large number of crypto assets [26].

2.4.4 Threats

Privacy

Blockchains that are publicly available, will not guarantee any data privacy. Anyone is able to join a public blockchain network, the data and transactions are seen for everyone [34], [45], [46]. Privacy definition in blockchain is maintained by breaking the flow of data. Therefore, regarding this security model, **numerous studies** have proposed *privacy concerns* and countermeasures to *increase anonymity* in the blockchain. Meiklejohn presented a framework of anonymity focusing on the holder of the coin [47].

Koshy researched the *traffic pattern* in Bitcoin and discovered that some subset of Bitcoin addresses *can be outlined to an IP address* by observing the transaction relay traffic [48]. A framework for traversing the Bitcoin network and creating statistics based on that was introduced by Feld. This tool allowed Feld to figure out that an average peer-list consist of addresses that mainly reside in the own autonomous systems of the peers. Therefore, the author concluded that *transaction linking might be possible* [49].

Security

The blockchain is a tamper-proof, shared ledger where transactions are made irreversible and non-repudiable because of one-way cryptographic hash functions. Immutability excludes the need for reconciliations because it gives a unique reconciled version of the truth - the *transaction history among peers* [46]. An immutable historic record, which is validated by community consensus creates trust in the system. It becomes extremely difficult for an individual or a group to tamper with the record unless they **control the majority of the miners** [31]. The system is designed with the purpose that honest nodes control the network [34]. In Figure 2.7 it is shown that, if attacker nodes jointly control more computational power than the others, the network is vulnerable to the **51% Attack**. Although the Bitcoin network is built as a decentralized network, centralization of mining power by some large mining pools increase the risk of the attack.

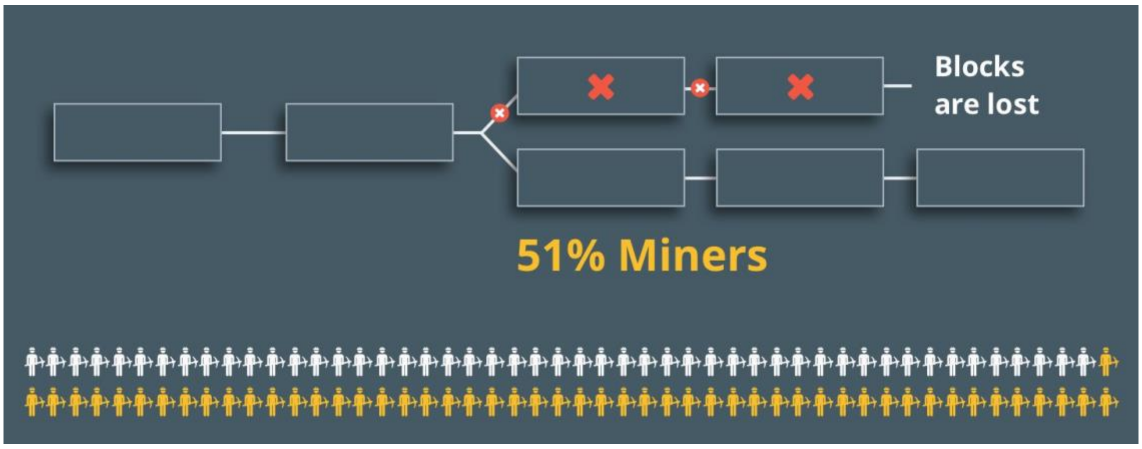

Figure 2.7 51% Attack theory [50].

A study by Beikverdi et al., demonstrates [51] that the **centralization factor of Bitcoin has been constantly growing** from 2011 (0.26) to 2014 (0.33). 0 stands for purely decentralized and 1 represents fully centralized. Additionally, there are researches claiming that the 0.5 assumption of computational power is not enough for safety.

Power cost

Global warming has led society to focus on environmentally friendly practices. The electricity, that is needed to run large-scale blockchain networks is colossal. Moreover, the Bitcoin network today **uses more electricity than several countries of the world**. With the blockchain adoption, the energy burden is becoming even more considerable. If more energy efficient solutions cannot be found for running applications based on blockchain technology, then there might be considerable opposition as far as it is large-scale implementation is concerned [54].

Bitcoin's power usage for transactions is not remotely sustainable as a large-scale replacement for the traditional financial system, which makes Bitcoin around 5033 times more power intensive, per transaction, than VISA [55]. In other words, Bitcoin is *energy intensive*.

A single transaction needs as much electricity as the daily consumption of 1.6 American households, and this number keeps growing. Bitcoin transactions are processed and validated by a decentralized network of **volunteers for performing calculations**, known as “hashes” to find answers to a complex mathematical algorithm in return for a reward of a small number of Bitcoins. The network of Bitcoin **miners** guarantees the security of the system with the cost of consuming a large amount of electricity. According to Deetman’s study, it is around 350 MW (megawatt), which is approximately equivalent to the electricity demand of 280000 American households. Therefore, if the Bitcoin network keeps growing the way it has done, it could lead to constant electricity consumption that lies between the output of a small electric power station and the total consumption of a small country such as Denmark by 2020 [56].

Limited governance

Compared to the physical limitations, governance is even more problematic. Incentives, protocols, and rewards affect system efficiency [31], [52]. During the last few years, the blockchain community has witnessed that every proposed *protocol change* is creating tension because it could threaten investments returns and affect their business models. Bitcoin network is a distributed system, however, it has extremely centralized governance. Perhaps, the power of governance is limited because the technology could **function independently**, outside the original network and rules.

The DAO (Decentralized Autonomous Organization) is a good example. After someone, profiting from an *unexpected code path*, managed to move $55 million [53] **into a clone** of The DAO which was held by the attacker. Right after the incident, the Ethereum community imposed a hard fork to reverse the transaction and created a new chain called **Ethereum Classic**. There are now two coexisting Ethereum chains, one with the $55 million transaction and the other one without. This case arises the question about fundamental immutability of blockchain and demonstrates that governance in distributed networks is a thorny matter because minorities are able autonomously to separate from the network while keeping the technology and assets trading on parallel forks. In conclusion, technology is not neutral, and technical adjustments might have practical implications for business models and power-balances.

3 Implementing blockchain technology

In this chapter, a design for managing copyright royalties in the music industry using blockchain technology and smart contracts is proposed. The author chose Fanvestory start-up as a *use case* because the purpose is not just theoretical, but practical implication is to test and implement the solution **in a real business environment**. The author of the thesis is one of the co-founders of Fanvestory start-up and has general knowledge about the industry. During implementation, the focus is mainly on the needs of the start-up. However, since the start-up is in the same sector and has **similar tasks as CMOs** have on a wider scale, *collecting and distributing royalties* to the stakeholders, it is possible to use this solution for their needs as well. Fanvestory is a platform where music fans can buy a piece of music and be entitled to its future royalties while supporting the careers of the artists' they believe in. Technically, the platform **enables fans to become stakeholders** of a song or an album (see Figure 3.1). This setting has been closed for fans for couple hundreds of years. If previously there were only a few stakeholders who could participate in the creation, then today it's possible to have thousands of *fans as stakeholders*, who have purchased a small share and earn copyright royalties together with the artist.

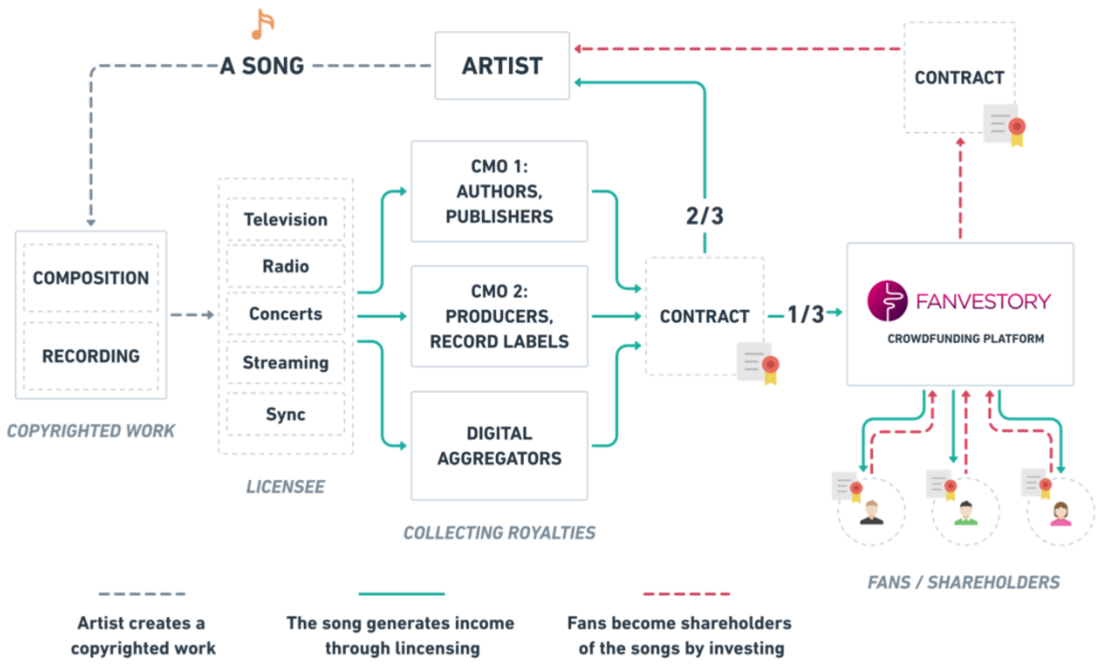

Figure 3.1 How Fanvestory works in the music industry.

3.1 Use cases and requirements

This section describes the use cases and non-functional requirements that apply to the system. The implementation outlined in this thesis is limited to four fundamental use cases: registration of the intellectual property, purchasing a share, licensing, distributing royalties.

Registering the property

Fanvestory start-up and CMOs should be able to *register copyrighted work* to a shared database, which should be accessible for certain parties, as can be seen from Table 3.1. The database should contain all the necessary information for managing copyright royalties, such as work title, artist, authors etc.

Table 3.1 Description of the use case "Registering the property".

Name	*Registering the property*
Goal	Register intellectual property to the database.
Pre-condition	User account has to be unlocked.
Post-condition	A new property has been created successfully.
Post-condition in special case	The property already exists.
Normal case	• Create a new instance of a **Property** struct and push data to this struct. • Add a property to the properties mapping. • A new property has been created.
Special case	The property creation failed because it already exists in the database.

Purchasing a share

The fans should be able to purchase a share in a copyrighted work and become shareholders with the creator. Since the purchase amounts might be as small as 10€, the system should support giving out small share amounts such as 0.001%. The process has to be fast and transparent because one work might have thousands of shareholders. A detailed description of this use case can be observed in Table 3.2.

Table 3.2 Description of the use case "Purchasing a share".

Name	*Purchasing a share*
Goal	Purchase a share in the copyrighted work.
Pre-condition	User account has to be unlocked. The property has to exist. The desired amount of shares has to be available.
Post-condition	Purchasing a share was successful.
Post-condition in special case	Purchasing a share has failed.
Normal case	• Move the shares from one account to another. • Create new purchase entity. • Create a new instance of a struct and push the data to this struct. • The share will be purchased.
Special case	The purchase was unsuccessful.

Licensing the property

Copyright royalties are generated for various types of licensing and usage. As shown in Table 3.3, it is important to be able to license a copyrighted work and have the capability to process this data. All the necessary data about licensing has to be collected and traceable.

Table 3.3 Description of the use case "Licensing the property".

Name	*Licensing the property*
Goal	License a copyrighted work.
Pre-condition	User account has to be unlocked. The property has to exist.
Post-condition	The work has been successfully licensed.
Post-condition in special case	Licensing the work has failed.
Normal case	• Create new license entity. • Create a new instance of a struct and push the data to this struct. • The work will be licensed.
Special case	Licensing the work was unsuccessful.

Distributing royalties

Royalties are the sums paid to rights-holders when their creations are sold, distributed, embedded in other media or monetized in any other way (see Table 3.4). All the rights-holders, large or small should be able to receive royalties from the copyrighted works, depending on the share they own. To know exactly, to whom and how many royalties to transfer, the system has to go through all the owners and calculate precise amounts.

Table 3.4 Description of the use case "Distributing royalties".

Name	*Distributing royalties*
Goal	Distribute copyright royalties to the owners (shareholders).
Pre-condition	User account has to be unlocked. The property has to exist.
Post-condition	The royalties have been successfully distributed to the owners.
Post-condition in special case	Distributing royalties to the owners has failed.
Normal case	• Create new royalty distribution entity. • Create a new instance of a struct and push the data to this struct. • Go through all the owners and calculate the royalties. • The royalties will be distributed among the owners.
Special case	The royalties could not be distributed.

Figure 3.2 demonstrates a simplified data model behind the use cases, for more details see Chapter 3.5.1.

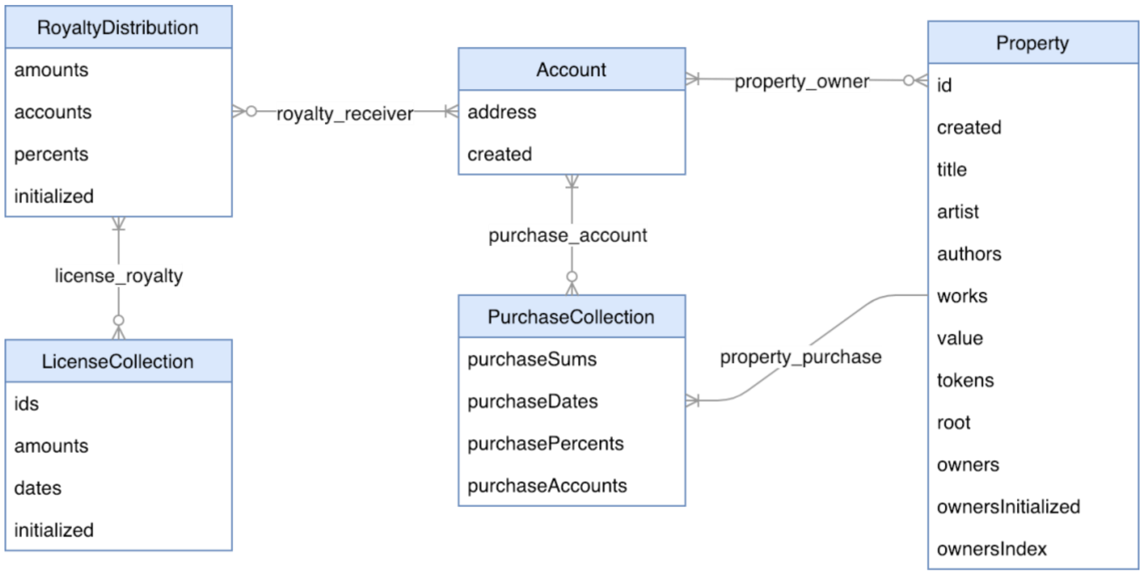

Figure 3.2 A simplified data model behind the use cases.

Besides user stories, there are also **two non-functional requirements** that apply to an entire system not just to one user individually.

NFR1: Privacy

The system should not be entirely public or private, but somewhere in between. Consortium systems operate under the leadership of different parties instead of a single entity. Therefore, together they should be able to decide who has access to the system.

NFR3: Security

It is important that the system is built in the way, that it is extremely difficult for hackers to manipulate the data. It is extremely difficult for an individual or a group to tamper with the record unless they control the majority of the miners. The risk of a 51% Attack has to be measured to prevent it from happening.

3.2 Choosing the technology

Ethereum and Hyperledger are flexible inasmuch as Corda was designed particularly to focus on the financial industry (see Table 3.5). Ethereum runs smart contracts which are meant for mass consumption and built to be decentralized. Solidity, which is a programming language for Ethereum is designed to be understandable and simple for implementation. The power of this technology is primarily its programmability, agreements are written in the code which allows executing transactions automatically. These digital agreements in other words smart contracts, have limitless conditions, formats and even call on other contracts, making Ethereum useful for arbitrating transactional events.

Popularity and stability and of Ethereum has grown noticeably and has proven to be highly robust against different attacks. Since the technology is open source, it is not tied into a specific IT environment of a single vendor. It is remarkably easy to set up the technology on Amazon Web Services. Taking all of the aforementioned into consideration, these are the reasons why the author chose this technology.

Table 3.5 Comparison of Ethereum, Hyperledger Fabric and Corda [57].

	Ethereum	**Hyperledger Fabric**	**R3 Corda**
Description	- Generic blockchain platform	- Modular blockchain platform	- Specialized distributed ledger platform for a financial industry
Governance	- Ethereum developers	- Linux Foundation	- R3
Mode of operation	Permissionless, public or private	Permissioned, private	Permissioned, private
Consensus	- Mining based on proof-of-work (PoW) - Ledger level	- Broad understanding of consensus that allows multiple approaches - Transaction-level	- Specific understanding of consensus (i.e. notary nodes) - Transaction-level
Code	- Solidity	- Go - Java	- Kotlin - Java
Currency	- Ether - Tokens via smart contract	- None - Currency and tokens via Chaincode	- None

3.3 Used technologies, services and, programming languages

The following section describes most of the important technologies, services and programming languages that the author used in this thesis.

JetBrains PhpStorm

JetBrains PhpStorm is a commercial, cross-platform integrated development environment (IDE) for Hypertext Preprocessor (PHP). It was designed specifically to facilitate the development of the application by providing tools and features for PHP and supporting front-end technologies.

Solidity

Solidity is a programming language that is contract-oriented and meant for writing smart contracts. It is used for implementing smart contracts on blockchain platform. Solidity has a large number of programming perceptions that exist in other languages - string manipulation, functions, variables, classes, arithmetic operations.

Web3.js

A collection of libraries named web3.js allows interacting with a local or remote Ethereum node, using an *HTTP or IPC connection*. Ethereum network is made up of nodes, which each contain a copy of the blockchain. Calling a function on a smart contract, it's necessary to query one of these nodes and tell it:

- The address of the smart contract;
- the function to call;
- the variables to pass to that function.

Ethereum nodes only speak a language called **JSON-RPC**, which is difficult for humans to understand. JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around processing.

As shown in Figure 3.3, a query to tell the node to call a function on a contract looks something like this:

```
{"jsonrpc":"2.0","method":"transfer","params":[{"from":"0xb9h4pvd61c
5d32be8058bb9l2m10870f07233155","to":"0xd46e8d86c04d32be8058bb8eb970
870f0cb39f67","gas":"0x76c0","gasPrice":"0x9324e72e500","value":"0x9
8j4sb2a","data":"0xd46e8dd147c532be8d46e8dd67c5d310bj58bb8eb970870f0
72445675058bb8l78n370f072445621"}],"id":10}
```

Figure 3.3 A query example written in JSON-RPC.

Figure 3.4 represents how Web3.js deals with these complex queries and instead of needing to construct the above query, calling a function in your code will look like this:

```
Fanvestory.methods.transfer("Kristjan Ulst").send({
      from: "0xb9h4pvd61c5d32be8058bb9l2m10870f07233155",
      gas: "1000000"
})
```

Figure 3.4 A query example for Web3.js.

Geth

Geth is a multipurpose *command line tool* that runs a full Ethereum node implemented in Go. The tool offers three interfaces:

- the command line subcommands and options,
- a JSON-RPC server,
- an interactive console.

Ethereum Remix

Remix, previously known as Browser Solidity, is a browser-based compiler and IDE that allows to *build Ethereum contracts* with Solidity language and to *debug transactions*. The compiler is written in JavaScript and it supports both usage in the browser and locally.

There are 3 types of environments Remix can be plugged to:

- JavaScript VM,
- Injected provider,

- or Web3 provider.

Node.js

Node.js is an open-source, cross-platform *JavaScript run-time environment* which allows executing JavaScript code outside of a browser. Both the browser JavaScript and Node.js run on the V8 JavaScript runtime engine. This engine takes the JavaScript code and converts it into **faster machine code**. Machine code is low-level code which the computer can run without needing to first interpret it. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

Supervisor

Supervisor is a client-server system that allows to *monitor and control a number of processes* on UNIX-like operating systems. Processes could be grouped into "process groups" and a set of logically related processes could be stopped and started as a unit. The operating system signals Supervisor immediately when a process is being terminated.

Supervisor is based on four components:

- **Supervisord**: A server piece which is responsible for starting child programs at its own invocation, responding to commands from clients, restarting crashed or exited sub-processes.

- **Supervisorctl**: Command-line client which provides a shell-like interface to the features provided by supervisord. Allows to connect to different supervisord processes, get status on the sub-processes and get lists of running processes of a supervisord.

- **A web server**: A user interface with functionality comparable to supervisorctl may be accessed via a browser if starting supervisord against an internet socket.

- **An XML-RPC interface**: HTTP server which serves the web UI serves up an XML-RPC interface that can be used to interrogate and control supervisor and the programs it runs.

Amazon Web Services

Amazon Web Services (AWS) is a cloud services platform, which offers compute power, content delivery, database storage and other functionalities helping to scale and grow. The author used this service for the blockchain environment.

3.4 Setup Ethereum environment

In order to create and deploy smart contracts, it is necessary to set-up and configure the environment. Therefore, installing software properties and adding Ethereum repository is needed - Figure 3.5. A server environment called *Node.js* and a collection of libraries *Web3.js* which were described in the previous section are also required. Ethereum blockchain environment is in Amazon because startups' other services are **built on the top of AWS** and it is easily scalable.

```
sudo apt-get install software-properties-common
sudo add-apt-repository -y ppa:ethereum/Ethereum
sudo apt-get update
sudo apt-get install ethereum
```

Figure 3.5 Installation of Geth and private blockchain on Ubuntu system [57].

After the environment is installed, the next step is to *initialize and configure* the blockchain using Geth command line tool. Geth requires two main parameters, a folder to store the chain data and an initialization file. The file **genesis.json** is genesis block which is the start of the blockchain and this file that defines it. It is a configuration file for the blockchain.

A new chain with genesis configuration and console will be initialized after executing the commands that are shown in Figure 3.6. The console has a number of libraries to interact with the blockchain.

```
geth --datadir ./datadir init genesis.json
geth --datadir ./datadir console
```

Figure 3.6 Initializing blockchain with Geth.

The *console* has a number of libraries to interact with the blockchain. It has a JavaScript console (read, evaluate and print loop), which can be opened with the console subcommand. The console subcommands will execute the Geth node and open the console.

Finally, the last step is starting the blockchain. The following command, shown in Figure 3.7, executes Geth interface and exposes it via WebSocket port 8545.

```
geth --datadir ./datadir --networkid 2018 --port 30306 --nodiscover
--ws --wsapi "db,personal,eth,net,web3,debug" --wsorigins "*" --wsad
dr="localhost" --wsport 8545 console
```

Figure 3.7 Open WebSocket API.

3.5 Smart contracts

This section will describe how to system works and dive in to the architecture of smart contracts.

Figure 3.8 demonstrates the system from a **high-level perspective**. From the top to bottom the model is an "*abstraction-first*" approach to visualize the software architecture. The diagram is not comprehensive and is simplified for sake of importance.

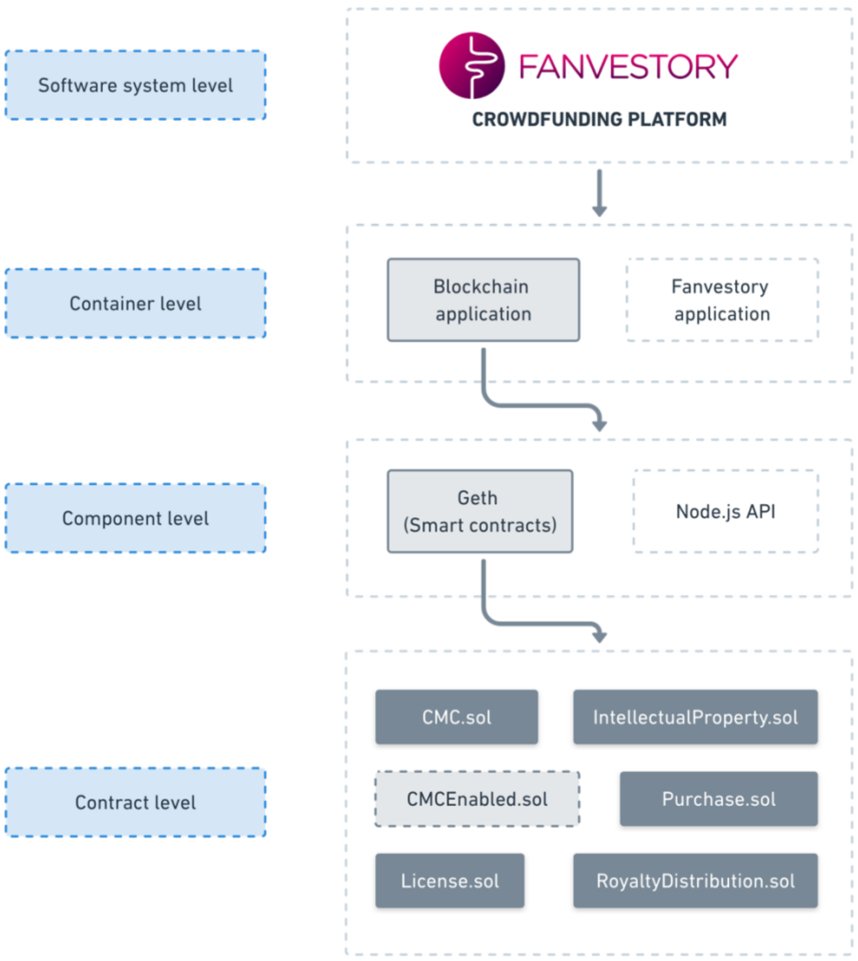

Figure 3.8 System high-level overview of Fanvestory platform. C4 modeling technique by Brown [58].

All the necessary data about *ownership, licensing, royalties and metadata* is in a distributed ledger. The contracts for Ethereum blockchain have been written in Solidity version 0.5.0.

3.5.1 Data structures

The *Property* struct (data structure) contains the core data of the system. As shown in Figure 3.9, the necessary information about **IP (Intellectual Property)** is stored in this struct, including the authors and owners.

```
struct Property {
    uint64 id;
    uint64 created;
    string title;
    string artist;
    string authors;// String arrays not supported as parameters yet,
so use custom separator in implementation
    string works;// String arrays not supported as parameters yet, s
o use custom separator in implementation
    uint64 value;// Value in money times 100, e.g. 1000EUR would be
100000, this is the total value of all tokens
    uint64 tokens;// Percent times 10000, e.g. 33% would be 330000
    address root;
    mapping(address => uint64) owners;// All the owners tokens of th
e Property
    mapping(address => uint64) ownerInitialized;
    address[] ownersIndex;
}
```

Figure 3.9 Property data structure.

Since the users are able to purchase shares of the IP, *PurchaseCollection* struct stores all the data regarding **purchasing and ownership**, presented in Figure 3.10. It is an array of purchases which is being used in *properties* mapping to filter out purchases by specific IP.

```
struct PurchaseCollection {
    // Purchase amounts
    uint64[] purchaseSums;
    uint64[] purchaseDates;
    // Tokens
    uint64[] purchasePercents;
    // Purchase owners
    address[] purchaseAccounts;
    uint64 initialized;
}
```

Figure 3.10 Purchase collection data structure.

For demonstration purposes, let's assume there are two purchases. User **A** has made a purchase for *100€* and received *0.1%* of shares in the IP and user **B** purchased *0.2%* shares for *200€*. As shown in Figure 3.11, in this case the data will be stored like this.

```
purchaseSums = [10000, 20000]
purchaseDates = [1544194702, 1544193702]
purchasePercents = [1000, 2000]
purchaseAccounts = [0x3a6391f2cfb0ef38f984941345d18b653fa889bb, 0x86
8be045edf63aacf03b3223d35d547656f008ce]
initialized = 1
```

Figure 3.11 A demonstration of purchase collection struct.

Following data structures called *LicenseCollection* and *RoyaltyDistributions* (see Figure 3.12) contain information about **collecting and distributing royalties** and are similar to the previous struct.

```
struct LicenseCollection {
    uint256[] ids;
    uint64[] amounts;
    uint64[] dates;
    uint64 initialized;
}

struct RoyaltyDistributions {
    address[] accounts;
    uint64[] amounts;
    uint64[] percents;
    uint64 initialized;
}
```

Figure 3.12 License collection and royalty distributions data structures.

3.5.2 Creating a property

The function *addProperty* is the application's core function that *creates and registers* new copyrighted work (Figure 3.13). The work gets a unique ID and will be pushed into *properties* array. All the *tokens*, which **represent work ownership** will be given to the owner of the contract. In other words, if the owner owns 100% of the work, the result is 1,000,000 *tokens*.

```
function addProperty(uint64 id, uint64 created, string memory title,
string memory artist, string memory authors, string memory works, ui
nt64 value, uint64 tokens, address root) public returns (bool) {
    // Check if this property does not exist
    require(properties[id].id == 0);
    // Create new empty variable
    address[] memory owners;
    // Create a new instance of a struct
    Property memory prop = Property({
        id : id,
        created : created,
        title : title,
        artist : artist,
        authors : authors,
        works : works,
        value : value,
        tokens : tokens,
        root : root,
        ownersIndex : owners
        });

    // Add new property to properties mapping
    properties[id] = prop;
    // Give all tokens to owner
    properties[id].owners[root] = tokens;
    properties[id].ownerInitialized[root] = 1;
    properties[id].ownersIndex.push(root);

    emit CreateProperty(id);

    return true;
}
```

Figure 3.13 A function for creating and registering a new work.

3.5.3 Purchasing a share

The function *addPurchase*, that is illustrated in Figure 3.14, allows users to buy ownership in a work. The *value* of the work defines how much ownership (*tokens*) the buyer will get. Inside this function, there is another function call *transfer*, which will move ownership from one user to another. In this case, a certain amount of ownership from the contract owner will be given to the buyer.

```
function addPurchase(uint64 id, uint64 timestamp, uint64 money, addr
ess account) public returns (bool) {
    IntellectualProperty _Property = IntellectualProperty(CMC(CMCAdd
ress).getContract("IntellectualProperty"));
    // Calculate how much percentage will be given
    uint64 percent = (money * 1000000) / _Property.getPropertyValue(
id);
    uint64 rootPercent = _Property.getRootPercent(id);
    // Check if there is enough percentage available
    require(rootPercent >= percent);
    // Call transfer function
    bool result = _Property.transfer(id, percent, account, _Property
.getRootAddress(id));
    initializePurchases(id);
    // Push the purchase to the collection
    purchases[id].purchaseSums.push(money);
    purchases[id].purchaseDates.push(timestamp);
    purchases[id].purchasePercents.push(percent);
    purchases[id].purchaseAccounts.push(account);

    emit PurchaseAdded(id, timestamp, money, account);
    return result;
}
```

Figure 3.14 A function for purchasing a share in work.

3.5.4 Licensing and distributing royalties

Licensing and royalty distribution between the shareholders are handled by the methods *createLicense* and *createDistributions*, shown in Figure 3.15.

Whenever the work is being licensed, the funds will be distributed between all the shareholders. The method *createDistributions* goes through all the shareholders and calculates precise amounts for each user, depending on how many shares the user has. The necessary data about licensing will be pushed to **Licenses** array and the data regarding royalty distributions is in **royaltyDistributions** array.

```
function createLicense(uint64 id, uint64 amount, uint64 date) public
returns (bool) {
    initializeLicense(id);

    uint256 royaltyId = Licenses[id].amounts.length + 1;
    // Push licenses to the collection
    Licenses[id].ids.push(royaltyId);
    Licenses[id].amounts.push(amount);
    Licenses[id].dates.push(date);

    RoyaltyDistribution _Dist = RoyaltyDistribution(CMC(CMCAddress).
getContract("RoyaltyDistribution"));
    _Dist.createDistributions(royaltyId, id, amount);

    emit CreateLicense(id, amount, date);

    return true;
}

function createDistributions(uint256 royaltyId, uint64 propertyId, u
int64 amount) public returns (bool) {
    initializeDistributions(royaltyId);
    IntellectualProperty _Property = IntellectualProperty(CMC(CMCAdd
ress).getContract("IntellectualProperty"));
    address[] memory owners = _Property.getPropertyOwners(propertyId
);
    // Go through all the Property owners
    for (uint64 i = 0; i < owners.length; i++) {
        // Owners account
        address ownerAccount = owners[i];
        // Owners Property share in tokens
        uint64 percent = _Property.getPropertyBalance(propertyId, ow
nerAccount);
        // Calculate how much of the incoming royalty goes to the ow
ner
        uint64 ownerAmount = (amount * percent) / 1000000;
        // Push royalties to the collection
        royaltyDistributions[royaltyId].accounts.push(ownerAccount);
        royaltyDistributions[royaltyId].amounts.push(ownerAmount);
        royaltyDistributions[royaltyId].percents.push(percent);
    }

    return true;
}
```

Figure 3.15 Functions for licensing and distributing royalties.

3.6 Interacting with smart contracts

There are two different aspects to understand when developing Ethereum blockchain applications:

1. **Developing smart contracts**: writing code which will be deployed to the blockchain network.

2. **Developing applications** to interact with the blockchain: writing code which allows to read and write data from the blockchain using *smart contracts*.

Web3.js fulfills the second aspect that was previously pointed out. The library allows developing clients that are interacting with the blockchain. Comparable to web development, where jQuery is being used to make Ajax calls to the web server, Web3.js is used to read and write the Ethereum blockchain. The following Figure 3.16 shows how the client communicates to Ethereum through Web3.js.

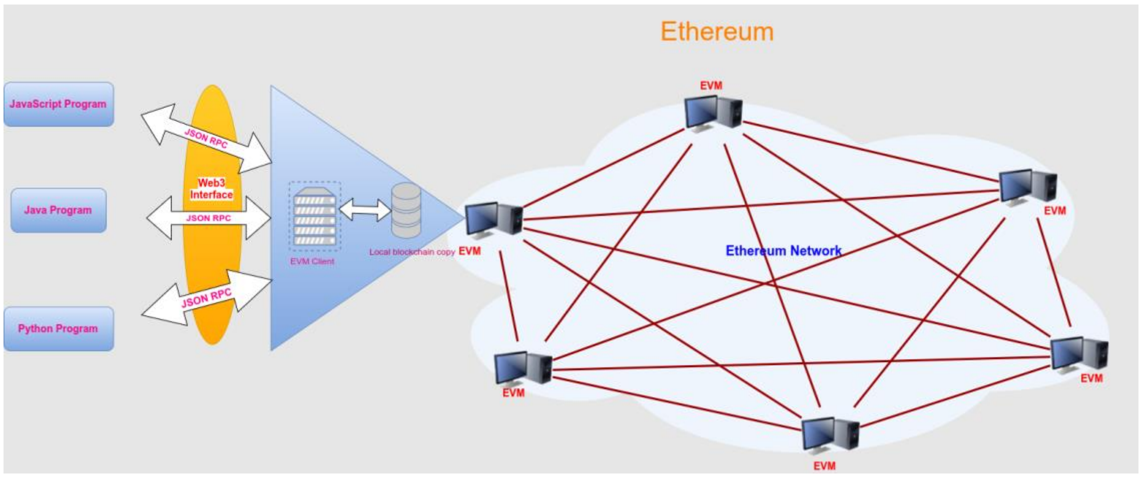

Figure 3.16 Ethereum Virtual Machine diagram [59].

To be able to read information from the smart contracts, two elements are necessary:

1. A JavaScript representation of the smart contracts to interact with
2. A method to call the functions on the smart contracts to read the data

For a JavaScript representation of the smart contract, it is possible to use a function called **web3.eth.Contract()**, which expects two arguments: *Application Binary Interface* (ABI) and *contract address*. ABI describes how a specific smart contract works by presenting the data in JSON array.

4 Results and Analysis

This chapter presents the results of the research done. The first part looks at the results on the Ethereum smart contracts. The second part of the chapter describes Fanvestory project example and presents the findings. Thirdly, the possibilities of using the presented solution in other industries will be discussed.

4.1 Ethereum and smart contracts

As a result of this thesis, a technical solution for managing copyright royalties was created. The **application is distributed**, meaning it does not belong to a single corporation or is limited by geographical locations. It is a consortium system that operates under the leadership of related parties. The real-time data is transparent and also removes the problem regarding the "black box". In addition, this solution creates a marketplace for the fans, by allowing them to be part of the system.

Created and deployed smart contracts in Ethereum network allow to register intellectual properties, purchase shares, license and distribute royalties. As shown in Figure 3.8, in total there are five different smart contracts: *CMC, IntellectualProperty, Purchase, License, and RoyaltyDistribution*. The entire code regarding the contracts is available as an Appendix A.

CMC, in other words, *Contract Managing Contracts* is a registry that keeps track of contracts in the system. After creation, each contract is deployed separately and added to the registry. There are two significant benefits for this approach, which provide **code upgradeability and maintainability**:

1. Inside a contract, it is possible to use another smart contract.
2. It allows updating contracts separately, without updating the whole code.

Besides the smart contracts, an important part of the technical solution is an interface that is interacting with contracts (see Appendix B and Appendix C). The created interface is necessary for reading and writing blockchain data via REST API.

4.2 Fanvestory project example

In 30th of April 2017 start-up Fanvestory launched a project with an Estonian rapper and conceptual artist Tommy Cash, who is famous for his notorious music videos. Through Fanvestory platform he sold 20% of the song “Rawr” future royalties for 10 years for the fans [60]. As a result, in one hour after the project was launched, 182 fans raised 12,000€ for the artist and became shareholders of that song. In June 2018, after collecting royalties for the song, the fans earned 6,000€ from concerts, radio, TV, downloads, streaming and synchronization. This project was chosen as an example to demonstrate the results of the thesis. The data regarding intellectual property, shareholders, purchases, licenses and royalty distributions was successfully **transferred to the blockchain network**.

This approach allows the music industry to collaborate on a global view of copyright ownership and royalties. Using blockchain-based IP registry gives transparency to copyright authors, owners, and users. The data is also *tamper-proof*, meaning once the work is added to the blockchain, it cannot be lost or changed. Instead of having numerous independent databases, with no communication with each other and where the data and the structure may vary, there is **one global shared registry**. A global database makes the processing and distributing copyright royalties more efficient with faster and cheaper ways to obtain a license since intermediaries’ operational and administrative costs between licensee and owner are reduced to the minimum.

It is estimated that with the **created solution** the royalty distribution time among 100 owners is shortened by an immense amount of time from 6 months to 5 minutes. Also, the amount of music payments that do not make it to their rightful owners is most likely eliminated (explained in Table 4.1).

Table 4.1 Royalty distribution currently and with the created solution. See Chapter 2.2.2 and Chapter 2.2.3.

Metric	Currently	With the created solution
How long does it take to distribute royalties among **100** owners?	6-12 months	5 minutes
Among **1 000** owners?	6-12 months	50 minutes
A percentage of overall music payments that **do not** make it to their rightful owners ("the black box").	20-50	Minimal to 0
An amount in US$ billions.	3.5-8.6	Minimal to 0

In addition, since Fanvestory is processing and distributing copyright royalties to thousands of shareholders as a large number of micro-transactions, the created solution gives to the start-up a competitive advantage by being more efficient.

4.3 Extension into other industries

The purpose of the current thesis was to solve a specific problem in the music industry, nevertheless, the potential of using the presented solution in other sectors is remarkable. In fact, different blockchain use cases by now are emerging in other creative industries, including **art, books, film, journalism and, games**, as well as in fields from e-voting, payments, and internet-of-things, which are described in Chapter 2.4.3.

Writers, singers, artists, designers and other content creators face ancient problems when it comes down to *getting paid and managing their creation*, in other words, intellectual property. Very often there are costly disputes about the ownership because it is unclear who owns what. Faulty data may lead to an unfair distribution of revenue. Intellectual property rights are often held by competing parties. Expecting each to provide data to a single centralized database, has **proven to be an insuperable challenge till now**. Thus, the proposed solution that was presented by the author in Chapter 3, may presumably solve numerous issues in other creative industries as well.

However, it is difficult to conclude whether or not the results would be satisfactory on a wider scale, where more parties, such as labels, publishers and CMOs are involved. The author concludes that if there is friction, **it is not** between a specific *technology and copyright*. Rather, the friction is between the social, political, and financial conditions that produced the blockchain technology environment or the social, political, and financial premises from which the current copyright system developed. This area needs further investigation because it might not be that simple to scale the suggested solution.

Furthermore, the application is data-intensive and executing code on blockchain side could be quite expensive. The system should be designed the way, that it does the computation mostly on the client side and at least as possible on the blockchain.

5 Conclusion

The chapter analyses whether the defined problem was solved, and the set objectives were achieved.

One of the most problematic issues in the music industry today is the "black box" - the rightful owner of the copyright revenue cannot be determined **due to poor transparency** of the industry-wide system, which was explained in Chapter 2.2.1. The fundamental reasons behind it are *a missing global registry* for copyright and *no international standards* for data reporting, which makes processing and distributing copyright royalties **inefficient** (see Chapter 2.2.2).

Blockchain technology has made it possible to create a single and distributed database of musical creators and their works. The application is distributed and does not belong to a single corporation, nor it is limited by geographical locations. Implementing this solution could provide correct contract information about music owners and copyright royalties.

A technical solution with Ethereum and smart contracts was built, proving that **properties of blockchain could be applied** to solve the issues that were pointed out in the research question. The created application covers the needs of the start-up company called Fanvestory. It is distributed and does not belong to a single corporation, nor it is limited by geographical locations. The data is transparent, resolves the issue of regarding the "black box", and is accessible to related parties in real-time. Moreover, distributing and processing time for royalties among 100 owners was reduced enormously, **from 6 months to 5 minutes**.

5.1 Future work

The author identified two key challenges and areas of improvements to future work.

Privacy improvements

With private blockchain network, the integrity of the information is only as strong as the honesty of each participant. The security framework is not so much unsafe **as it is a social contract**. Nodes are expected to behave but can act maliciously until they are removed *by the consortium* through an off-chain decision process.

Many assume that a private blockchain network is secure purely due it's inconvenient naming of being *"private"*. Of course, all transactions are private to the nodes within the membership of the consortium. However, once membership is granted, a node will gain access to the confidential information, not just between the members, but between **everyone in the consortium**. Future work should investigate how enterprises could securely share confidential data on the blockchain.

Reducing human error

Since the blockchain is **used as a database**, the data going into the database needs to be of *high quality*. The data stored on a blockchain is not inherently trustworthy, so events need to be recorded accurately in the first place.

It is clear that the blockchain network involves different parties, whether they are *developers, users or even hackers*. When discussing opportunities for improvement, **educating** users and developers is a first very important step in reducing human error. Simultaneously, developers have to be educated to provide *secure platforms and minimize bugs*. Human error was something that was not in the scope of this thesis and human error probability needs investigation in blockchain technology.

To summarize the above, the field of blockchain technology, especially in terms of smart contracts, seems to be rather unresearched and has much to offer for future research.

References

[1] B. Rosenblatt, "Watermarking Technology and Blockchains in the Music Industry," Digimarc Corporation, Beaverton, 2017.

[2] Royalty Exchange, "Music Royalties Guide," [Online]. Available: https://www.royaltyexchange.com/learn/music-royalties. [Accessed 8 October 2018].

[3] U.S. Copyright Office, "Copyright Law of the United States," 2016. [Online]. [Accessed 24 September 2018].

[4] A. K. Fleisher, "Confused by Music Copyright? Here Are 5 Things You Definitely Need to Know," 2018. [Online]. Available: https://www.digitalmusicnews.com/2018/03/11/music-copyright-basics/. [Accessed 25 September 2018].

[5] C. Cooke, "MMF (Music Managers Forum)," 2015. [Online]. Available: https://themmf.net/site/wp-content/uploads/2015/09/digitaldollar_fullreport.pdf. [Accessed 5 June 2018].

[6] I. Shepard, "How does the music industry work? Give me the high-level overview!," 2017. [Online]. Available: https://www.themusicmaze.com/music-industry-work-give-high-level-overview/. [Accessed 15 October 2018].

[7] A. X. Wang, "How Musicians Make Money — Or Don't at All — in 2018," 2018. [Online]. Available: https://www.rollingstone.com/music/music-features/how-musicians-make-money-or-dont-at-all-in-2018-706745/. [Accessed 26 September 2018].

[8] R. PQ, "How Music Royalties Work in the Music Industry," 2018. [Online]. Available: https://iconcollective.com/how-music-royalties-work/. [Accessed 4 October 2018].

[9] Rethink Music, "Fair Music: Transparency and Payment Flows in the Music Industry," BerkleeICE, Boston, 2015.

[10] European Commission, "State of the Union 2016: Commission proposes modern EU copyright rules for European culture to flourish and circulate," 2016. [Online]. Available: http://europa.eu/rapid/press-release_IP-16-3010_en.htm. [Accessed 10 June 2018].

[11] European Commission, "Directive on collective management of copyright and related rights and multi-territorial licensing," 2014. [Online]. Available: http://europa.eu/rapid/press-release_MEMO-14-79_en.htm. [Accessed 1 July 2018].

[12] Teosto, "Teosto develops a blockchain platform for music copyright organisations," 2017. [Online]. Available: https://www.teosto.fi/en/teosto/news/teosto-develops-blockchain-platform-music-copyright-organisations. [Accessed 4 July 2018].

[13] PRS for music, "UK royalty payment dates," 2018. [Online]. Available: https://www.prsformusic.com/royalties/royalty-payment-dates#anchor_1489742717076. [Accessed 5 July 2018].

[14] S. Sartin, "The Problem With Royalties," 2018. [Online]. Available: https://medium.com/choonhq/the-problem-with-royalties-a2d2e5250f0b. [Accessed July 6 2018].

[15] V. Shponka, "Music Industry Problems and How They Can Be Solved with the Blockchain," 2018. [Online]. Available: https://rubygarage.org/blog/blockchain-in-music-industry. [Accessed 6 July 2018].

[16] P. Resnikoff, "20-50% of Royalties Never Reach the Artist, Study Finds…," 2015. [Online]. Available: https://www.digitalmusicnews.com/2015/07/15/20-50-of-royalties-never-reach-the-artist-study-finds/. [Accessed July 11 2018].

[17] IFPI, "IFPI Global Music Report 2018," 2018.

[18] B. Oktaviana and A. P. U. Siahaan, "Three-Pass Protocol Implementation in Caesar Cipher Classic Cryptography," *Journal of Computer Engineering,* vol. 18, no. 4, pp. 26-29, 2017.

[19] N. G. McDonald, *Past, present, and future methods of cryptography and data encryption,* University of Utah, 2009.

[20] K. Moore, S. Dash, E. Ross and C. Lin, "Caesar Cipher," 2018. [Online]. Available: https://brilliant.org/wiki/caesar-cipher/. [Accessed 20 October 2018].

[21] H. Delfs and H. Knebl, "Symmetric-Key Cryptography," in *Introduction to Cryptography,* Berlin, Springer, Berlin, Heidelberg, 2015, pp. 11-48.

[22] Oracle Corporation, "Key Encryption," 2010. [Online]. Available: https://docs.oracle.com/cd/E19424-01/820-4811/6ng8i26bn/index.html. [Accessed 20 October 2018].

[23] A. Salomaa, "Classical Two-Way Cryptography," in *Public-Key Cryptography,* Turku, 2013, pp. 2-20.

[24] Z. Lyasota, "Digital Signature in Blockchain," 2018. [Online]. Available: https://dzone.com/articles/digital-signature-2. [Accessed 20 July 2018].

[25] M. Selvamanikkam, "Digital Signature Generation," 2018. [Online]. Available: https://medium.com/@meruja/digital-signature-generation-75cc63b7e1b4. [Accessed 2 November 2018].

[26] V. Buterin, *A Next-Generation Smart Contract and Decentralized Application Platform,* 2014.

[27] S. Peyrott, "An Introduction to Ethereum and Smart Contracts: a Programmable Blockchain," 2017. [Online]. Available: https://auth0.com/blog/an-introduction-to-ethereum-and-smart-contracts-part-2/. [Accessed 21 August 2018].

[28] C. Dannen, Introducing Ethereum and Solidity, Brooklyn: Apress, 2017, pp. 89-92.

[29] J. Biggs, "Sierra Leone just ran the first blockchain-based election," 2018. [Online]. Available: https://techcrunch.com/2018/03/14/sierra-leone-just-ran-the-first-blockchain-based-election/. [Accessed 5 September 2018].

[30] K. Leary, "Blockchain might be about to change the way we vote," 2017. [Online]. Available: https://www.weforum.org/agenda/2017/09/blockchain-could-be-about-to-change-how-you-vote. [Accessed 6 September 2018].

[31] K. Wüst and A. Gervais, Do you need a Blockchain?, Zurich, 2017.

[32] T. Aste, P. Tasca and T. Di Matteo, "Blockchain Technologies: The Foreseeable Impact on Society and Industry," IEEE, 2017, pp. 18-28.

[33] N. Popper, "Decoding the Enigma of Satoshi Nakamoto and the Birth of Bitcoin," 2015. [Online]. Available: https://www.nytimes.com/2015/05/17/business/decoding-the-enigma-of-satoshi-nakamoto-and-the-birth-of-bitcoin.html. [Accessed 3 August 2018].

[34] A. Kharpal, "Everything you need to know about the blockchain," 2018. [Online]. Available: https://www.cnbc.com/2018/06/18/blockchain-what-is-it-and-how-does-it-work.html. [Accessed 3 August 2018].

[35] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[36] B. Marr, "A Very Brief History Of Blockchain Technology Everyone Should Read," 2018. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/02/16/a-very-brief-history-of-blockchain-technology-everyone-should-read/#50c535e77bc4. [Accessed 6 August 2018].

[37] S. Bradley, "What is Blockchain?," 2018. [Online]. Available: https://www.techadvisor.co.uk/feature/internet/what-is-blockchain-3671209/. [Accessed 7 August 2018].

[38] ETMarkets.com, "What is the difference between bitcoin and ethereum?," 2017. [Online]. Available: https://economictimes.indiatimes.com/markets/stocks/news/what-is-the-difference-between-bitcoin-and-ethereum/articleshow/62171361.cms. [Accessed 11 August 2018].

[39] V. Buterin, Interviewee, *Decentralizing Everything with Ethereum's Vitalik Buterin | Disrupt SF 2017*. [Interview]. 18 September 2017.

[40] N. Bauerle, "How Could Blockchain Technology Change Finance?," 2017. [Online]. Available: https://www.coindesk.com/information/how-blockchain-technology-change-finance/. [Accessed 2 September 2018].

[41] R. Alexander, Ripple and XRP for Beginners: The Guide to the XRP-Coin and the Blockchain Technology, Independently published, 2018.

[42] S. Waterman, "Nasdaq says Estonia e-voting pilot successful," 2017. [Online]. Available: https://www.cyberscoop.com/nasdaq-estonia-evoting-pilot/. [Accessed 10 September 2018].

[43] K. Aasmae, "Why ripples from this Estonian blockchain experiment may be felt around the world," 2016. [Online]. Available: https://www.zdnet.com/article/why-ripples-from-this-estonian-blockchain-experiment-may-be-felt-around-the-world/. [Accessed 11 September 2018].

[44] B. Copigneaux, L. Probst, V. Lefebvre and J. Brown, Blockchain, Brussels: Digital Transformation Monitor, 2018.

[45] cointelegraph.com, "How Estonia Brought Blockchain Closer to Citizens: GovTech Case Studies," 2017. [Online]. Available: https://cointelegraph.com/news/how-estonia-brought-blockchain-closer-to-citizens-govtech-case-studies. [Accessed 14 September 2018].

[46] M. La Rosa , P. Loos and O. Pastor, Business Process Management, Springer, 2016.

[47] L. Schou-Zibell and N. Phair, "How secure is blockchain?," 2018. [Online]. Available: https://www.weforum.org/agenda/2018/04/how-secure-is-blockchain/. [Accessed 14 September 2018].

[48] S. Meiklejohn and C. Orlandi, "Privacy-Enhancing Overlays in Bitcoin," in *Financial Cryptography and Data Security*, Berlin, Springer, 2015, pp. 127-141.

[49] P. Koshy, D. Koshy and P. McDaniel, "An Analysis of Anonymity in Bitcoin Using P2P Network Traffic," in *Financial Cryptography and Data Security*, Berlin, Springer, 2014, pp. 469-485.

[50] S. Feld, M. Schonfeld and M. Werner, "Analyzing the Deployment of Bitcoin's P2P Network under an AS-level Perspective," in *Procedia Computer Science*, Munich, Elsevier B.V, 2014, pp. 1121-1126.

[51] A. Tar, "Proof-of-Work, Explained," 2018. [Online]. Available: https://cointelegraph.com/explained/proof-of-work-explained. [Accessed 2 November 2018].

[52] A. Beikverdi and J. Song, "Trend of centralization in Bitcoin's distributed network," Japan, IEEE, 2015.

[53] Katalyse.io, "An Analysis of the Opportunities and Threats in Blockchain Technology," 2018. [Online]. Available: https://medium.com/the-mission/an-analysis-of-the-opportunities-and-threats-in-blockchain-technology-6f55d647be3e. [Accessed 20 September 2018].

[54] C. Malmo, "Bitcoin Is Unsustainable," 2015. [Online]. Available: https://motherboard.vice.com/en_us/article/ae3p7e/bitcoin-is-unsustainable. [Accessed 22 September 2018].

[55] S. Deetman, "Bitcoin Could Consume as Much Electricity as Denmark by 2020," 2016. [Online]. Available: https://motherboard.vice.com/en_us/article/aek3za/bitcoin-could-consume-as-much-electricity-as-denmark-by-2020. [Accessed 23 September 2018].

[56] G. Pappalardo, T. Di Matteo and T. Aste, "Blockchain Inefficiency in the Bitcoin Peers Network," London, 2017.

[57] M. Leising, "The Ether Thief," 2017. [Online]. Available: https://www.bloomberg.com/features/2017-the-ether-thief/. [Accessed 17 September 2018].

[58] M. Valenta and P. Sander, "Comparison of Ethereum, Hyperledger Fabric and Corda," Frankfurt School Blockchain Center, Frankfurt, 2017.

[59] Ethereum, "Installation Instructions for Ubuntu," 2016. [Online]. Available: https://github.com/ethereum/go-ethereum/wiki/Installation-Instructions-for-Ubuntu . [Accessed 10 November 2018].

[60] S. Brown, "The C4 model for software architecture," 2018. [Online]. Available: https://c4model.com/. [Accessed 1 December 2018].

[61] P. Ramanujam, "Ethereum and Blockchain - 2," 2017. [Online]. Available: http://iotbl.blogspot.com/2017/03/ethereum-and-blockchain-2.html. [Accessed 14 December 2018].

[62] Fanvestory, "Tommy Cash music deal," 2017. [Online]. Available: https://fanvestory.com/campaign/15/rawr. [Accessed 18 December 2018].

[63] M. Palacio, "Safe Creative," 2017. [Online]. Available: http://en.safecreative.net/2017/02/22/in-5-minutes-the-european-commissions-new-copyright-directive/. [Accessed 6 June 2018].

[64] J. Yli-Huumo, D. Ko, S. Choi, S. Park and K. Smolander, "Where Is Current Research on Blockchain Technology?—A Systematic Review," PLOS ONE, West Virginia, 2016.

[65] B. Bodó, D. Gervais and J. P. Quintais , "Blockchain and smart contracts: the missing link in copyright licensing?," *International Journal of Law and Information Technology*, vol. 26, no. 4, p. 311–336, 2018.

Appendix A - Smart contracts

CMC.sol

```
pragma solidity ^0.5.0;

import './CMCEnabled.sol';


contract CMC {

    mapping(bytes32 => address) public contracts;

    function setContract(bytes32 _name, address _address) public ret
urns (bool)/* onlyOwner */{
        CMCEnabled(_address).setCMCAddress(address(this));
        contracts[_name] = _address;

        return true;
    }

    function getContract(bytes32 _name) external view returns (addre
ss) {
        return contracts[_name];
    }

}
```

CMCEnabled.sol

```
pragma solidity ^0.5.0;

contract CMCEnabled {

    address public CMCAddress;

    function setCMCAddress(address _CMC) external {
        CMCAddress = _CMC;
    }

}
```

IntellectualProperty.sol

```
pragma solidity ^0.5.0;

import './CMCEnabled.sol';
```

```
contract IntellectualProperty is CMCEnabled {

    event CreateProperty(uint64 id);
    event Transfer(uint64 id, uint64 amount, address to, address fro
m);

    struct Property {
        uint64 id;
        uint64 created;
        string title;
        string artist;
        string authors;// String arrays not supported as parameters
yet, so use custom separator in implementation
        string works;// String arrays not supported as parameters ye
t, so use custom separator in implementation
        uint64 value;// Value in money times 100, e.g. 1000EUR would
be 100000, this is the total value of all tokens
        uint64 tokens;// Percent times 10000, e.g. 33% would be 3300
00
        address root;
        mapping(address => uint64) owners;// All the owners tokens o
f the Property
        mapping(address => uint64) ownerInitialized;
        address[] ownersIndex;
    }

    // Key is the property ID
    mapping(uint64 => Property) public properties;

    function addProperty(uint64 id, uint64 created, string memory ti
tle, string memory artist, string memory authors, string memory work
s, uint64 value, uint64 tokens, address root) public returns (bool)
{
        // Check if this property does not exist
        require(properties[id].id == 0);
        // Create new empty variable
        address[] memory owners;
        // Create a new instance of a struct
        Property memory prop = Property({
            id : id,
            created : created,
            title : title,
            artist : artist,
            authors : authors,
            works : works,
            value : value,
            tokens : tokens,
            root : root,
            ownersIndex : owners
            });

        // Add new property to properties mapping
        properties[id] = prop;
        // Give all tokens to owner
        properties[id].owners[root] = tokens;
        properties[id].ownerInitialized[root] = 1;
        properties[id].ownersIndex.push(root);

        emit CreateProperty(id);

        return true;
    }

    function getProperty(uint64 _id) public view returns (uint64 id,
uint64 created, string memory title, string memory artist, string me
mory authors, string memory works, uint64 value) {
        //All mapping keys exist in Solidity so check if id field ha
s been populated on throw error if not
        require(properties[_id].id > 0);
        Property memory prop = properties[_id];

        return (prop.id, prop.created, prop.title, prop.artist, prop
.authors, prop.works, prop.value);
    }

    function getPropertyBalance(uint64 id, address account) public v
iew returns (uint64 balance) {
        require(properties[id].id > 0);

        return properties[id].owners[account];
    }

    function getPropertyValue(uint64 id) public view returns (uint64
value) {
        require(properties[id].id > 0);

        return properties[id].value;
    }

    function getPropertyOwners(uint64 id) public view returns (addre
ss[] memory owners) {
        require(properties[id].id > 0);

        return properties[id].ownersIndex;
    }

    function getRootAddress(uint64 id) public view returns (address
root) {
        require(properties[id].id > 0);

        return properties[id].root;
    }

    function getRootPercent(uint64 id) public view returns (uint64 p
ercent) {
        require(properties[id].id > 0);
        address rootAccount = properties[id].root;
        return properties[id].owners[rootAccount];
    }

    function transfer(uint64 id, uint64 amount, address to, address
from) public returns (bool) {
        require(properties[id].id > 0);
        require(properties[id].owners[from] >= amount);
        require(amount > 0);

        if (properties[id].ownerInitialized[to] <= 0) {
            properties[id].ownerInitialized[to] = 1;
            properties[id].ownersIndex.push(to);
        }

        properties[id].owners[from] -= amount;
        properties[id].owners[to] += amount;

        emit Transfer(id, amount, to, from);

        return true;
    }

}
```

Purchase.sol

```
pragma solidity ^0.5.0;

import './CMC.sol';
import './CMCEnabled.sol';
import './IntellectualProperty.sol';
import './License.sol';
import './RoyaltyDistribution.sol';

contract Purchase is CMCEnabled {

    event PurchaseAdded(uint64 id, uint64 timestamp, uint64 money, a
ddress account);

    struct PurchaseCollection {
        uint64[] purchaseSums;
        uint64[] purchaseDates;
        uint64[] purchasePercents;
        address[] purchaseAccounts;
        uint64 initialized;
    }
    // Key is the property ID
    mapping(uint64 => PurchaseCollection) public purchases;

    function addPurchase(uint64 id, uint64 timestamp, uint64 money,
address account) public returns (bool) {
        IntellectualProperty _Property = IntellectualProperty(CMC(CM
CAddress).getContract("IntellectualProperty"));
        // Calculate how much percentage will be given
        uint64 percent = (money * 1000000) / _Property.getPropertyVa
lue(id);
        uint64 rootPercent = _Property.getRootPercent(id);
        // Check if there is enough percentage available
        require(rootPercent >= percent);
        // Call transfer function
        bool result = _Property.transfer(id, percent, account, _Prop
erty.getRootAddress(id));
        initializePurchases(id);
        // Push the purchase to the collection
        purchases[id].purchaseSums.push(money);
        purchases[id].purchaseDates.push(timestamp);
        purchases[id].purchasePercents.push(percent);
        purchases[id].purchaseAccounts.push(account);

        emit PurchaseAdded(id, timestamp, money, account);

        return result;
    }

    function getPurchases(uint64 id) public view returns (uint64[] m
emory sums, uint64[] memory dates, uint64[] memory percents, address
[] memory accounts) {
        // Check if the Property exists
        require(purchases[id].initialized > 0);

        return (purchases[id].purchaseSums, purchases[id].purchaseDa
tes, purchases[id].purchasePercents, purchases[id].purchaseAccounts)
;
    }

    function initializePurchases(uint64 id) public returns (bool) {
        // Create purchases array, if it does not exist
        if (purchases[id].initialized <= 0) {
            // Create purchase entity
            uint64[] memory purchaseSums;
            uint64[] memory purchaseDates;
            uint64[] memory purchasePercents;
            address[] memory purchaseAccounts;
            // Create a new instance of a struct
            PurchaseCollection memory pc = PurchaseCollection({
                purchaseSums : purchaseSums,
                purchaseDates : purchaseDates,
                purchasePercents : purchasePercents,
                purchaseAccounts : purchaseAccounts,
                initialized : 1
                });
            purchases[id] = pc;
        }
    }
```

```
}
```

License.sol

```
pragma solidity ^0.5.0;

import './CMC.sol';
import './CMCEnabled.sol';
import './IntellectualProperty.sol';
import './RoyaltyDistribution.sol';

contract License is CMCEnabled {

    event CreateLicense(uint64 id, uint64 amount, uint64 date);

    struct LicenseCollection {
        uint256[] ids;
        uint64[] amounts;
        uint64[] dates;
        uint64 initialized;
    }

    //Key is the IntellectualProperty ID
    mapping(uint64 => LicenseCollection) private Licenses;

    function initializeLicense(uint64 id) public returns (bool) {
        if (Licenses[id].initialized <= 0) {
            // Create license entity
            uint256[] memory ids;
            uint64[] memory amounts;
            uint64[] memory dates;
            // Create a new instance of a struct
            LicenseCollection memory coll = LicenseCollection({
                ids : ids,
                amounts : amounts,
                dates : dates,
                initialized : 1
                });
            Licenses[id] = coll;
        }
    }

    function createLicense(uint64 id, uint64 amount, uint64 date) pu
blic returns (bool) {
        initializeLicense(id);

        uint256 royaltyId = Licenses[id].amounts.length + 1;
        // Push licenses to the collection
        Licenses[id].ids.push(royaltyId);
        Licenses[id].amounts.push(amount);
        Licenses[id].dates.push(date);
        RoyaltyDistribution _Dist = RoyaltyDistribution(CMC(CMCAddre
ss).getContract("RoyaltyDistribution"));
        _Dist.createDistributions(royaltyId, id, amount);

        emit CreateLicense(id, amount, date);

        return true;
    }

    function getLicenses(uint64 id) public view returns (uint256[] m
emory ids, uint64[] memory amounts, uint64[] memory dates) {
        require(Licenses[id].initialized > 0);

        return (Licenses[id].ids, Licenses[id].amounts, Licenses[id]
.dates);
    }

}
```

RoyaltyDistribution.sol

```
pragma solidity ^0.5.0;

import './CMC.sol';
import './CMCEnabled.sol';
import './IntellectualProperty.sol';

contract RoyaltyDistribution is CMCEnabled {

    struct RoyaltyDistributions {
        address[] accounts;
        uint64[] amounts;
        uint64[] percents;
        uint64 initialized;
    }

    //Key is License ID
    mapping(uint256 => RoyaltyDistributions) private royaltyDistribu
tions;

    function initializeDistributions(uint256 royaltyId) public retur
ns (bool) {
        // Create royalty distribution entity
        address[] memory accounts;
        uint64[] memory amounts;
        uint64[] memory percents;
        // Create a new instance of a struct
        RoyaltyDistributions memory dist = RoyaltyDistributions({
            accounts : accounts,
            amounts : amounts,
            percents : percents,
            initialized : 1
            });
        royaltyDistributions[royaltyId] = dist;

        return true;
    }

    function createDistributions(uint256 royaltyId, uint64 propertyI
d, uint64 amount) public returns (bool) {
        initializeDistributions(royaltyId);
        IntellectualProperty _Property = IntellectualProperty(CMC(CM
CAddress).getContract("IntellectualProperty"));
        address[] memory owners = _Property.getPropertyOwners(proper
tyId);
        // Go through all the Property owners
        for (uint64 i = 0; i < owners.length; i++) {
            // Owners account
            address ownerAccount = owners[i];
            // Owners Property share in tokens
            uint64 percent = _Property.getPropertyBalance(propertyId
, ownerAccount);
            // Calculate how much of the incoming royalty goes to th
e owner
            uint64 ownerAmount = (amount * percent) / 1000000;
            // Push royalties to the collection
            royaltyDistributions[royaltyId].accounts.push(ownerAccou
nt);
            royaltyDistributions[royaltyId].amounts.push(ownerAmount
);
            royaltyDistributions[royaltyId].percents.push(percent);
        }

        return true;
    }

    function getRoyaltyDistributions(uint256 id) public view returns
(address[] memory accounts, uint64[] memory amounts, uint64[] memory
percents) {
        require(royaltyDistributions[id].initialized > 0);

        return (royaltyDistributions[id].accounts, royaltyDistributi
ons[id].amounts, royaltyDistributions[id].percents);
    }

}
```

Appendix B - Interacting with smart contracts

gateway.js

```
var Web3 = require('web3');
var config = require('./config');
var Helper = require('./helper');

var web3 = new Web3(new Web3.providers.WebsocketProvider('ws://local
host:8545'));
var contract = new web3.eth.Contract(config.abi, config.contractAddr
ess);

module.exports = {
    rootAccount: {
        account: config.rootAccount,
        password: config.rootPassword
    },
    unlock: function (account) {
        return new Promise(function (resolve, reject) {
            try {
                web3.eth.personal.unlockAccount(account.account, acc
ount.password, 600, function (err, res) {
                    resolve(account);
                });
            } catch (err) {
                reject(err)
            }
        });
    },
    createAccount: function () {
        return new Promise(function (resolve, reject) {
            try {
                var accountData = web3.eth.accounts.create();
                resolve({
                    address: accountData.address,
                    privateKey: accountData.privateKey
                });
            } catch (err) {
                reject(err)
            }
        });
    },
    createProperty: function (data) {
        if (!data) {
            throw 'No POST data';
        }
        Helper.requireKeys(data, ['id', 'created', 'title', 'artist'
, 'authors', 'works', 'value', 'tokens']);

        var id = data.id;
        var created = data.created;
        var title = data.title;
        var artist = data.artist;
        var authors = data.authors.join('|');
        var works = data.works.join('|');
        var value = data.value * 100; //Contract requires in cents
        var tokens = data.tokens * 10000; //Contract required intege
r

        return this.unlock(this.rootAccount).then(function (acc) {
            return contract.methods.createProperty(id, created, titl
e, artist, authors, works, value, tokens).send({
                from: acc.account,
                gas: 300
            }).then(function (receipt) {
                return contract.getPastEvents('CreateProperty', {
                    filter: {id: id}
                }).then(function (events) {
                    if (events.length !== 1) {
                        throw 'Expected Purchase event list to have
length of "1", "' + events.length + '" given';
                    }

                    var eventData = events[0].returnValues;
                    return {
                        id: eventData.id
                    };
                });
            });
        });
    },
    getProperty(id) {
        return contract.methods.getProperty(id).call({gas: 100}).the
n(function (res) {
            return {
                id: res.id,
                created: res.created,
                title: res.title,
                artist: res.artist,
                authors: res.authors.split('|'),
                works: res.works.split('|'),
                value: res.value / 100
            };
        });
    },
    getBalance(id, account) {
        return contract.methods.getPropertyBalance(id, account).call
({gas: 100}).then(function (res) {
            return {
                balance: res / 10000
            };
        });
    },
    purchase(data) {
        if (!data) {
            throw 'No POST data';
        }
        Helper.requireKeys(data, ['id', 'timestamp', 'money', 'accou
nt']);

        var id = data.id;
        var timestamp = data.timestamp;
        var money = data.money * 100; // Contract requires in cents
        var account = data.account;

        return this.unlock(this.rootAccount).then(function (acc) {
            return contract.methods.purchase(id, timestamp, money, a
ccount).send({
                from: acc.account,
                gas: 300
            }).then(function (receipt) {
                return contract.getPastEvents('Purchase', {
                    id: id,
                    timestamp: timestamp,
                    money: money,
                    account: account
                }).then(function (events) {
                    if (events.length !== 1) {
                        throw 'Expected Purchase event list to have
length of "1", "' + events.length + '" given';
                    }

                    var eventData = events[0].returnValues;
                    return {
                        id: eventData.id,
                        timestamp: eventData.timestamp,
                        money: eventData.money,
                        account: eventData.account
                    };
                });
            });
        });
    },
    getPurchases: function (id) {
        return contract.methods.getPurchases(id).call({gas: 100}).th
en(function (res) {
            var purchases = [];
            for (var i in res.sums) {
                purchases.push({
                    sum: res.sums[i] / 100,
                    date: res.dates[i],
                    percent: res.percents[i] / 10000,
                    account: res.accounts[i]
                });
            }

            return purchases;
        });
    },
    transfer: function (data) {
        if (!data) {
            throw 'No POST data';
        }
        Helper.requireKeys(data, ['id', 'amount', 'to', 'from']);

        var id = data.id;
        var amount = data.amount * 10000;
        var to = data.to;
        var from = data.from;

        return this.unlock(this.rootAccount).then(function (acc) {
            return contract.methods.transfer(id, amount, to, from).s
end({
                from: acc.account,
                gas: 300
            }).then(function (receipt) {
                return contract.getPastEvents('Transfer', {
                    id: id,
                    amount: amount,
                    to: to,
                    from: from
                }).then(function (events) {
                    if (events.length !== 1) {
                        throw 'Expected Purchase event list to have
length of "1", "' + events.length + '" given';
                    }

                    var eventData = events[0].returnValues;
                    return {
                        id: eventData.id,
                        amount: eventData.amount,
                        to: eventData.to,
                        from: eventData.from
                    };
                });
            });
        });
    },
    createRoyalty: function (data) {
        if (!data) {
            throw 'No POST data';
        }
        Helper.requireKeys(data, ['id', 'amount', 'date']);

        var id = data.id;
        var amount = data.amount * 100;
        var date = data.date;

        return this.unlock(this.rootAccount).then(function (acc) {
            return contract.methods.createRoyalty(id, amount, date).
send({
                from: acc.account,
                gas: 300
            }).then(function (receipt) {
                return contract.getPastEvents('CreateRoyalty', {
                    id: id,
                    amount: amount,
                    to: date
                }).then(function (events) {
                    if (events.length !== 1) {
                        throw 'Expected Purchase event list to have 
length of "1", "' + events.length + '" given';
                    }

                    var eventData = events[0].returnValues;
                    return {
                        id: eventData.id,
                        amount: eventData.amount,
                        to: eventData.date
                    };
                });
            });
        });
    },
    getRoyalties: function (id) {
        return contract.methods.getRoyalties(id).call({gas: 100}).th
en(function (res) {
            var royalties = [];
            for (var i in res.ids) {
                royalties.push({
                    id: res.ids[i],
                    amount: res.amounts[i] / 100,
                    date: res.dates[i]
                });
            }

            return royalties;
        });
    },
    getRoyaltyDistributions: function (id) {
        return contract.methods.getRoyaltyDistributions(id).call({ga
s: 100}).then(function (res) {
            var dist = [];
            for (var i in res.accounts) {
                dist.push({
                    account: res.accounts[i],
                    amount: res.amounts[i] / 100,
                    percent: res.percents[i] / 10000
                });
            }

            return dist;
        });
    }
```

```
};
```

server.js

```
var http = require('http');
var url = require('url');
var Helper = require('./helper');
var Gateway = require('./gateway');
var Logger = require('./logger');

http.createServer(function (req, res) {
    var json = {
        status: 'success',
        data: {}
    };

    try {
        var parts = url.parse(req.url, true);
        var path = parts.pathname;
        var pathParts = Helper.trimChar(path, '/').split('/');

        Logger.log('Incoming "' + req.method + '" request to "' + pa
th + '"');
        var handler = null;
        if (path.indexOf('/user/create') === 0 && req.method === 'PO
ST') {
            handler = Gateway.createAccount();
        } else if (path.indexOf('/property/create') === 0 && req.met
hod === 'POST') {
            handler = Helper.handlePost(req).then(function (post) {
                return Gateway.createProperty(post);
            });
        } else if (path.indexOf('/property/get') === 0 && req.method
=== 'GET') {
            var id = pathParts[2];
            handler = Gateway.getProperty(id);
        } else if (path.indexOf('/property/balance') === 0 && req.me
thod === 'GET') {
            var id = pathParts[2];
            var account = pathParts[3];
            handler = Gateway.getBalance(id, account);
        } else if (path.indexOf('/property/purchase') === 0 && req.m
ethod === 'POST') {
            handler = Helper.handlePost(req).then(function (post) {
                return Gateway.purchase(post);
            });
        } else if (path.indexOf('/property/purchases') === 0 && req.
method === 'GET') {
            var id = pathParts[2];
            handler = Gateway.getPurchases(id);
        } else if (path.indexOf('/property/transfer') === 0 && req.m
ethod === 'POST') {
            handler = Helper.handlePost(req).then(function (post) {
                return Gateway.transfer(post);
            });
        } else if (path.indexOf('/royalty/create') === 0 && req.meth
od === 'POST') {
            handler = Helper.handlePost(req).then(function (post) {
                return Gateway.createRoyalty(post);
            });
        } else if (path.indexOf('/royalty/get') === 0 && req.method
=== 'GET') {
            var id = pathParts[2];
            handler = Gateway.getRoyalties(id);
        } else if (path.indexOf('/royalty/distribution/get') === 0 &
& req.method === 'GET') {
            var id = pathParts[3];
            handler = Gateway.getRoyaltyDistributions(id);
        } else {
            throw 'Invalid url';
        }

        handler.then(function (data) {
            json.data = data;
            return Helper.sendJson(res, json);
        }).catch(function (err) {
            json.status = 'error';
            json.data.error = err.toString();
            Helper.sendJson(res, json);
        });
    } catch (err) {
        json.status = 'error';
        json.data.error = err.toString();
        Helper.sendJson(res, json);
    }

}).listen(80);
```

helper.js

```
var Logger = require('./logger');

module.exports = {
    handlePost: function (req) {
        return new Promise(function (resolve, reject) {
            var input = '';
            req.on('data', function (data) {
                try {
                    input += data;
                    if (input.length > 1e6) {
                        req.connection.destory();
                        throw 'Flood detected';
                    }
                } catch (err) {
                    reject(err.toString());
                }
            });

            req.on('end', function () {
                try {
                    Logger.log('Got JSON input ' + input);
                    var json = (input) ? JSON.parse(input) : null;
                    resolve(json);
                } catch (err) {
                    reject(err.toString());
                }
            });
        });
    },

    sendJson: function (res, jsonData) {
        var json = JSON.stringify(jsonData);
        Logger.log('Sending JSON output ' + json);
        var buffer = new Buffer(json);
        res.writeHead(200, {
            'Content-Type': 'application/json',
            'Content-Length': buffer.length
        });
        res.end(buffer);
    },

    requireKeys: function (data, keys) {
        for (var i in keys) {
            if (!data[keys[i]]) {
                throw 'No "' + keys[i] + '"';
            }
        }
    },

    trimChar: function (string, charToRemove) {
        while (string.charAt(0) === charToRemove) {
            string = string.substring(1);
        }

        while (string.charAt(string.length - 1) === charToRemove) {
            string = string.substring(0, string.length - 1);
        }

        return string;
    }
};
```

Appendix C - Contract migrations

01-init.js

```
var MigrationInit = {
    CMC: function (cb) {
        var cmcContract = web3.eth.contract([
            // Omitted for readability
        ]);
        var cmc = cmcContract.new(
            {
                from: web3.eth.accounts[0],
                data: '', // Omitted for readability
                gas: '300'
            }, function (e, contract) {
                if (typeof contract.address !== 'undefined') {
                    cb(contract.address, cmc);
                }
            });
    },
    IntellectualProperty: function (cb) {
        var intellectualpropertyContract = web3.eth.contract();
        var intellectualproperty = intellectualpropertyContract.new(
            {
                from: web3.eth.accounts[0],
                // Omitted for readability
                data: '',
                gas: '300'
            }, function (e, contract) {
                if (typeof contract.address !== 'undefined') {
                    cb(contract.address);
                }
            });
    },
    License: function (cb) {
        var licenseContract = web3.eth.contract([
            // Omitted for readability
        ]);
        var license = licenseContract.new(
            {
                from: web3.eth.accounts[0],
                data: '', // Omitted for readability
                gas: '300'
            }, function (e, contract) {
                if (typeof contract.address !== 'undefined') {
                    cb(contract.address);
                }
            });
    },
    Purchase: function (cb) {
        var purchaseContract = web3.eth.contract([
            // Omitted for readability
        ]);
        var purchase = purchaseContract.new(
            {
                from: web3.eth.accounts[0],
                data: '', // Omitted for readability
                gas: '300'
            }, function (e, contract) {
                if (typeof contract.address !== 'undefined') {
                    cb(contract.address);
                }
            });
    },
    RoyaltyDistribution: function (cb) {
        var royaltydistributionContract = web3.eth.contract([
            // Omitted for readability
        ]);
        var royaltydistribution = royaltydistributionContract.new(
            {
                from: web3.eth.accounts[0],
                data: '', // Omitted for readability
                gas: '300'
            }, function (e, contract) {
                if (typeof contract.address !== 'undefined') {
                    cb(contract.address);
                }
            });
    }
};

personal.unlockAccount('0x00e36d234e499449e9f5f06604305dcf483d6dee',
'pass', 600);

var output = {};
MigrationInit.CMC(function (cmcAddress, cmc) {
    output.CMC = cmcAddress;
    MigrationInit.IntellectualProperty(function (ipAddress) {
        output.IntellectualProperty = ipAddress;
        MigrationInit.License(function (licenceAddress) {
            output.License = licenceAddress;
            MigrationInit.Purchase(function (purchaseAddress) {
                output.Purchase = purchaseAddress;
                MigrationInit.RoyaltyDistribution(function (royaltyA
ddress) {
                    output.RoyaltyDistribution = royaltyAddress;
                    console.log(JSON.stringify(output));
                });
            });
        });
    });
});
```

02-cmc.js

```
var Web3 = require('web3');

var abi = [
    // Omitted for readability
];

/*****
 * ASCII hex encoded values table for debugging. Acutal conversion i
s done below with web3.utils.fromAscii
 var mapping = {
    IntellectualProperty: 0x496e74656c6c65637475616c50726f7065727479
,
    License: 0x4c6963656e7365,
    Purchase: 0x5075726368617365,
    RoyaltyDistribution: 0x526f79616c7479446973747269627574696f6e
};
 */
var config = {
    "CMC": "0xc3478511d4842fa675a3d19610d1825e1f976c2f",
    "IntellectualProperty": "0xd658aa3fdf20c1a85f7cda738e7488ce1bd68
d1e",
    "License": "0x8de5100ca2127e0e97cf157c33f0f5dbe4607d92",
    "Purchase": "0x45893c6fc3cdbcf86b86b5d2e9b87f2e2ada2de1",
    "RoyaltyDistribution": "0x079dc1e79f069a806d9bfd8dfafb49b13c9556
97"
};

var web3 = new Web3(new Web3.providers.WebsocketProvider('ws://local
host:8545'));
var contract = new web3.eth.Contract(abi, config.CMC);
var account = '0x00e36d234e499449e9f5f06604305dcf483d6dee';

web3.eth.personal.unlockAccount(account, 'pass', 600, function (err,
res) {

    contract.methods.setContract(web3.utils.fromAscii('IntellectualP
roperty'), config.IntellectualProperty).send({
        from: account,
        gas: 300
    });

    contract.methods.setContract(web3.utils.fromAscii('License'), co
nfig.License).send({
        from: account,
        gas: 300
    });

    contract.methods.setContract(web3.utils.fromAscii('Purchase'), c
onfig.Purchase).send({
        from: account,
        gas: 300
    });

    contract.methods.setContract(web3.utils.fromAscii('RoyaltyDistri
bution'), config.RoyaltyDistribution).send({
        from: account,
        gas: 300
    });

});
```