Music Copyright Management using Smart Contracts and Tokenization on the Ethereum Blockchain

Ruth Brennan

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

Supervisor: Dr Donal O’Mahony

August 2022
Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that, unless otherwise stated, is my own work.

________________________________________
Ruth Brennan

April 17, 2022
Music Copyright Management using Smart Contracts and Tokenization on the Ethereum Blockchain

Ruth Brennan, Master of Science in Computer Science
The University of Dublin, Trinity College, 2021

Supervisor: Dr Donal O’Mahony

Abstract
An increase in transparency and speed could improve the systems in place today for music copyright management and royalty payments in order to return the control of music copyright to its creators. The ownership of the copyright for a song is distributed between a group of people. There is no one specific global authority on the distribution of music copyright ownership. It instead exists in separate databases maintained and protected by individual companies or countries. This, along with a convoluted payment chain, leads to significant delays in the payment of royalties to artists, composers and other song copyright owners. With its inherent transparency, security and decentralisation, Blockchain is proposed to target these issues in music copyright management. This paper details a proof-of-concept copyright management, royalty payment and copyright marketplace decentralised application (DApp) built on the Polygon Blockchain. Tokenisation is used to represent the shares of ownership in a song’s copyright. Tokens are developed using the ERC1155 multi-token standard, which enables all tokens relating to individual copyright to be governed by the state of a single smart contract. This solution presents a novel utilisation of lazy minting to facilitate share initialisation and a forward-compatible secondary fee implementation following the ERC-2981 royalty standard. The application presents a royalty payment distribution through a pull payment method and utilises events for informative integration with blockchain explorers like Polyscan and wallet software like Metamask. Emphasis is placed on the implementation security, user focus and transparency of the system.
Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr Donal O’Mahony, for his unwavering support, extensive guidance and constant encouragement throughout my research.

I would like to thank my fellow classmates in Integrated Computer Science for their shared advice, support, and dedication during our college careers together, especially this last year.

Finally, I would like to acknowledge my appreciation for my family, my parents and my brothers, for their constant encouragement, help and guidance throughout all my life’s endeavours.

Ruth Brennan

University of Dublin, Trinity College
August 2022
# Contents

Abstract

Acknowledgements

1 Introduction

1.1 Motivation

1.2 Research Objectives

1.3 Use Cases

1.4 Technical Approach

2 Blockchain

2.1 Introduction

2.1.1 Consensus

2.1.2 Blockchain Classification

2.1.3 Smart Contracts

2.2 Ethereum

2.2.1 Components of Ethereum

2.2.2 Addresses and Key Pairs

2.2.3 Ethereum Virtual Machine (EVM)

2.2.4 External Owned Accounts (EOA) and Contract Accounts

2.2.5 Wallets

2.2.6 GAS

2.3 Decentralized Applications (DApps)

2.3.2 DApp Front-Ends

2.4 Decentralized Storage

2.4.1 Inter-Planetary File System (IPFS)

2.5 Tokens

2.5.1 Token Types
4.3.4 Lazy Minting

4.4 Polygon
  4.4.1 Deployment
  4.4.2 Bridges

5 Implementation
  5.1 Application Contract
  5.2 Token Contract
  5.3 DApp
  5.4 Token Metadata
    5.4.2 Metadata
    5.4.2 Metadata Protection
    5.4.3 Encryption & Key Protection Store
    5.4.4 Audio Steganography
  5.5 Interface
    5.5.1 Contract Creation Screen
    5.5.2 Voucher Redemption Screen
    5.5.3 Token Management Screen
    5.5.4 Token Marketplace Screen
    5.5.5 Contract Interaction Screen
    5.5.6 Polyscan

6 Security Evaluation
  6.1 Underflow/Overflow
  6.2 Modifiers
  6.3 Stack Depth
  6.4 Race Conditions/Front Running
  6.5 Reentrancy
  6.6 Denial of Service Attack (DOS)
List of Figures

1.1: Music Royalty Payment Chain Example ........................................ 2
2.1: Private Key, Public Key, Address Relationship ........................... 13
2.2 ERC Creation Process .................................................................... 20
2.3 Polygon Architecture .................................................................... 23
3.1 Music Rights .................................................................................. 25
4.1 System Framework ......................................................................... 31
4.2 DAPP Development Process Flow ...................................................... 37
4.3 Proposed Application Architecture .................................................. 33
4.4 Lazy Minting Workflow .................................................................... 40
5.1 Contract Creation Screen ................................................................. 49
5.2 Voucher Signing Pop-up ................................................................. 49
5.3 Voucher Redemption Screen ............................................................ 50
5.4 Owned Token Management Screen .................................................. 50
5.5 Token Marketplace with connected account screen ....................... 51
5.6 Contract Interaction Screen ............................................................. 51
5.7 Polyscan Contract View - Token Transactions ............................... 52
5.8 Polyscan Contract View - Contract Transactions ............................ 52
1 Introduction

This section introduces the motivation for the dissertation work. The background to the work is explored, the research objectives are defined, and the primary use cases are detailed. Finally, the section outlines the technical approach to implementing the planned work.

1.1 Motivation

The popularity of the internet has accelerated the circulation of digital resources, especially music. With innovative efforts to help artists bring music to fans in new ways, recorded music revenues grew globally for the sixth consecutive year, driven primarily by paid subscription streaming [84]. As artists continue to expand their geographical footprint and cultural reach, music has become more globally connected than ever before. The problem of managing the copyright of digital resources, like music, has become more prominent. Unfortunately, the connectivity, speed and easy access to music from a consumer side are not reflected in the music payment supply chain.

Money earned from music is ruled by copyright law, and a song’s copyright ownership is usually divided between a group. There is no central database of song copyright ownership, and finding ownership information usually requires requesting information from individual companies, who maintain their own databases. This can lead to massive delays in payments of royalties [66]. This lack of accumulated information along with the propagation of payments through a convoluted payment chain, with delays dependent on company accounting cycles, leads to considerable delays in cash flow for artists. For example, it can take, on average, 3-9 months for an artist to be paid for a single stream [71].
Blockchain has recently been applied to various fields such as finance, distribution, and public services beyond the category of cryptocurrency. The introduction of blockchain technology has a breakthrough meaning for the solution of music copyright management.

Smart contracts allow royalties generated from things like streams on streaming platforms, which have blanket licences for streaming music prenegotiated, to pay an artist immediately after a song is streamed by eliminating intermediaries. This will be done by using blockchain as a database for music copyright ownership by way of tokenization of shares in copyright. This quick revenue stream is essential for the prosperity of artists, especially in smaller music markets. This constant cash flow allows artists more choice over the management of their music. Due to a lack of cash flow artists often find themselves selling shares in their copyright in exchange for cash advances, used to market, tour or distribute music, that heavily favours the businesses that offer them [72]. Even if the cash flow is not significant enough to completely replace these cash advances, blockchain offers artists more freedom in how they sell shares in their music.

Since transaction history is available to all and easy to navigate using block explorers, artists can see how much others, in a similar situation, were able to profit from the sale of shares or royalties. This encourages honest dealings and with blockchains, a peer-to-peer system. This gives the power to the artist by alleviating the need for a middleman. This use of blockchain to solve issues of transparency is
especially prevalent in the value chain, one for valuation in sales of copyright, but, two, it is also relevant in informing licensing negotiations.

1.2 Research Objectives

The management of music copyright needs to be reformed to bring the ownership, control and profit from music to its creators. Such reform should enable music creators, artists and composers to decide how copyright ownership should be distributed and allow each owner autonomy over the sale of their ownership. Furthermore, the negotiations of licenses and their accompanying royalties would benefit from transparency regarding past valuations of music and past licensing agreements.

Ethereum is a technology home to digital money, global payments, and applications. The community has built a booming digital economy, bold new ways for creators to earn online, and so much more. A core mission for all decentralised blockchain technology, including Ethereum, is to provide a trustless network to build platforms, DApps, that disrupt business models or invent new ones [17].

Thus, this dissertation proposes a system that utilises the Ethereum blockchain and tokenisation and builds the decentralised application to create a music copyright management solution. Such a solution, in pursuit of this goal, should conform to the following requirements:

1. Transparency

In order to encourage fair dealings and enforce accountability in profits made from music copyright, transparency should be at the forefront of the design process. This would include transparency with the valuation of music (through historical royalty payments), valuation of copyright ownership (through the historical sale of ownership shares) and when payments are made and distributed to ensure all owners are being paid fairly and in the correct proportion.

2. Data Security and Integrity

Music copyright information and the audio to which it relates should be stored in a secure and immutable manner, to prevent data compromise.

3. Interface Useability

Interfacing with the system as a whole should not require prior knowledge of the underlying blockchain technology, token smart contract procedures or decentralized storage mechanism.
4. Ownership Tracking and Trading

There should be ease of tracking the ownership of music copyright between groups of people and allow each of those owners autonomy over their portion of the copyright. This autonomy should include the ability to sell and transfer ownership of those shares as a means of profiting from them.

5. Royalty Payment and Distribution

The system should include a royalty payment mechanism that will record the payment and the licensing agreement it relates to transparently, and history should be viewable by any interested party. These royalty payments should be quickly and efficiently distributed to all parties who own shares, in the copyright related to the license, at the time of payment.

6. Self-Sovereignty

The core functionality of the system and storage of data should be available to the user in a fully decentralized manner so that control of the copyright is entirely in the hands of its owners.

1.3 Use Cases

The proposed music copyright management system provides means to track copyright ownership, make and distribute royalty payments and sell shares in ownership of copyrights. Example use cases for the platform are given below for context.

Users identify themselves to the system by connecting their wallets. The system interacts with the wallet software to facilitate transaction signing, payments and verification.

- Users can introduce a new song's copyright to the system by deploying a new contract. The system asks for information about the details of the copyright, including owners, share distribution and the audio file that the copyright pertains to. The system deploys this contract to the blockchain network and facilitates the tokenisation of shares in the copyright, for each of the owners, by creating vouchers. The owners, via the system, can redeem these vouchers to mint tokens. This actualises the tokenisation of their shares, as they are permanently represented by tokens on the blockchain. Once the token has been established on the blockchain, the user can access the other use cases for that particular copyright.
The system facilitates a marketplace for the sale of tokens. It is possible also to use other popular decentralized token marketplaces, e.g. OpenSea or Rarible, for these sales. The token is listed as for sale at a special price. Once the token is purchased, ownership of the token and all ownership functionality is transferred to the purchaser. The owner maintains all ownership privileges until the purchase and transfer are confirmed.

The system allows royalty payments to be made following pre-negotiated licensing agreements. These payments are recorded on the blockchain, and payments are distributed to the copyright owners per their percentage of ownership.

1.4 Technical Approach

Ethereum enables application developers to upload programs into the Ethereum Virtual Machine's (EVM) state, and users make requests to execute these code snippets with varying parameters. We call the programs uploaded to and executed by the network smart contracts. By nature, these contracts deployed to the blockchain are immutable, and all interactions between users and the contracts are verified and appended to Ethereum's shared public ledger of transactions. Tokens on Ethereum are backed by smart contracts that govern their state and actions. The application state is stored in a separate smart contract that will optionally facilitate users’ interactions with their tokens and sales and additional optional features and security measures that bolster the purpose of the system. The application smart contract acts as the backend for the system. This makes execution transparent and verifiable as the decentralised network supports it.

A system user introduces a new song's copyright by deploying a new contract that will govern all tokens representing shares of the copyright. The role of the contract owner will be distinct from that of the token owner and will be reserved for the original creators of the music. Contract owners and token owners will have distinct privileges that will not interfere to ensure the sovereignty of token ownership. These privileges will be implemented and enforced by access roles and modifiers within the smart contract. They can only be changed upon token minting or ownership transfer dictated by the contract implementation. The generation and deployment of the smart contracts are done in the client, using a JavaScript framework for interfacing with Ethereum, known as Web3.js.

Copyright metadata and the FLAC audio file representing the song the copyright relates to are stored on the decentralised storage platform known as the InterPlanetary File System (IPFS), which allows for the data to be stored in a redundant and immutable manner.
The token contract governs the core of the tokenisation of the shares and this does not need to be interacted with through the application contract. The application contract provides a foundation for system principles on top of which a user interface for the overall system is built. As the token and application contracts make up the system's core, the client-side can be seen as a complementary means of providing interface usability. On top of this, it also provides complimentary features to enhance a user's experience. It provides a means for token metadata protection features as a bonus to the overall goal.
2 Blockchain

2.1 Introduction

Distributed Ledger Technology (DLT) is the decentralisation of services through peer-to-peer networks (commonly known as Blockchain technology).

Blockchain networks have recently gained popularity for providing immutable ledgers distributively. This technology was originally adopted as the backbone for a public distributed ledger system that would process asset transactions as digital tokens between (P2P) Peer-To-Peer users for the cryptocurrency BitCoin [1]. Blockchain networks are distinguished by their fundamental characteristics of disintermediation, trustless execution, public accessibility (data transparency) and tamper resilience [2]. This makes them an advisable backend for projects seeking to emphasise traceability, data security and integrity, for example, cryptocurrencies [3].

It is sensible to interpret the term “blockchain networks” on two levels. First, understanding that blockchains refer to a framework of immutable data organisation, and secondly, the blockchain networks on top of which we define data deployment and maintenance [2]. In terms of data organisation, blockchain technologies use a number of existing and available cryptographic techniques [4],[5]. These techniques enable blockchains to cryptographically associate a users’ on-chain identities with the transactions of their tokenised assets. This means that blockchains can prove authentication for asset transfer and asset ownership. Furthermore, a blockchain maintains transactional records by cryptographically chaining record subsets in the form of data “blocks” to their chronological predecessors [6].

Due to the cryptographic references, attempts at data tampering are immediately detected. From a network organisation perspective, an attempt at replicating an agreement in the transaction history of an open-access, weakly synchronised network among trustless nodes is combated by blockchain consensus protocols. Consensus protocols offer the ability to achieve agreement on the global blockchain data state among many trustless nodes without the need to define identity authentication and with low messaging overhead [8]. We can interpret the blockchain as a universal memory of the blockchain network and view the blockchain network as a distributed virtual machine shaped by every node therein [2].
2.1.1 Consensus

In order to achieve consensus, blockchain networks have chosen to adopt an incentive-based block creation process, block mining [8]. This key feature provides fault tolerance and censorship resistance to the blockchains. The most common consensus algorithm to date (at the time of writing) is Proof-of-Work (PoW), used by BitCoin and Ethereum.

Proof of Work (PoW)

This protocol is founded on the concept that the block with the most “work” is accepted first. This “work” is performed by finding a nonce that generates a particular sequence when hashed with the block data. The only way to find this nonce is by exhaustive search. Due to the one-way nature of hash functions, this makes faking work infeasible. Verification of “work” is done in a single operation that checks if it matches the particular sequence, validating that the block can be trusted. This method prevents the writing of fraudulent blocks to the DLT as a malicious actor(s) would have to generate more work than 51% of the network combined to generate longer chains than the majority of the network [9].

Initially, this method was designed to fight against denial of service attacks and other service abuse such as spam on a network. PoW is also currently the only mechanism tried and tested astonishingly successfully against collusion attacks. A successful attack would require a lot of power and time to work out [12]. Proof-of-Work involves serious trade-offs in terms of its heavy power consumption for the generation of work and, recently, computing power centralization through mining pools. A solution to these issues can be found in alternatives such as Proof of Stake (PoS).

Proof of Stake (PoS)

This protocol is similar to Proof of Work in that they are both virtual mining mechanisms and lottery-based consensus. Mining is the process of adding transactions to the distributed ledger. In Proof of Stake, the value of the miner’s stake determines the likelihood of them discovering the next addition [10]. In comparison to Proof of Work, Proof of Stake participants are stakeholders and thus have no benefit to tamper with the system. To reach consensus in PoS is very like in PoW. The fork with the highest stake is chosen for the main ledger.

Initially, these mechanisms were implemented in the Blockchain behind the PeerCoin cryptocurrency in 2012 [11]. The stake used in this instance was coinage, which refers to the age of the coin at stake, that is to say, the time since the coin was the last spent or used.
The simplest form of stake that can be used for a cryptocurrency blockchain is the coin itself. The participants show that they own a portion of the sum and thus have a stake in the currency. In both of these cases, the difficulty of mining would be inversely proportional to the coinage or the number of coins [10]. Overall, Proof of Stake is a fundamentally faster alternative to Proof of Work. It is also relatively costless and very energy saving in comparison. The law of commons is an issue in PoW in which the network's security may decrease over time. This is due to the fact that reward may, over time, become easier to acquire and thus reduces the incentive to mine. In PoS, however, for example, coin, a user possesses a higher incentive to protect the system.

We previously discussed the majority attack in the context of PoW. In PoS, a node with over 50% can mount an attack [11]. If we look at this regarding cryptocurrency, we see that someone with over half the stake in that currency could mount an attack. However, they would reduce trust in the entire currency and lead to depreciation in the currency which will ultimately negatively affect the attacker.

Another area of concern conquered by PoS is a monopoly. Noticeable in PoW, those with access to expensive supercomputers and cheap electricity have a massive advantage as miners. In PoS, especially with coinage employed, mitigates this well. A poor miner can hold onto their coin for a relatively long time to increase their probability of discovering a new addition. This act results in an even distribution of power and means every node in proof of stake can validate transactions which boost network security [10].

2.1.2 Blockchain Classification

Blockchains can be categorised from two perspectives: generations and access control. As the first mainstream blockchain, Bitcoin carries the generation 1 title [13] as do other purely cryptocurrency-focused platforms following the same pattern. Ethereum, proposed in 2013, started a new approach. It added a layer of abstraction between the execution code and the distributed ledger. Approaches like this have been deemed generation 2 [14]. There are two approaches to access control classification; access to the network and node identity. In platforms like Ethereum, anyone can join the network without restriction. Private networks restrict user participation. For identity access control on permissioned blockchains, the identity of the nodes is known. For permissionless blockchains, the nodes are anonymous or pseudo-anonymous [13].
2.1.3 Smart Contracts

The concept of smart contracts was originally created by Nick Szabo [15]. The idea behind a smart contract is to embed contractual clauses into the software. It was introduced to the blockchain by Ethereum [12]. Smart contracts on a blockchain are immutable pieces of code residing inside a block's data. Transactions can trigger the code, and once the other nodes in the network validate the output, the results are saved to the blockchain. Like any data in the blockchain, this code is censor resistant and difficult to stop without altering most of the nodes in the network [16].

2.2 Ethereum

Ethereum is a public, open-source platform based on blockchain technology. Ethereum can be viewed as a world computer built on a peer-to-peer (P2P) network. It allows trusted and decentralised applications to run on top of Ethereum and avoid centralised authority and the single-point-of-failure problem [17]. Staying with the comparison of the Ethereum network as a singular world computer, it follows that computational resources, such as CPU and memory, are limited and can be strained under the pressure of a large scale user base and DApps. Therefore, it is logical that using the Ethereum world computer machine will cost money. This cost is paid in the form of cryptocurrency. The official definition of Ethereum, as found on Ethereum.org, states that:

‘Ethereum is a decentralised platform that runs smart contracts: applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third-party interference. These apps run on a custom-built blockchain, an enormously powerful shared global infrastructure that can move value around and represent the ownership of property.’

As stated above, Ethereum is classified as a generation 2 blockchain. The most recognisable blockchain of Generation 1 is the Bitcoin blockchain. It tracks the state of units of bitcoin and their ownership. Bitcoin can be thought of as a distributed consensus state machine, where transactions cause a global state transaction and alter the ownership of coins [6]. The rules of consensus, of course, constrain these state transactions, allowing the users to cover a common state of the system after several blocks are mined.

Comparative to Bitcoin, Ethereum is an open-source and distributed state machine on the foundation of blockchain technology, and it brings the full potential of blockchain technology to our attention. Instead of tracking only the state of currency ownership, it also tracks general-purpose data [6]. Ethereum has a memory that stores both data and code. The blockchain tracks how this memory changes over time. Generally speaking, Ethereum can be described as a state
machine based on transitions. Where the transaction formula would be defined as follows:

\[ \sigma' = \gamma(\sigma, T) \] [18]

Where the initial state is described by \( \sigma \), the transformed state by \( \sigma' \), \( T \) for transaction, \( \gamma \) as the transformation function.

2.2.1 Components of Ethereum

Ethereum boasts all the features of a public blockchain, including public/private key encryption, cryptographic hash function and Merkle trees. In this section, the paper introduces and defines components of Ethereum's blockchain before continuing to discuss the most relevant components in more detail.

- **P2P network**
  The Ethereum main network is addressable on TCP port 30303 and runs a protocol called DEVP2P [22].

- **Node**
  If a user is running a node, they can access on-chain data through the node. A full node would download the whole blockchain, and comparatively, a light node would not download all the data and connect to a full node to download desired data from it [17]. A full node would be a computer running all necessary software, including a full distributed ledger and P2P routing software.

- **Miner**
  A Miner runs a node mining the network and processing the blocks on the blockchain. Usually, the miner maintains a full node running professional mining software. However, not all full nodes are mining nodes. If there is a code upgrade, the mining node needs to update its software with up-to-date code. This can be done through a soft fork, a backward compatibility implementation. Although it can be done through a hard fork, hard forked code is not compatible with old code [17].

- **Transactions**
  Transactions on Ethereum are network messages that include a sender, recipient, value and data payload. Transactions are the starting point of all activity in the Ethereum system. Transactions are the "inputs" that cause the Ethereum Virtual Machine to evaluate contracts, update balances, and more generally modify the state of the Ethereum blockchain.

- **State machine**
All state transitions on the Ethereum network are processed by a stack-based virtual machine called the Ethereum Virtual Machine (EVM) [20]. This is expanded upon in section 2.2.3.

Data structures
The state of Ethereum is stored locally by each node in the network as a database that contains transactions history and system state in a serialised hashed data structure, a Merkle Patricia tree [4].

Consensus algorithm
Nakamoto Consensus utilises a sequence of single signature blocks, weighted by importance according to PoW, to determine the longest chain and, therefore, the current state [12]. Ethereum, at the time of writing, weights its blocks by PoW, specifically Ethash. Ethereum does have plans to migrate to a PoS weighted voting system in the near future, according to Ethereum.org.

Ether Currency Units
Ethereum's native cryptocurrency is named Ether. It is identifiable by “ETH” or with the symbols Ξ and, less often, ♦. Ether is subdivided into smaller units, the smallest of which is Wei. One Ether equates to 1 quintillion Wei (1 * 1018 or 1,000,000,000,000,000,000).

2.2.2 Addresses and Key Pairs
For symmetric cryptography, a key needs to be shared between parties prior to information exchange. This key exchange is not necessary for the implementation of asymmetric cryptography. As the term 'asymmetric' would suggest, the encryption and decryption keys are separate and are termed the public and private keys. They are considered a "pair" because the public key is derived from the private key, and the public key is calculated from the private key using elliptic curve multiplication. In normal circumstances, the public key is widely available, and the private key is kept secret by an individual party and is never used directly in the system. They are never transmitted or stored on Ethereum [22]. Usually, they are managed and stored in an encrypted form by Ethereum Wallet software. Only account addresses and digital signatures are ever transmitted and stored on the Ethereum system. Due to the separate nature of the keys, encryption using this method can be used in unsafe channels [19].

The key pair represents an Ethereum account by providing a publicly accessible account handle, the address, and authentication for interactions with smart contracts. Using the one-way hash function, Keccak-256, unique identifiers, Ethereum addresses, can be derived from the public key. Ethereum addresses are
hexadecimal numbers consisting of the last 20 bytes of the Keccak-256 hash of the public key [12].

It is helpful to think of the private key as a pin for a bank account. It controls access to the unique piece of information needed to create a digital signature. If you lose the private key, you lose access to the data it protects.

Elliptic curve cryptography provides the method for a message (i.e. the transaction details) to be combined with the private key to create a code, only be produced with knowledge of the private key, called a digital signature [22]. Anyone can verify the authenticity of the digital signature by checking that it matches the transaction details and the Ethereum address to which access is requested. This verification resolves without a shadow of a doubt that the transaction came from someone with access to the private key, as it corresponds to the public key behind the Ethereum address.

Digital Signatures are needed to sign transactions and give permission to spend any funds linked to the account because they can be used to authenticate users. Digital signatures are also used to authenticate the owners of contracts.

2.2.3 Ethereum Virtual Machine (EVM)

The Ethereum Virtual Machine (EVM), which was previously referred to as the Ethereum computer, carries out commands. The EVM validates the command the same way Bob validated the message. If the EVM determines that the command came from Alice, it will execute, and the token smart contract will be updated to reflect the transaction. In other words, Ethereum state transitions are processed by the Ethereum Virtual Machine (EVM), a stack-based virtual machine that executes bytecode (machine-language instructions). EVM provides a powerful and Turing-complete programming language. Every opcode executed on EVM will be run on each node in the Ethereum network. Turing-completeness means that a computer can solve any mathematics formula assuming that the algorithm is correct if we have enough time and memory.

EVM programs, called "smart contracts," are written in high-level languages (e.g., Solidity) and compiled to bytecode for execution on the EVM [20]. Every node in the Ethereum network runs a local copy of the EVM to validate contract execution. The
Ethereum blockchain records this "world computer"'s changing state as it processes transactions and smart contracts [21].

2.2.4 External Owned Accounts (EOA) and Contract Accounts

Externally owned accounts have a private key [22]. The private key means that they have control over access to their funds or contracts. This is opposed to a contract account, which has smart contract code, which an EOA does not have. A contract account does not possess a private key. Instead, it is owned and thus controlled by the code of its' smart contract [22]. This smart contract is a software program that has been recorded on the Ethereum blockchain when the contract was created and is executable by the EVM.

Contracts have addresses like EOAs, as discussed in the previous section. This means that they can send and receive Ether. Ether is the cryptocurrency native to Ethereum. When Ether is transferred or received to or by a contract, this transaction containing the contract address triggers the contract to run in the EVM, using the transaction's data as input. This data can point to a specific function and what parameters this function should be passed. In this way, it is apt to describe a transaction as calling a function in a smart contract.

Since a contract account does not own a private key, it can not initiate transactions, and this can only be done by EOAs [22]. A contract transaction occurs as a reaction to an EOA or by a call from another contract, allowing the building of complex execution paths. An example of an execution path could be; an EOA sending a request translation to a multi-signature smart contract wallet to send ETH to another address. Another example could be a DApp having contract A call contract B to maintain a shared state across users of Contract A.

2.2.5 Wallets

Concerning Ethereum, the term wallet refers to a software application that serves as a primary user interface to Ethereum. It controls access to a user's funds by managing their keys and addresses, tracking their balance and creating and signing transactions [23]. Some wallets can also interact with contracts, for example, a token contract. Every wallet has a key-management mechanism as a fundamental component. Some wallets are capable of interfacing with Ethereum based decentralized applications (DApps). One such example is Metamask. Metamask is a browser extension and can connect to various Ethereum nodes and test blockchains. MetaMask keeps a list of the most popular token standards and automatically detects ownership of those and displays them. For less common tokens, they must be added manually [47].
2.2.6 GAS

Previously this paper has discussed transactions as signed messages that originate from externally owned accounts (EOA) transmitted by the Ethereum network and recorded on the Ethereum blockchain. We have also looked at transactions as the triggers to state changes or causes of contract executions. In this section, we will take a look at transaction gas.

Gas is a measure that is used to determine how many steps a transaction will need on the EVM. If a transaction is complex, meaning it requires more computation resources (CPU or memory), a user will need to pay more gas. The metric for gas is the smallest subdivision unit of Ethereum's ether currency, Wei. A transaction payload contains a gas unit price at the transaction time and a gas limit. A gas limit is an upper bound that the user is willing to pay in a single transaction. If the gas limit is exceeded, Ethereum will revert the transaction, and any funds used will be returned to the user's account. Once the gas limit is exhausted, the transaction will stop and throw an out-of-gas exception, and all state changes made by the transaction are reverted. Ethereum uses gas as a fee for using computation resources, and this fee consists of three components. A computation fee and transaction fee, and a storage fee. If a contract saves data to Ethereum, all nodes in the network will save it, which is very costly. Ethereum will charge for storage and encourage less storage usage. If the operation is to clear a storage item, Ethereum will do this for free.

**Why does Ethereum need Gas?**

Generally speaking, there are three main reasons: finance, theory, and computation. From a finance perspective, the purpose is to incentivize miners to execute transactions and smart contracts using their own time and resources [24]. If a user wishes to have their transaction prioritized, they can submit a transaction with a higher gas limit. In this way, transactions could be processed sooner by miners incentivized by higher transaction fees. As compensation for computation resources that the miner invests in, gas becomes more crucial after consensus migrates to Proof of Stake (POS).

In the PoS era, miners no longer get rewarded by mining blocks and packing transactions. It is more important for miners to process transactions and get paid for expending resources on the blockchain. The theoretical purpose is to align participants' incentives on the network [24]. Much of blockchain theory discusses how to mitigate harmful or malicious actors in a trustless environment. Gas partially addresses this issue: Miners are incentivized to work on the network, and users are
disincentivized from acting poorly or writing malicious code as they are putting their own ether (in the form of gas) at risk.

From a computational point of view, the computational reason behind gas goes back to an old, foundational aspect of computing theory, the Halting Problem [25]. The Halting Problem is the issue of determining whether an arbitrary program will stop running or if it will run forever just by looking at the description and the input values. In 1936, Alan Turing determined that no machine could solve the Halting Problem. In the EVM, this means a miner can never begin processing a transaction and know 100% that the transaction will not go on forever. With gas, specifically gas limit, a finite amount of gas is always attached to a transaction. Even if a miner began processing a transaction that was coded to continue indefinitely, either from a bug or an attack on the network, the gas would eventually run out, the transaction would end, and the miner would still be compensated.

2.3 Decentralized Applications (DApps)

The founding vision of Ethereum was a way to make a general-purpose blockchain programmable for a variety of uses. This encapsulated Ethereum growing into a platform for programming decentralized applications (DApps). A DApp is a web application built on an open, decentralized, peer to peer infrastructure. DApps are, in their most basic form, composed of smart contracts on a blockchain and a web front-end user interface. In addition, many DApps include other decentralized components like a decentralized (P2P) storage protocol/platform or a decentralized (P2P) messaging protocol/platform. These components can be balanced between partly centralized or partly decentralized [26]. For example, a front-end can be developed as a web app that runs on a centralized server or a mobile app that runs on a device. The backend and storage can be on private servers and proprietary databases, or one can use a smart contract and P2P storage.

The advantages of DApps are plentiful and include resiliency, transparency and censorship resistance [26]. Resiliency is afforded to DApps because its business logic is controlled by a smart contract, meaning that the backend is fully distributed and managed by blockchain. The on-chain nature of a DApp allows everyone to inspect code and be sure of its functionality, and all interactions are stored on-chain as well. As long as a user can access an Ethereum node, they can interact with the app without any centralized control.

The following subsections and subsequent section of this report will discuss decentralized applications in three parts as relevant to the design and implementation sections of this paper; backend, front-end and data storage.
2.2.1 DApp Backends

The backend of a DApp is a smart contract, which controls the application's business logic and related state. Gas is, of course, a major consideration for DApps. It means that any computation executed in a smart contract can be expensive, and design should therefore keep gas prices to a minimum. This means identifying aspects of the application that need a trusted and decentralized execution platform [22]. This backend architecture allows programmers to create and build architectures in which networks of smart contracts can call and pass data to each other. This involves reading and writing state variables, and complexity is only limited by the block's gas limit. Making it possible for architectures to be built on top of other contracts.

2.3.2 DApp Front-Ends

The Frontend interfaces with the EVM and solidity with standard web technologies like HTML, CSS and JavaScript. Interactions with Ethereum, such as signing messages, sending transactions, and managing keys, are often conducted through the web browser, via an extension such as MetaMask [22].

2.4 Decentralized Storage

Due to high gas costs and low block gas limits, smart contracts are not well suited to storing or processing large amounts of data [22]. Invoking a need for off-chain data storage. Off-chain storage could be done in a centralised manner, for example, in a typical cloud database. Preferably data can be stored on a decentralised P2P platform such as IPFS or Swarm. Decentralised P2P storage is ideal for storing and distributing large static assets such as images, videos, and the resources of the application's frontend web interface (HTML, CSS, JavaScript, etc.) because they provide advantages like low-latency data retrieval, efficient content caching, reliable fault-tolerant storage, censorship resistance and file versioning and archival functionality [27]. Decentralised storage is cheaper than using on-chain storage on Ethereum, and it avoids unnecessary bloating of the network with large amounts of data.

Furthermore, because of the limitation to the Ethereum block size, about 20k bytes for each Ethereum block on March 19th 2018 [28], it is impossible to store the audio files in the blockchain. In addition, each node needs to synchronise the whole data on the blockchain. If we store many large files on the blockchain, it is a waste of network resources. Making external decentralised storage valuable for the proposed solution in this paper.
2.4.1 Inter-Planetary File System (IPFS)

Unlike the worldwide web, which uses location-based addressing, the IPFS uses content-based addressing [29]. Every file has a unique hash that can be compared to a fingerprint. When a user wants to download a particular file, they ask the network for a file with this hash. Someone on the network will provide it to them. We know that the file has not been tampered with because the hash function allows us to verify what we have received—giving the IPFS built-in security and file integrity. On top of this, content-based addressing makes deduplication easy.

Nodes in the IPFS serve local copies of content to the network. When files are requested, a local node caches the response and continues to seed the file back to the network. However, if all nodes serving a particular file go offline, the file is no longer available. Solutions such as Filecoin [30] provide an incentive to encourage storage and proactively distribute files in the form of ETH. NFT storage allows us to utilise this functionality freely.

2.5 Tokens

A cryptographic token is a digital asset that lives on top of a cryptocurrency or blockchain. It is often a programmable asset that is managed by a smart contract. In Ethereum, ETH is a token, and other tokenised assets exist in Ethereum except they do not have their own blockchain or distributed ledger. Instead, they are built on top of an existing one.

Tokenisation is a concept that describes the representation of tokens as the right to something. Tokenisation is a way to convert the right to something into a digital artefact [31]. The advantages of this form of representation include higher liquidity, general programmability and immutable proof of ownership. Furthermore, the programmable nature of tokens can facilitate automated management of investor rights and compliance regarding the assets that have been tokenised, giving it the potential to increase speed. The immutable nature of blockchain provides absolute traces of historic transfers and proof that digital ownership has not been tampered with.

The purpose of tokens is primarily as a medium of exchange that they can act as a currency. For example, they could be deemed the local currency within a DAApp. Leveraging a token’s programmability can be used to trigger certain functions in the smart contract of the DAApp. Tokens can be linked to off-chain assets. They are apt for serving as a means of fundraising, pre-order or investment, and building an ecosystem or community. A token’s value is determined by its supply, demand and trust that the participating community has in it. This trust is built through credibility and quality service.
2.5.1 Token Types

A high-level token categorisation distinguishes payment, security, and utility tokens [32]. Many tokens are hybrids concerning this categorisation, but the distinction between each of the functionalities helps define the token roles in the design of this paper's proposed solution. Primarily, the purpose of security tokens is an investment instead of the added value to the functionality of a product obtained through utility tokens. To define the token types more succinctly:

- **Security Tokens** are assets in the form of debt or equity claims on the issuer. These tokens are analogous to equities, bonds or derivatives [33].

- **Utility Tokens** are typically backed by a particular project or DApp. They have a definable benefit, for example, access, and they intend to "provide access digitally to an application or service through a blockchain-based infrastructure. The issue of utility tokens does not require supervisory approval if the digital access to an application or service is fully functional when the tokens are issued." [32].

- **Payment Tokens** fulfil a payment function with little or no other function.

2.5.2 Token Contracts

A token contract defines the basic functionality of a token. The basics of this functionality include bookkeeping token holdings, transferring ownership of tokens and emitting events to record the transfers of ownership in the logs. The safe transfer functionality is an example of one such mechanism. Additionally, contracts can implement functionalities like the creation/destruction of tokens, which are referred to as the minting or burning of tokens. The minting process, from a high level, has the following steps that it goes through:

- Creating a new block
- Validating information
- Recording information into the blockchain

All of this involves immediate changes to the state of the contract, meaning it costs gas. Token contracts can also include authentication and roles, access control, information provision and utilities.

Due to the versatile ability of these contracts and to ensure compatibility, best practices for the implementation of token contracts are defined in the form of standard token interfaces. These interfaces allow software like DApps or wallets to recognize tokens and interact with them. Contracts in the Ethereum universe adhere to the Abstract Binary Interface (ABI) [34], which identifies functions by signatures that consist of the first four bytes of the keccak256 hash of the function name together with the parameter types. A contract's bytecode contains instructions to compare the first four bytes of the call data to the signature of its functions.
Bytecode can determine if a contract complies with an interface standard. This section references the available coding patterns in solidity, the prevalent programming language on Ethereum, for the community approved standards and best practices.

2.5.3 Token Contract Standards

The term ERC means Ethereum request for comments and refers to a document that smart contract programmers are using in the Ethereum blockchain platform to write. These are rules that the Ethereum based tokens must comply with.

![Fig 2.2 ERC Creation Process](image)

The ERC-20 token standard is the most common token standard in that it is the most widely used and the most general. It "provides basic functionality to transfer tokens and allows tokens to be approved so another on-chain third party can spend them." It is often referred to as the fungible token standard. Tokens of this description are constructed so that each fraction of a token is equal to the next. For example, Eth is fungible, meaning that one ETH is equal to another and all other ETH coins. These tokens are interchangeable and divisible. Blockchain is regarded as the ideal technology for managing all forms of digital assets due to decentralisation, security, and immutability. This would not be practicable with such interchangeable tokens. For cryptocurrencies, such tokens are OK, and in fact, fungibility is a key element of any currency. To conform to the API, it requires six functions as well as three optional functions and two optional events.

The ERC-721 Non-Fungible Token Standard [36] concerns tokens that are termed non-fungible. Non-fungible tokens are distinct from fungible tokens in that they represent one-of-a-kind, collectable items. They are one-of-a-kind since they cannot be divided or exchanged for other non-fungible tokens. NFTs can be considered non-fungible tokens that provide a number of unique uses for blockchain technology. The most well-known example of non-fungible, collectable tokens is Crypto Kitties.
Unlike fungible assets like Eth, every CryptoKitty is unique, and no two CryptoKitties are alike; it is impossible to break a CryptoKitty into smaller pieces, exchange them, and reassemble them to make an equally valued CryptoKitty. To conform to the API requires ten functions and three events.

The ERC777 Token Standard [36] maintains backwards compatibility while defining advanced features to reduce friction in token interactions. Primarily, it gets rid of the double transaction verification of ERC20, lowers transaction overhead and allows users to reject incoming tokens from a denylisted address. To conform to the API requires thirteen functions and five events.

The ERC1155 Multi-Token Standard [38] builds on top of the previous standards and introduces the management of any combination of fungible and non-fungible tokens in a single contract, including the ability to transfer or mint multiple tokens at once. To conform to the API requires six functions and four events.

This standard allows the development of a best of both worlds token, a fungible token that is a development on the base of Non-Fungible Tokens that allow exchanging like a fungible token but become inherit non-fungible values in terms of distinct metadata, indivisibility and individual value interpretation. This new standard also allows a single contract to manage a collection of semi-fungible tokens [39] with the least amount of data necessary to distinguish them from the others. The contract state contains configuration data per token ID and contains all the behaviour governing the collection.

The advantages of the multi-token standard include optimisations to make transactions more efficient and safer, bundling transactions together, and thus reducing gas fees. Multiple tokens in this standard can be sent or minted in a single transaction - offering significant savings on gas costs and preventing the need to wait for each block in single transfers. The ERC1155 implementation guarantees that event logs emitted provide an accurate record of all token balances enabling database or chain explorers to listen and provide indexed and categorised searches of every token in the contract. All token metadata is moved from the chain to an external JSON file stored off-chain, most often on decentralised storage. The JSON format makes localisation possible with multiple languages and gives better integration with wallets and other software like DApps. It is important to note that a key feature of this smart contract metadata standard is that they do not contain the artwork, images, or files themselves, but only the links or URIs to them and their metadata. These tokens reference off-chain resources for such files and information so that the blockchain itself is not responsible for hosting this data.
2.5.4 Other Contract Standards

There are three other main proposals and standards referenced in this document, which will be described here.

**ERC-2981, the NFT Royalty Standard**, allows for a digital asset to present a simple, standardized and gas efficient solution to any third party about the expected contractual royalties to be paid out. In essence, a focus on simplicity with an aim to help wider adoption from marketplaces. Royalties in this sense refer to a secondary fee made by the artist/creator on sales that do not include the artist/creator of the item being sold. For example, a painter makes a percentage of the profit on selling their artwork from the original purchaser to secondary purchasers. This concept then propagates for all purchases of the artwork. Though it is called the royalty standard, this concept will be referred to as secondary fees in this document to avoid confusion about the royalties incurred with music copyright.

The second standard to be discussed is the **EIP-884 called the Delaware General Corporations Law (DGCL) compatible share token**. It was published on the 14th of February 2018. Recent to this date, the Delaware State Senate passed an act that explicitly allows blockchains to maintain corporate share registries. EIP-884 allows for the creation of tradable ERC-20 tokens, where each token represents a numberless share issued by a Delaware corporation. EIP-884 is designed to represent equity issued by any Delaware corporation, whether private or public. By deploying an EIP-884 token, a firm is enabled to raise funds, either via initial public offering (IPO), or by private equity sale, in a manner conforming to Delaware Corporations Law but bypassing the need for a custom share registry or the involvement of a traditional stock exchange or transfer agent.

EIP-884 tokens ensure conformance with the following principles, exceeding the base ERC-20 standard. The token contract must provide the following three functions of a Corporations Stock Ledger:

- It must enable the corporation to prepare the list of shareholders.
- It must record transfers of shares.
- Shares must not be divisible.

As the tokens proposed in this solution represent shares, in music copyright, they are also designed to adhere to these guidelines.

As a security measure and to provide a better user experience when signing messages, the Ethereum community has developed **EIP-712** [54], a standard for signing typed, structured data. Signatures created with EIP-712 are "bound" to a specific instance of a smart contract running on a specific network.
2.6 Polygon

One of the bigger issues with using Ethereum is scale. As the number of people using Ethereum has grown, the blockchain has reached certain capacity limitations. On the layer one Ethereum blockchain, high demand leads to slower transactions and nonviable gas prices. The main goal of using polygon is to increase transaction speed (faster finality) and transaction throughput (high transactions per second) without sacrificing decentralization or security. This is fundamental to the meaningful and mass adoption of DApps.

*Polygon* is a layer two solution, which means that it is built on top of Ethereum. Polygon helps Ethereum to scale transactions by processing them on various new sidechains. Transactions on polygon are bundled together, checked for validity and then written to the Ethereum main chain. Polygon handles 65k transactions per second compared to Ethereum mainnet's 15-25tx/s [46], enabling polygon to combat congestion, bottlenecks & steep gas prices.

![Fig 2.3 Polygon Architecture](image)

The polygon architecture begins with the Ethereum layer. Polygon uses the Ethereum base layer to leverage Ethereum's high security. It is implemented as a set of smart contracts on Ethereum. This layer is used for finality and checkpointing, staking, dispute resolution and messaging between Ethereum and polygon chains.

The security layer provides validators as a service's functionality that allows polygon chains to use polygon validators that can validate the state of the polygon chain at any time in exchange for a fee. This layer is implemented as a meta-blockchain
parallel to Ethereum. It is responsible for validator managing, registering, deregistering, rewards shuffling and chain validation. It can be implemented directly on Ethereum and leverage Ethereum miners as validators.

The Polygon Networks Layer consists of sovereign blockchain networks where each network can maintain the following functions: transaction collation, consensus and block production.

The execution layers are responsible for interpreting and executing transactions included in polygons chains and consist of the execution environment and logic.
3  Music Copyright

3.1 Introduction

Music Copyright is the way by which the right to perform, transmit, reproduce and distribute music in any form is managed. Through the copyright, rights holders can determine how to profit from the music protected by the copyright. A song is an artist's intellectual property from the moment it is fixed in a tangible form, including anything from a rough demo to loosely written lyrics. They automatically own that work and retain the exclusive rights to its reproduction, distribution, and performance. Artists can also grant or deny those rights to outside parties; when the work is shared and consumed by the public, the song's recording will begin to generate royalties.

3.3.1 Types of Music Copyright

There are two types of music copyright; composition/publishing and masters recording rights, and they each incur specific royalties.

The first half of a song's copyright is its composition rights. Composition rights pertain to the original idea of a song put into a tangible form [62]. Once an artist has captured or recorded a song, they are considered its rightsholder by default. Composition rights are defined by such idiosyncratic characteristics as a song's lyrics, beat, melody, and structure. They belong to the song's writer(s) and publisher(s), who then collect performance and mechanical royalties from the composition.
The other inherent copyright, or half, every song has is its one-of-a-kind recording [62]. Recording rights are quite literal compared to composition rights; they refer to a very specific version of a song. A single composition can have countless unique recordings that vary considerably in terms of their style and sound or even artist. They can even be controlled by different labels and/or artists.

3.3.2 Split Sheets

In order for co-writers of a song to establish who owns what percentage of the song and the publishing rights that lead to publishing royalties. As an industry best practice, this ownership breakdown should be established as early in the creative process as possible via a split sheet [62]. This shared document clearly states the level of ownership each songwriter has in a copyrighted work. This ownership is determined by a mutual agreement between the writers, often based on how much they contributed to the overall songwriting. For example, the person who wrote a song's lyrics will likely receive a larger share than the person who wrote the bridge.

3.3.3 Identified Issues

One of the most substantial issues with the current developments of the music industry, as identified by [63],[64] and [65], is payment for the use of music copyright. A song's copyright ownership is usually divided between a group, and there is no comprehensive central database of song ownership data. Different licensing bodies maintain their own central copyright databases. This means that different countries and indeed different companies maintain separate music copyright databases [53]. The outcome of this is to pay the owners of a song. A payer must request searches across different limited proprietary databases. This can lead to massive delays, or even stagnation, in royalties payments. Part of this issue is the use of what is referred to as the “black box”. The "black box" refers to royalties that are never paid to the rightful owners due to poor licensing data, and this has caused billions of dollars in accumulated revenue to be unreachable by the artists to whom it is owed [66].

Blockchain can target this issue with the creation of a networked database of copyright ownership which would help solve the previously mentioned issue with music licensing [63],[67],[68] and [69]. A blockchain-based database would include all the currently sparse information in different proprietary databases. Thanks to blockchain technology, all the data would be updated instantly and automatically, secure, verifiable, and transparently available to all.

It is also suitable for smart contracts to manage royalty payments, [67],[68], [69] and [70]. Smart contracts would enable micropayments to be made quickly and with low
transaction costs. Currently, royalty payments are made, depending on the location of the royalty incurrence, at a few set times during the year, usually four. Smart contracts could allow royalties generated from streams on streaming platforms, who have blanket licences for streams of music existing to pay an artist immediately after a song is streamed. This quick revenue stream is essential for the prosperity of artists in small music markets. This constant cash flow allows artists more choices over their music management. Due to a lack of cash flow, artists often find themselves selling shares in their copyright in exchange for cash advances used to market, tour or distribute music, that heavily favours the businesses that offer them [72]. Even if the cash flow is not significant enough to replace these cash advances, blockchain offers artists more freedom in how they sell shares in their music. Since transaction history is available to all and easy to navigate using block explorers, artists can see how much others in a similar situation were able to profit. This encourages honest dealings. With blockchain, a peer-to-peer system gives the power to the artist by alleviating the need for a middleman. This use of blockchain to solve transparency issues is especially prevalent in the value chain, one for valuation in sales of copyright, but two, it is also relevant in licensing negotiations. The valuation of music and the royalties due have long since been a dark room to most, even within the music industry [62]. With available historical data, all parties can be informed as to the historical valuation of royalties.

3.2 Audio File Watermarking

3.2.1 Audio Steganography

Steganography is a method of concealing messages in different file formats without adding perceivable noise to the contents [42]. Steganography is often confused with encryption/decryption. The “carrier” file is perceivably unchanged to the public with encryption in steganography. It undergoes extreme changes and can effortlessly be viewed to be an encrypted file/message. Audio Steganography utilises Audio files as carriers to pass on the message.

Classification of types of audio steganography [43] include:

- The LSB Algorithm divides audio files into chunks where the least significant bit of each chunk carries bit information of the message.

- Echo Hiding involves embedding data into audio signals by introducing a short echo to the host signal. The nature of the echo is resonance added to the host audio.
• Spread spectrum involves encrypting the secret message using a cryptographic algorithm and then embedding it in the cover audio using DSSS (direct-sequence spread spectrum).

• Phase encoding applies Fourier transform onto audio divided into chunks. Phase changes in accordance with the message are applied to the first chunk of the audio file, and the audio file is regainable using the reverse Fourier transform. Phase changes in audio cannot be perceived by the human auditory system [40]; hence the proposed solution opted for the Phase Coding technique in the implementation.

3.3 Existing Market Analysis

3.3.1 BAND royalty NFTs

Band Royalty describes itself as a music NFT ecosystem. It enables users to purchase NFTs on the Ethereum Blockchain, linked to original animated images fashioned to the likeness of top musicians, in a range of rarities. Aside from rarity deciding an NFT's value through scarcity it also contributes to the utility of the token within the ecosystem, in how many collection pools it can be staked. Band Royalty owns shares in the masters' recordings of over fifty songs from platinum-selling artists. As such, they are entitled to collect royalties generated from those shares, including streaming and publishing royalties. 50% of the royalty payments received from industry royalty collection firms (like BMI, ASCAP) by BAND gets paid out on a schedule throughout the year to those who have staked their BAND NFTs. Stakers can stake their NFTs in one of three music pools. They can choose to take a share of the royalties generated from print music, mechanical and public performances, or synchronization to video. The minimum staking time is 90 days, but stakers can choose to keep their NFT staked for as long as five years. The longer the staking period, the more revenue they can generate. Owning a Band Royalty NFT does not equate to ownership of royalties. It is a buy-in to use the service. As BAND Royalty earns more revenue from NFT sales, the platform continues to buy more music royalties to add to the 3 pools, further increasing the pooled royalty stream income.

These BAND tokens are created using the ERC-721 Non-Fungible Token standard, and this means that each token is managed by an individual contract instance. Each token is linked to a unique digital artwork through its metadata, which is stored as a JSON object on the IPFS. The image linked in this metadata is also stored unencrypted on IPFS. Payments are paid out by BAND proportional to BAND Royalty NFT stakes. These NFTs are available for purchase on OpenSea, and those who have
staked the NFT(s) they own earn a secondary fee from any further sale of any Band NFT.

3.3.2 Opulous

Opulous's Security Non-Fungible Token (S-NFT) allows purchasers to receive rights to music royalties through a limited edition NFT representation. Each token represents a $1 share of the song. There are only 50,000 available per song, and there will never be more produced. Owners of the tokens get a share of profits generated by the master record. Profits include music royalties generated from streaming, neighbouring rights, and sync rights of the single the token relates to. The artist is expected to distribute royalty distributions quarterly in OPUL to token holders' digital wallets. OPUL is opulous's currency, it is implemented as an ERC20 token like other cryptocurrencies on Ethereum.

All tokens are deployed to the Ethereum blockchain, and S-NFTs are set to be tradeable upon the deployment of the Opulous NFT Exchange. Despite earning a stable royalty income, too many artists are turned away by traditional banks or forced into unfair deals with major labels. With Opulous, musicians can fund new projects with DeFi loans up to the value of the royalties they generate over 12 months. On the other side, investors can stake their crypto assets on the platform to generate high returns.

3.3.3 Royal

Royal enables anyone to own a piece of their favourite songs to earn royalties alongside the artist. Artists choose what % of the song's royalties to put up for sale. They can also bundle fan experiences, special tracks, or digital art. Buyers purchase these streaming royalties in the form of tokens directly from the artist. Any extra bundled benefits will be attached to this token. They refer to these tokens as Limited Digital Assets (LDAs), which they define as a special type of NFT. LDAs are tokens representing ownership in music. Each token is deployed to polygon and comes with: (a) A percentage of the royalties from streaming services. (b) A licence for non-commercial use of the digital art associated with the token. (c) Additional benefits that the artist chooses to include. Once someone has bought a token, they can claim royalties for the song after they have accrued. The time to payout will vary depending on the artist, who is responsible for redistributing payments they receive for the copyright.

The tokens implement the ER-1155 metadata but with a few non-standard compliant changes for better integration with OpenSea. These do not affect the token minting or transfer fundamentals but do hinder its ability to comply with other market
platforms. ERC-1155 intends for tokens to be distinguishable and store as little data as possible. In order to achieve this, the URI value of the metadata, stored by the token contract, allows for ID substitution by clients. If the string `{id}` exists in any URI, clients MUST replace this with the actual token ID in hexadecimal form. This allows for a large number of tokens to use the same on-chain string by defining a URI once for that large number of tokens. An example of a compliant URI would be: https://token-cdn-domain/{id}.json. On OpenSea ERC-1155 tokens override the openzeppelin standard-compliant implementation to apply the ERC-721 metadata standard to the ERC-1155 tokens it creates [83]. This makes ERC-1155 metadata unreachable by other marketplace implementations.

Royal stores a link to the audio file as part of a tokens metadata. This link directs users to a centralized server where admins have complete control of the data and can update or change it at will, which means that the audio file connected to the metadata is not immutable. It is preferable for the audio file to represent the music for which it is selling the copyright and that this file cannot be changed as the copyright cannot be.

As of January 2022, Ethereum and Polygon NFT royalties are set by exchanges, which makes royalty enforcement challenging. ERC-2981 is a royalty standard that will likely be implemented across NFT exchanges in the near future. The standard's royalty info function returns a public address for the intended royalty recipient(s) and a royalty amount in the sale currency. An exchange would query the function with the NFT's tokenId and sale price and then remit royalties accordingly. Currently, secondary fees on OpenSea are implemented in such a way that only the OpenSea can operate with them. It is likely that in the future, Opensea will, like many other popular NFT marketplaces, switch to using the royalties standard implementation for royalties so that they can be enforced across platforms.
4 Solution Design

4.1 System Overview

The proposed solution aims to encapsulate copyright ownership tracking among multiple owners, copyright royalty payments, royalty distribution and the sale of copyright shares. The presented solution aims to have each share in copyright managed by an individual token and for each share of that copyright to be managed by a single contract. This contract will facilitate the royalty distribution to the owners of each share as well as the sale of shares. Though this functionality is implemented by the token contract, the solution proposes an application contract, upon which a decentralized app is built. This application contract, core to the overall system goals, facilitates easy interaction with the token contract and additional security, storage and helpful features. This section centres upon the discussion of token contracts, DApps and blockchain elements examined in the previous section as the foundational concepts of this proposed solution.

To develop such an application, the choice of a platform is essential because the platforms differ in aspects like functionalities, access control, and consensus algorithms. These variations have a direct, profound impact on the performance, scalability and reliability of the applications built on top of these platforms.
4.2 Tokenisation of Shares

Each share is represented as a token on the blockchain. Each tokenized share grants the ability to link copyright shares directly to the work they represent via a token's metadata. Each token, once minted, will be associated with a particular private key, which acts as proof of ownership of the share. This is a form of transparent and tamper-resistant ownership because the owner's private key can confirm the authenticity of ownership. Due to the historical transaction data stored on the blockchain, it is also possible to verify the content creator and authenticate the token via the content creator's public key.

4.2.1 Why Tokenisation?

It is possible to represent shares with any number of technologies, but this paper submits that Ethereum tokens are the best fit to manage music copyright. The P2P nature of Ethereum allows users to move money or make agreements directly with someone else without a need for intermediary companies. This is also true for trading tokens, which can happen peer-to-peer without needing platforms to take large cuts as compensation.

Through commerce guarantees, Ethereum creates a more level playing field. Users have a secure, built-in guarantee that funds will only change hands if they provide what was agreed. Since the contract is deployed to the blockchain, users have a clear and honest guarantee that funds will only change hands if explicit agreements are met.

Transparency is inherent due to the blockchain's immutable transaction history and token metadata being publicly verifiable – it is simple to prove ownership history.

This immutable history means that it's nearly impossible to manipulate that data to "steal" ownership once a transaction is confirmed. As new blocks are added to the chain, earlier blocks become harder and more expensive to modify. Soon, generally within a few blocks, the cost to "change history" becomes so great that it is effectively impossible, and the information recorded in the blockchain can be considered permanent. Not only does this immutable history and built-in transparency secure the ownership of tokens. Past transactions can inform future ones. We can have clear and irrefutable past valuations of a song. This creates a more honest and open approach to the valuation and monetization of music in general.

The final reason is availability. Ethereum never goes down, meaning that tokens will always be available to sell. This property of permanence and stability is central to the token value proposition. By using a blockchain as a durable shared data storage
medium, tokens can be trusted to endure as long as the blockchain remains operational, bringing up another interesting property of blockchains. A blockchain incentivises its own survival by rewarding node operators with cryptocurrency in exchange for keeping the network alive. As long as there are people attracted to the economic reward, there will be someone motivated to keep the network online. This ensures the survival of all historical data, including tokens.

4.2.2 Token Design

As discussed above, tokens can take the role or type of a security, utility or payment token. The proposed design for the token that represents a share is, like most tokens today, a hybrid. It claims the title of a security token as a share because it equates to a share in the copyright of a song, which has inherent value. This inherent value means that it can be bought and sold, enabling the free trade of the music copyright ownership.

It also holds properties of a utility function because its nature allows functions like royalty distribution and copyright payments.

The numerous advantages to the newest token standard, the multi-token standard (ERC-1155), paramount to which is the ability to manage many tokens in a single contract, makes it the best choice for this design's token representation. The choice to abide by an existing token standard are two-fold. Token standards are tried, tested and approved by the Ethereum community as secure, stable and efficient. The commonalities in these standards allow for easy integration with existing software like wallets and DApps. The ERC-1155 standard enables efficient transactions, bundling, guaranteed log trace and efficient gas reductions over the previous standards, making it the best fit for this proposal. Adhering to this standard means that minting, transfers and metadata are standardised. This principle of “not reinventing the wheel” but building on top of it allows the token contract to be built upon by other developers.

It is possible to operate over multiple tokens in a single transaction efficiently, as all of the state is held in a single contract, due to the implementation of the functions balanceOfBatch and safeBatchTransferFrom, which enable more straightforward and less gas-intensive queries and transfers for multiple tokens [48]. In the spirit of the standard, mint batch functionality is also included.

The smart contract can enforce hard caps on token supply, so purchasers of a token can be sure of scarcity and value. The exact distribution of the tokens and their value can be determined by the user.
For example, if a user creates a contract that will manage tokens related to the ownership of the masters recording of “Fix You” by Coldplay and we assume that the only people who own that copyright are the four members of Coldplay, then it would be up to them to distribute ownership in accordance with their split sheet. They are enabled to distribute this ownership so that each token represents an individual share. Each share corresponds to percentage ownership based on the total number of tokens enforced by the hard cap.

\[ \text{Ownership} = \frac{\text{Number of tokens owned}}{\text{Total number of tokens available to be owned}} \]

In this case, all tokens can be viewed as conceptually fungible as each token represents the same value and is interchangeable. In relation to token fungibility, this is a hybrid of the original implementation of a fungible token (ERC-20) that inheritance properties of a non-fungible token (ERC-721). Shares cannot be divisible as is explained in the ERC-884 standard, and this is a property of the ERC-721 standard. Unlike ERC20, ERC1155 lacks a decimals field to ensure each token is distinct and cannot be partitioned. These tokens also have distinct metadata that is distinguished by its ID. Finally, although the token represents the same share in the song, it may not hold the same value. A buyer may view a token minted by the lead singer of Coldplay, Chris Martin, to be more valuable than a token minted by the lead guitarist, Jon Buckland, due to the notoriety of the individual. This phenomenon has led to the recent, colloquial adoption of the term semi-fungible to represent a hybrid of the two implementations, although conceptually, these tokens are fungible.

It is also possible for the members of Coldplay to each own a single token that represents percentage ownership. In this case, there are only four tokens, each representing potentially different ownership shares. For example, one band member could have ownership of 25%, and another could have ownership of 30%, but a single token could represent each share. Suppose each of these tokens represents a different share. In that case, we will term these tokens non-fungible because each has an inherently different value in terms of the percentage ownership share of the masters.

More than likely, a user will choose to implement a combination of the two options, with a number of shares representing different proportions of ownership and perhaps a number of tokens representing the same percentage of ownership. To use the Coldplay example a final time, Chris Martin could own a 20% share and choose to represent this as four tokens that each hold a value of 5% and Jon Buckland could own a 30% share and choose to represent this as a single token. The incentive to
split shares would be that they are available to trade in smaller chunks, and so the artist can accumulate funds in more minor sales, which are likely to be more frequent. It is recommended that the user have each token represent an individual share.

4.2.3 Secondary Fees

A secondary fee means that when a copyright share is sold, the contract owner(s) make a small fee on the exchange. This is a relatively new notion to profit on the sale of shares in which the original owner/content creator is not a party, which is why this aspect is optional for individual user adaptation. In keeping with the Ethereum development ethos, its adoption would be left to the consensus of the Ethereum community or the even consensus among platform users. It is potentially worthwhile because an artist can profit from higher demand of stakes in their art even if the demand was not as high when they sold their share.

As discussed above, implementation for this is varied and incompatible across token marketplaces today. With many custom implementations, recently, a standard was published in the hopes that implementation would be consistent across platforms in the near future. This proposed solution implemented this standard for forward compatibility.

4.2.4 Royalty Payments

With the token contract keeping track of token ownership and with that percentage share of the copyright, it can distribute payments safely to the owners of the copyright quickly, with a built-in guarantee of payment allocation and without the need for third-party intervention. This payment scheme using blockchain allows anyone to see how copyright has been historically valued, and this transparency would be integral to the future valuation of copyright and licensing negotiations. As a best practice, Ethereum developers recommend pull over push payment methods in this scenario [48]. This shifts the risk associated with transferring ether to the receiver.

There are several reasons why a push payment external call could fail. If the receiving address is a contract, it could have a fallback function implemented that throws an exception once it gets called. Another reason for failure is running out of gas. This can happen in cases where a lot of external calls have to be made within one single function call, for example, when sending the profits of a bet to multiple winners. Because of these reasons, developers have opted to follow a simple principle: never trust external calls to execute without throwing an error.
The Pull over Push pattern is an excellent way to mitigate this vulnerability when sending ether, especially when performing multiple transfers at once. Due to the isolation of the error-prone transfer functionality, one failed transfer does not lead to revert of all successful operations. Additionally, it is now the responsibility of the requesting user to make sure that they can receive ether, which they are incentivised to do. As a rule of thumb, developers should use the Pull over Push pattern when

- They want to handle multiple ether transfers with one function call.
- They want to avoid taking the risk associated with ether transfers.
- There is an incentive for users to handle ether withdrawal on their own.

All of which apply to the royalty payment distribution scenario. This pull method does not negate the speed at which shareowners will receive royalties, once the blockchain confirms the royalty payment, funds are available to be pulled by the copyright holders.

4.2.5 Modularization

Modulization is a best practice for developing in all languages but is especially prevalent in Solidity smart contract development.

Security is the distance from the implementation to what the code is expected to do. Security is very hard to prove, especially when the code set is unnecessarily large and complex. And this means:

- Functions should small
- Files should be small
- Independent logic should be separated by module
- Each module should implement a single logic
- Use explicit and easy-to-understand name convention

Writing contracts to be as simple as possible without sacrificing performance allows others to quickly understand and audit code, which is fundamental to the Ethereum development philosophy. Further, making token contracts understandable can give purchasers a clear understanding of what they are purchasing.

As each contract manages the shares for specific copyright, a new copyright requires a new contract. This involves deployment costs, which can be gas intensive and in some cases impossible for large contracts. Ethereum contracts with too many functions and too much code will hit the maximum contract size limit of 24KB [28]. To avoid this in the proposed design, the main token contract implements core functionality and facilitates interoperability between the library contracts.
4.3 Decentralised Application

A single authority has absolute control over all activity in a centralized application. This can lead to limitations like mandatory fees for listing and selling items, lack of privacy or control over user's own accounts and possibly even improper transaction security. This section presents a decentralized application that aims to combat these drawbacks and enable easy and user-friendly interaction with the proposed system's token contracts to showcase the ownership tracking, copyright payments, royalty distribution and trading of token functionality.

![Diagram of DAPP Development Process Flow]

4.3.1 Application Contract

The application smart contract is the backbone of the Ethereum blockchain DApp and runs on the Ethereum Virtual Machine (EVM). It is a digital agreement that reads, writes and executes the system business logic.

This contract coordinates token sales, contract management and payments to contracts. This is done at no extra gas fee, as calls do not change the state of the blockchain and thus do not cost extra. It enables us to combine calls to the token contract and save gas. Interacting with the application allows the user to have blockchain specific aspects of token interaction handled for them. Under the situation of out-of-gas (OOG) exception, at least one 64th of GAS will be reserved for sender to handle OOG exception and to stop the execution. This mechanism tries to avoid bubbling up the exception and is handled by the application contract.

As the application contract centres around the token contract, it is approved by the token contract to perform actions such as token transfers between accounts once the conditions of the token listing are met, for example, price. This trade functionality is compatible with all ERC-1155 compliant tokens, meaning that they are able to be listed, bought and sold on the platform as well. This is also true of the token implementation. All tokens created with this scheme are compatible with other ERC1155 token marketplaces but only for core ERC1155 functionality such as transfers. It would not include project-specific elements like payments. The project implements the marketplace functionality according to Ethereum community approved best practices and was developed to showcase the system's potential as a whole fully.
4.3.2 Application Overview

This section presents the proposed decentralized application. It is developed using the Polygon network interface provided by web3js and the ReactJS API. It allows web application events sent by the Ethereum network and the ability to provide transactions to the network.

The DApp talks to the front-end server via the blockchain browser; instead of talking to a backend server as with centralized applications, it talks directly to the blockchain, which hosts all the code and data. The individual components of the DApp, and are designed with reference to [49],[50],[51], are as follows:

- **Front-end Server**: This server facilitates user interaction with blockchain, IPFS and Node JS.
- **Blockchain**: This is the primary backend of the application containing all of the code and transaction data. This design uses Polygon as the blockchain to power the proposed solution.
- **Inter-Planetary File System**: For storing token metadata and audio files.
- **Mongo DB JSON Database**: This is a supplementary database for optimizing data storage for marketplace specific operations, and its use is optional for customers of the marketplace.

![Fig 4.3 Proposed Application Architecture]
4.3.3 Metadata, Encryption and Steganography

The ERC1155 standard requires a specific metadata format as stated in previous sections. This is to aid in the goal of achieving the least amount of information possible needed to distinguish between tokens. Also stated in the standard is that token metadata must be stored as an external JSON file. The DApp configures the JSON files for each token and stores them, along with the audio file, on IPFS.

Audio Files are stored on IPFS as Free Lossless Audio Codec (FLAC) files. This lossless compression format supports hi-res sample rates, takes up about half the space of WAV, and stores metadata. It is royalty-free and is considered the preferred format for downloading and storing hi-res albums [52]. The audio file's IPFS URI is stored inside the metadata. As it is stored on the IPFS, it is only addressable by its hash. If the content address leaks, everyone with the content address can access the file. In order to withdraw this limitation, we can encrypt metadata to protect their security of them. Key management is used to manage the decryption key. Digital signatures avoid the man-in-the-middle attack during the key exchange process.

Audio steganography is a digital watermarking technology that embeds digital information into digital media. We can use a watermark to add the token contract information to the audio file and allow tracking of the file ownership offline [53]. In this case, the audio file would first be uploaded upon contract creation to IPFS embedded with the contract hash. This embedding would be done with phase encoding.

As discussed previously, phase coding exploits HAS insensitivity to the relative phase of different spectral components. It is based on replacing selected phase components from the original audio signal spectrum with hidden data. It's considered one of the most secure mechanisms for audio watermarking without compromising sound quality, which is vitally important for our system. Once illegal propagation of this audio file happens, the contract owner(s) can discover the leak. This discovery is especially relevant if a song is unreleased. The idea here is to de-incentivise the illegal propagation of music as much as possible to maximise the earning potential for copyright owners.

4.3.4 Lazy Minting

The marketplace DApp also facilitates 'Lazy Minting'. The basic premise of lazy minting is that instead of creating a token directly by calling a contract function, the contract creator prepares a cryptographic signature of some data using their Ethereum account's private key.
The signed data acts as a "voucher" or ticket that can be redeemed for a token. The voucher contains all the information that will go into the actual token. It may optionally contain additional data that is not relevant to the token once minted, e.g. price to redeem the voucher. The signature proves that the contract creator authorized the creation of the specific token described in the voucher. If the signature is valid and belongs to an account authorized to mint NFTs, a new token is created based on the voucher and transferred to the buyer.

In the proposed solution, lazy minting allows the proposal of an acceptance mechanism for the distribution of ownership. This means that ownership of the contract by the original copyright holders can be evaluated by access roles that are assigned upon redeeming a voucher. We are further able to secure our vouchers by tying them to only be redeemable by a particular externally owned account (EOA). It also allows us to defer the costs of minting all of the shares from the contract creator to all of the owners of the shares.

4.4 Polygon

4.4.1 Deployment
Polygon is a decentralized Ethereum scaling platform [46]. It enables the building and creation of user-friendly DApps with low transaction costs without sacrificing security. The market contract and token contracts are deployed to Polygon in this solution. Since Polygon is built on top of Ethereum, the DApp interacts with Polygon the same way it would with the Ethereum mainnet via a blockchain browser, e.g. Metamask. All transactions are processed on Polygon using the native currency,
Matic. As of the demonstration of this project on 25th March 2022, 0.0005223 Matic is the equivalent of 1 Eth [46].

4.4.2 Bridges

Polygon supplies a two-way trustless transaction channel between Polygon and Ethereum by introducing a cross-chain bridge [56]. This cross-chain bridge enables users to transfer tokens (including currency as both ETH and MATIC are ERC-20 tokens) without incurring third party risks and market liquidity limitations. Polygon's bridge uses a dual-consensus architecture to optimise for speed and decentralisation. As discussed in the polygon architecture section, there was a conscious decision by Polygon developers to create the system to support arbitrary state transitions on sidechains, which are EVM-enabled. As this project token adheres to a confirmed token standard, it can be transferred between polygon and Ethereum. It should be noted that vouchers, for lazy minting, can only be redeemed on the blockchain on which they were created, but post minting are free to be transferred.
5 Implementation

5.1 Application Contract

The application contract is responsible for executing the business logic of the system. It maintains copyrights created on or added to the system and their token shares. This involves retrieving tokens for sale or owned for display, recording vouchers that have yet to be redeemed (for lazy minting), maintaining the token contract registry and facilitating function calls to the token contracts. Some of the notable aspects of this contract's design are discussed in this section.

The application contract is, in part, a factory contract. A factory contract creates and deploys new smart contracts from a smart contract class. In this way, it can deploy multiple smart contracts with different parameters directly from the blockchain itself, directly creating instances of objects. Solidity treats smart contracts as objects. This made it possible for the application contract to deploy multiple instances of token contracts with high gas efficiency for a usually gas-intensive task. The clone factory pattern, provided with EIP-1167 created by Peter Murray, Nate Welch and Joe Messerman [73], promotes this. Since the same contract is being deployed with different parameters, each instance will have identical bytecode. This makes storing all bytecode for each deployment repeatedly wasteful and promotes unnecessary gas costs for the bytecode. This proposal utilises a mechanism that deploys one abstracted instance of the token contract that does not represent copyright and has all other instances that represent copyright behave as proxies. These proxies delegate calls to the first instance of the token contract and allows functions to run in the context of proxy contracts. This enables each instance of the token contract to have its own state and to use the instance of the abstracted token contract deployed upon application set-up as a library.

The application contract also maintains a name registry for token contracts. It records the address of these factory-created token contracts for the DApp. This enables easy locating and interaction with contracts to facilitate the system's goals. In this case, the proposed solution uses the Name Registration Pattern to store the mappings between contract names and contract addresses so that it can find the contract address according to the contract name. This method also provides the ability to track different versions of token contracts, though the system implementation does not support this because the token contracts have to remain immutable to retain their value.

The application contract facilitates batching of calls to token contracts. Batching can reduce gas costs because it can reduce common data processing. Executing a
function N times will process the common "header" field N times and call the function N times. However, the function is called only once using batching, and the common "header" field is processed only once. Therefore, it saves gas costs by reducing "CALLDATALOAD," memory allocation, and function call operations. The larger the N, the larger the savings.

The application contract maintains data integrity by implementing restrictions on data calls and updates with access control. Callers to functions in the contract can only access data that pertains to their address. Function modifiers enforce this. In solidity, a modifier is used to change function behaviour. In this case, modifiers are used to check the condition before executing a function.

5.2 Token Contract

The token contract facilitates the tokenisation of music copyright shares. A single token contract manages a number of shares (tokens) by way of its state and in accordance with the ERC-1155 Multi-Token standard. It facilitates the distribution and payment of royalties to owners of the tokens by way of pull payments. All shares are assigned upon contract creation to the original copyright holders (songwriters) by creating vouchers assigned to each holder. This accounts for the use case that a royalty payment may be made before each token is minted. Payments can be assigned prior to all tokens being minted, but a voucher holder cannot claim royalties before they mint their tokens. This is intentional as the minting of a token solidifies the copyright holder as an original creator of the art that the copyright pertains to.

Notable aspects of the implementation of this contract include:

This contract provides a general role-based access control mechanism and easy management of ownership of tokens. Multiple hierarchical roles can be created and assigned to multiple accounts. Owners of tokens are granted roles that come with certain privileges. These privileges include the right to transfer tokens they own. This involves the granting of token ownership roles. This implementation also includes granting roles to token minters; provided the minter remains the owner of the token after minting, they are granted both roles, but upon transfer of the token ownership, the role is revoked. The minter maintains an administrative role for the contract that manages all tokens, which gives them the ability to mint new tokens, provided the hard cap has not been reached. Function modifiers enforce the access of these roles.

This contract allows splitting royalty payments among a group of accounts. The sender does not need to be aware that the funds will be split in this way since it is
handled transparently by the contract. The split can be in equal parts or any other arbitrary proportion. The way this is specified is each account's number of shares. Of all the funds that this contract receives, each account will then be able to claim an amount proportional to the percentage of total shares they own. This contract follows a pull payment model. This means that payments are not automatically forwarded to the accounts but held in this contract, and the actual transfer is triggered as a separate step by calling the release function.

This contract also facilitates the lazy minting of tokens. Lazy minting allows the system to defer the creation of tokens on the blockchain until they can be created by the rightful copyright holders of those shares. Vouchers are created for each copyright holder, containing information about their shares and are digitally signed to prove that the contract creator authorised the creation of the specific token(s) described in the voucher. Using signatures for authorisation can be tricky since a third party could potentially take some data signed in one context and present it somewhere else. For example, they may take a signature authorising the creation of an NFT on a test-net and present it to a contract deployed on the Ethereum mainnet. Unless the data being signed contains some context information, this kind of "replay attack" is relatively trivial to perform and hard to defend against. As a security measure and to provide a better user experience when signing messages, the Ethereum community has developed EIP-712 [54]. A standard for signing typed, structured data. Signatures created with EIP-712 are "bound" to a specific instance of a smart contract running on a specific network. They also contain type information so that tools like MetaMask can present more details about the data being signed to the user instead of an opaque string of hex characters. This token contract implements the EIP 712 domain separator (_domainSeparatorV4) used as part of the encoding scheme, and the final step of the encoding is to obtain the message digest that is then signed via ECDSA (_hashTypedDataV4). The application contract maintains the vouchers until they are redeemed, at which time the token contract mints them and creates them on the blockchain. These vouchers are tied to specific addresses and can only be redeemed by those addresses so that only the copyright holders can redeem vouchers. This is intensional as the minting of a token solidifies the copyright holder as an original creator of the art that the copyright pertains to. This allows the creator administrative contract controls and, case dependant, the ability to receive secondary fees, in accordance with their original share in the copyright.

This contract implements the royalty payment standard to provide ongoing funding for artists/creators. This functionality is provided by the token contract library, to which an individual contract is a proxy. This means that opting to or not to use this functionality does not increase the fees required to deploy this contract. Token
buyers will assess the royalty payment as a factor when making NFT purchasing decisions.

5.3 DApp

A decentralised application (DApp) can be considered a web application with key components distributed to a P2P network. In this way, DApp lowers the risk of a single point of failure (SPOF) and ensures user experience.

The technologies used for this implementation were as follows:

- **Database: Mumbai Testnet**

The Mumbai Testnet is an instance of Polygon blockchain and powered by the same underlying software. The Mumbai Testnet is used for testing and experimentation without risk to real funds or the main Polygon Network. Although the currency has no real value, this network also uses Matic and can be obtained via Mumbai Matic faucets [74].

- **Contract Programming Language: Solidity**

The most popular smart contract programming language is Solidity. Solidity is an object-oriented programming language created specifically by the Ethereum Network team for constructing and designing smart contracts on Blockchain platforms [78]. It also provides the ability to utilise the OpenZeppelin Library [48]. OpenZeppelin is a library that helps the development of smart contracts in Solidity with high security. It provides a “backing of a solid foundation for smart contract development through community reviewed code” [48]. In particular, the contracts library provides a standard implementation of the ERC-1155 token standard, role-based permissions scheme with high flexibility and utilities like cryptography and multicall.

- **Frontend: React.js [80], Next.js [81] and Tailwind [82]**

- **Frontend and Contract Interaction: Web3.js**

The Web3.js library [75] is a JavaScript interface for Ethereum which conforms to the Generic JSON Remote Procedure Call (RPC) Specification [76] used by other Ethereum clients. It is platform-independent and operates within the browser. Web3 enabled testing using a local Ethereum node and finally was pointed at a remote node connected to the Mumbai testnet. The remote node was provided by Infura [77], a service offering public nodes that serve blockchain RPC requests for decentralised applications.
- **Frameworks:**

  Hardhat [79] was used to deploy, test and compile smart contracts. It was also used to mimic a local node for testing purposes.

- **Wallet Software: Metamask**

  Metamask is a wallet software with a browser plug-in for managing an account.

- **Hosting**

  The DApp Client, web frontend server, can reside anywhere: static web pages or mobile phones. This solution proposes that contents be hosted by IPFS, to make it possible that all contents (web page, resources) are distributed in a decentralised way.

- **Node.js server**

  Node.js was used to develop the app locally, using testrpc/Ganache for testing. This server-side program is included in the implementation as code on-chain cannot communicate with off-line service directly. If code on-chain needs to interact with third-party services, such as an external digital audio watermarking service, it needs a server-side program. Secondly, the server-side program can be used as a buffer or index engine. It is common for client-side programs to provide search functionality or validate on-chain data based on the server-side service.

### 5.4 Token Metadata

#### 5.4.2 Metadata

The token metadata includes a link to the audio file so that each share has an immutable connection to the work that it represents a share in. Before the decentralized application can create the blockchain record for a token, it needs to store all off-chain resources that comprise the token "package". Once everything has been stored on the IPFS, it can use the IPFS URI for the metadata to link from the on-chain token to everything else. The IPFS network is a decentralized data store, where each file is addressed by a hash of the content instead of a URL. Using its hash value to address a file prevents tampering. We can upload data on the IPFS network and store the corresponding hash value in the contract.

This solution stores token FLAC audio files and metadata using the NFT.Storage HTTP API. This uses the store operation, which accepts metadata and asset files in one request and updates the metadata to link to the asset files using IPFS URIs. This method expects the metadata to conform to the ERC-1155 metadata schema. This
standard is backwards compatible with ERC-721 metadata and is generally well supported by various wallets and marketplaces.

The URI value of the metadata allows for ID substitution by clients. If the string \{id\} exists in any URI, clients MUST replace this with the actual token ID in hexadecimal form. This allows for many tokens to use the exact on-chain string by defining a URI once for that large number of tokens. An example of such a URI: https://token-cdn-domain/{id}.json would be replaced with https://token-cdn-domain/0000000000000000000000000000000000000000000000000000000000000004cce0.json if the client is referring to token ID 314592/0x4CCE0.

An example of a token's metadata:

```json
{
  "name": "Master Recording to Fix You",
  "description": "The masters recording to fix you by Coldplay",
  "audio": IPFSHash,
  "properties": {
    "Performed by": "ColdPlay",
    "Locations": {
      "Mixed At": "Quad Recording Studios, Westwood One",
      "Mastered At": "Sterling Sound",
      "Recorded At": "The Town House, Parr Street Studios, Chicago Recording Company, Sarm West Studios, Air Studios, Hollywood Bowl",
    },
    "Date": "05-09-2005"
  }
}
```

When adding custom properties, they are put in the properties object, rather than at the top level. Although wallets and other clients will not understand the meaning of the custom fields, they will be able to show the name, description and audio file since those all conform to the ERC-1155 and ERC-721 specifications.

5.4.2 Metadata Protection

Each audio file uploaded to IPFS has its own fingerprint hash as an identifier. People who have this value can retrieve the audio file it corresponds to from IPFS. The token contract on the blockchain stores the metadata hash to confirm that the data is unchangeable and traceable. In some cases, copyright holders may want the audio file to be available to all so that it is confirmable to all precisely which masters recording a contract relates to. Others may be satisfied with descriptions and wish to protect the download of audio files pertaining to a song. This solution proposes two combative measures to prevent access to the audio files.
5.4.3 Encryption & Key Protection Store

The first measure is to encrypt audio files to protect their security. The application operates a key management system that manages the decryption key. There is potential for man-in-the-middle attacks during the key exchange process, but digital signatures are used to avoid this. The management of these keys involved including a third party, the key protection center, which will be centralized. The application contract introduces a listing fee for the use of the key protection centre to incentivize its maintenance and updates. The key protection centre helps to agree the key with the contract creator and will decrypt the file for token owners or minters to access the file. The key agreement uses the Station-to-Station protocol, based on the classic Diffie-Hellman key exchange protocol. During encryption, the key will be provided as input to the AES algorithm [53].

5.4.4 Audio Steganography

As discussed previously, audio steganography is a digital watermark of sorts that embeds digital information into digital media. This system proposes the optional inclusion of audio steganography to protect audio files uploaded with this system by implanting the file with the token contract information so that if illegal propagation of the file occurs, it is easily revealed. In this paper, a phase encoding method is used to embed the token contract address.

The input FLAC file used as a carrier is separated into the header and data parts and is recombined to form the encoded FLAC file after operations are performed on the data portion of the audio. The data part is segmented into chunks, and Fourier transform is applied to each chunk. Phase shift is applied in the first segment with encoded contract hash converted into bits. Hidden data is regained by applying the inverse Fourier transform on the modified array of complex numbers. An encoded audio file is obtained by recombining the header portion. To decode the file is run through the Fourier transform, and the phase differences are extracted from the first chunk of data to get back the secret message. This would be used for confirming ownership.

This type of coding for steganography is preferred compared to the LSB algorithm and others discussed in section 3.2.1, where the carrier signal is corrupted partly due to noise introduced when changing the LSB.
5.5 Interface

5.5.1 Contract Creation Screen

This screen allows the user to enter details pertaining to the new copyright they are introducing to the system. They provide details to be stored in metadata and contract data structures. The user uploads the audio file to which the copyright should be linked to the system, which is subsequently formatted and uploaded to IPFS. Once the user confirms that the details entered are correct, then they click on the button which initiates the contract deployment, and the wallet software asks for confirmation of transactions approval. Below is a voucher signing interaction triggered once the contract deployment is confirmed.
5.5.2 Voucher Redemption Screen

Fig 5.3  Voucher Redemption Screen

This is the screen that users who have been assigned vouchers are able to redeem them. They confirm they are willing to accept the minting gas costs and the system mints the token for them. Once the token minting is confirmed the user is able to interact with it.

5.5.3 Token Management Screen

Fig 5.4 Owned Token Management Screen

This is the screen where the user can see all share tokens they own. They are able to list these tokens for sale on the marketplace. They are also able to withdraw payments due, this can be done from any token that pertains to a particular copyright and all funds due from that copyright will be transferred to the token(s) owner. Past owners are able to claim payments due to them after a token has been transferred but no new payments will be assigned to them once the sale of the token is confirmed.
5.5.4 Token Marketplace Screen

This is the screen where the user can see tokens for sale in the marketplace and make purchases of shares they would like to own. Shown also is the metamask browser extension. This manages the account for a user. This user is connected to the application and has 3.7 matic available to spend.

5.5.5 Contract Interaction Screen

This is the screen where any user can make a payment due for a license to the copyright to which the license pertains.
5.5.6 Polyscan

These screenshots show the block explorer, Polyscan's view, of the transactions, contract creation, token minting and token ownership transfers that took place during the demo presentation for this project. This is included as it showcases how readily available transaction data for copyright is available.
6 Security Evaluation

The security of the operation is paramount to the proposed system. The system does not record any personally identifiable information, other than what the artist chooses to include in the metadata of a token. The security of smart contracts face challenges due to the fact that they are deployed on Ethereum and are therefore accessible and transparent. There are disastrous consequences to the vulnerability of smart contracts, as they execute the business logic of this system. This section details well-known vulnerabilities and attacks and discusses the recommended preventive strategy and how they are implemented for this system, with a focus on active attacks. Active attacks concern attempts where the attacker is attempting to alter the live operation of the system, as distinct from passive attacks where the attacker is attempting only to obtain data for later use.

6.1 Underflow/Overflow

If a number is outside the range of a variable type, it will lead to overflow/underflow. EVM assigns fixed size to variables of type integer. This means that an integer has its range. Programmers can inadvertently not check input or variable values under/overflow in the calculation process. These vulnerabilities allow attackers to misuse code and create unexpected logic flows [18]. The typical technique to guard against this is to replace solidity standard math operators with mathematical libraries. OpenZeppelin [48] provides a widely accepted implementation of Safe Math. This system uses OpenZeppelin safe math library to replace all standard mathematics operators with the Safe Math library functions.

6.2 Modifiers

In solidity, a modifier is used to change function behaviour. A function with a public modifier can be called from the contract itself, inherited contract, and other contracts. An external function cannot be called by the member functions of the contract itself. A private function can be called only by the member functions of the contract itself. Internal functions can be called by the contract itself and inherited contract functions. Suppose an attacker gets control of a caller contract by delegating calls to a public function under the context of a caller contract. In that case, the main problem is a combination of insecure visibility modifiers and misuse of delegate calls with arbitrary data. Deferring to the best practice of following the function structure first to check the condition, then update the state of the contract, and finally, interact with other contracts. This avoids most modifier security vulnerabilities if you use the order of condition, behaviour, and interaction.
6.3 Stack Depth

The Ethereum Virtual Machine (EVM) constraints the call stack at a hard cap of 1024. This means that a function will fail if the nested call stack exceeds this. If an attacker recursively calls a constraint 1024 times, it will lead to a send failure as a result of this call stack depth constraint.

6.4 Race Conditions/Front Running

External calls to contracts combined with the multi-user nature of blockchain can potentially lead to users racing code execution and obtaining unexpected states. An Ethereum node pools transactions and packs them into a block when transactions are submitted. As only one miner solves the consensus question, the miner picks which transactions to pool and pack into the block. Only these selected transactions, packed into blocks, are considered valid. The decision to include transactions is generally a matter of price. An attacker can watch the pool of transactions, take transaction data, create a new transaction with a higher gas price, and use this transaction to modify privileges or update a contract’s state. With a higher gas price, this new transaction will be included in the block before the original transaction. There is two users capable of this type of attack; users who modify the gas price and miners. This system implements two mitigate measures to combat and prevent this attack. Firstly each contract has an upper bound of gas price, which prevents attackers from increasing the gas price. Since miners can still order transactions in a new block regardless of gas price, the system implements a commit-reveal pattern. The user sends the transaction with hidden data, a hashed value, and after the transaction is packed into a block, the user sends another transaction with solution details. This prevents miners and other users from knowing the solution details and stops them from creating a new transaction with solution details to perform the front-running attack.

6.5 Reentrancy

A procedure is re-entrant if its execution can be interrupted in the middle, initiated over (reentered), and both runs can complete without any errors in execution. This technique was famously used in the DAO hack to withdraw funds from a crowd-funded contract. This solution utilises the withdrawal design pattern, separating accounting and transfer logic to ensure that all internal state changes are complete before invoking an external call. This also aids in designing away the potential for cross function race conditions like internal functionals that call external functions and targets potential denial of service attacks. The called contract could have a fallback function that reverts, which would halt contract execution,
meaning that subsequent necessary operations would not be executed. The withdrawal design pattern favours pull over push payments and avoids denial of service attacks that aim to target a block’s gas limit. Royalty payments could have been made by looping through an array of shareholders and pushing payments to them for their convenience. Attackers could easily take advantage of this by creating corrupt account recipients to cause the function to hit gas limits, making a token contract unusable, which would be catastrophic for all token use cases.

In order to prevent this kind of reentrancy vulnerabilities in both the token and market smart contracts, this system utilises a number of methods. The first is using the built-in transfer function when sending ether to external contracts. This only sends 2300 gas, which is insufficient to support the destination address/contract by calling another contract (i.e. reentering the sending contract). The second measure is to ensure all state-changing logic happens before matic is sent in an external call. This follows the checks-effects-interaction pattern and is the practice of placing any code with external calls to unknown addresses as the last operation in a localised function or code execution.

6.6 Denial of Service Attack (DOS)

A DDOS attack aims to make a contract un-functional or irresponsive for a certain period or even forever, blocking the users of contracts. For example, if the market contract were to mint and distribute all token shares once a contract is created, it would loop over an array of shareholders and mint and distribute tokens for each. If array size becomes super huge, which is highly probable as users are likely to want a large number of tokens to sell them in small doses, the iteration will lead to exhaustion of GAS. Thus, this functionality would always fail.

To avoid exhaustion of GAS, this solution proposes:

- Using the withdrawal pattern, which requires participants to withdraw their own funds and using lazy minting for shareholders to mint their own tokens.

- It provides a fail-safe pattern to avoid malfunction of the contract owner, by create a multi-sig contract, where contract owner(s) are signers. It uses a time lock which allows contract execution after a certain period and includes time out logic for progressive state judgment.

- There is also potential for the owner and operator of the market contract to act as an operation account to solve contract malfunction problems caused by DoS attack.
6.7 Man-in-the-Middle

There is potential for a man-in-the-middle attack during the key exchange process for decrypting metadata files. This type of attack involves the attack intercepting and altering messages between two parties in direct communication. The marketplace attempts mitigation of such attacks by ensuring data is encrypted with the receiving parties' public key, for data privacy and signed with the sending parties' private key for message integrity.
7  Conclusion

The reformation of the management of music copyright is shown in this paper to be important to the creators of music for their continued creativity and prosperity. The delays and lack of transparency in the current model imply that a new approach is warranted. The proposed decentralized application and tokens provide a more artist-focused approach to copyright management by eliminating the need for intermediaries and providing transparency in both the sale of copyright and payment of royalties.

7.1 Research Objectives

A number of research objectives were outlined at the start of this paper, with the view of evaluating their feasibility and relevance in the proposed solution. They are assessed under the headings below.

7.1.1 Transparency

The nature of blockchain development provides a fully available and immutable transaction history. The open access to the Ethereum network means that everyone, not only professional musicians or business people, can register their copyright, ensuring that anyone who can create a melody has the ability to manage the copyright for it.

This implementation also makes a conscious effort to include clarifying information in transactions, with payments referencing the license under which they are due and storing voucher details in the market contract about the token(s) to be minted, as seen in section 5.3. This is all secondary to the implementation of events that are published by both the token and market contract containing the details of the contract transactions, section 5.1, 5.2—allowing them to be thoroughly recorded on the blockchain for easy viewing on blockchain explorers like polyscan. The application also strives to showcase details of all transactions being approved or signed by implementing wallet readable naming, storage and data structures, section 5.1.

7.1.2 Data Security and Integrity

The objective of this requirement is to ensure that music copyright information and token metadata are kept secure. The built-in security of the IPFS’s content addressing system ensures that metadata and audio files can not be swapped out, section 2.4. This means that the integrity of the copyright reference will remain intact. The proposed system also provides additional security for the audio file in
the form of encryption and steganography, section 5.4.2. These additional features require centralization of the key protection centre and the steganographic library interaction, so they are optional for the user to implement under their preference for the trade-off between decentralization and extra security.

7.1.3 Interface Usability
The interface focused on abstracting away the finer details of contract deployment, token minting, metadata creation, IPFS interaction and payment schemes. This abstraction includes ensuring that metadata creation and uploading to IPFS are managed along with the creation of vouchers upon contract deployment. The client application manages all abstraction aspects, and that token management is maintained by the application as well, section 4.3. Users are not required to manage their public and private keys and can rely on a user-friendly wallet interface to interact with the application for this functionality, 5.3.

7.1.4 Ownership Tracking and Trading
The tokenization process detailed in section 4.2 details the representation of shares in the copyright as tokens. How the application facilitates the creation, management, and trading of tokens is subsequently detailed in section 4.3. The autonomy of ownership is discussed 4.2 and implemented through access control mechanisms.

7.1.5 Royalty Payment and Distribution
The facilitating of payments and the pull payment distribution method are detailed in section 4.2.4, with the necessity for a pull payment scheme seconded in section 5.2. All transaction history, including payments to the contract and payment withdrawal, are easily verifiable by browsing a contract on polyscan.

7.1.6 Self-Sovereignty
Token contracts are fully autonomous from market contracts and are usable without the need for interaction with the market contract {}. Owners of tokens can mint, buy or sell, in accordance with their access privileges on other platforms or by direct interaction with the contact. The application also eliminates the need for a middleman to collect and distribute royalties.

7.2 Future Work
This paper strives to present a comprehensive design and proof of concept implementation of a music copyright management system using Ethereum smart
contracts. There are still improvements that can be made as the technology matures and industry practices cement themselves.

7.2.1 Integration with distributed music streaming services

On Ethereum, smart contracts are accessible and transparent – like open APIs – so the proposed DApp can even include a smart contract that someone else has written. Compatibility is, in this way, built into Ethereum products and specifically in this design because Ethereum best practices and community-approved approaches were implemented. This means that music revenue generation products, like decentralized streaming services, can be easily integrated to work with the project. Allowing other projects to build on the objectives of this one. [60] and [63] describe such distributed music streaming services.

7.2.2 Ethereum Proof of Stake

Ethereum has plans to migrate from proof of work consensus to proof of stake. This would reduce the cost of operating on the mainnet and may drive traffic away from second-layer solutions like Polygon. As Polygon is built on top of Ethereum, all state information is accessible on both networks and tokens are bridgeable, section 4.4.2. This means that the application could migrate from Polygon to mainnet if Ethereum mainnet’s adoption of PoS renders Polygon unnecessary.
Bibliography


[17] Ikuya Takishima, Ethereum: the ultimate guide to the world of Ethereum, [Ikuya Takishima], 2017


[73] Peter Murray, Nate Welch and Joe Messerman, eip-1167: Minimal Proxy Contract, https://eips.ethereum.org/EIPS/eip-1167

[74] Polygon Faucet, https://faucet.polygon.technology/


[78] Solidity, https://docs.soliditylang.org/en/v0.8.13/


[80] React JS, https://reactjs.org/


Appendix A: Source Code

The source code for the proof-of-concept system outlined in this paper can be found on GitHub at https://github.com/ruthbrennankk/music_copyright_management

The final commit hash is f07926200bbd2496b3dbf93df609329a46054bea