Special Paper
방송공학회논문지 제29권 제7호, 2024년 12월 (JBE Vol. 29, No. 7, December 2024)
https://doi.org/10.5909/JBE.2024.29.7.1123
ISSN 2287-9137 (Online) ISSN 1226-7953 (Print)

Smart Contract of Trustworthy Music Management for WEB Service

Gi Woong Chae[a)], Youngmin Kim[b)], and Sang-Kyun Kim[a)‡]

Abstract

Efficient music data management and fair and prompt distribution of profits to music copyright holders remain long-standing problems in the music industry. Blockchain and smart contracts attract attention as effective technologies to solve these problems. This paper demonstrates the basic architecture for a music management system using blockchain and smart contracts. The functions of music registration (registration of copyright holder), music sales, music usage fee settlement, and finally, music disposal must all be reflected in smart contracts To manage music decentralized and efficiently. Using these smart contracts, we focus on implementing music management services for fair and immediate profit distribution through sequential execution processes between major entities. Gas reduction methods on smart contracts are applied to save gas; its reduction rate reaches about 30%. Through repeated tests, the main functions (e.g., contract creation, music purchase, settlement) were shown to work well, and the stability of the proposed smart contract was proved.

Keyword : Music industry, Blockchain, Smart contract, Online music distribution, Music revenue settlement

Ⅰ. Introduction

In the pre-Internet era, the music industry experienced long-term sustained, with a handful of record labels known as major record labels (Sony BMG, Universal Music Group, Warner Music Group, and EMI). Music was physically distributed so that record labels controlled entire supply chain, like retail CDs. Because of this, most of the value chain and all transaction data were captured by record labels.

With the popularization of the Internet, physical distribution became meaningless. New and disruptive business models, such as NAPSTER's peer-to-peer music-sharing network, have dramatically increased music piracy. As sales in the music industry declined exponentially, the need for restructuring the existing business model became apparent.

One of the significant turning points in the industry was the launch of the iTunes Store, an online platform created by Apple, which has radically changed how we consume music. Songs were no longer a physical commodity, and consumers could purchase music through any Apple device

a) Department of Convergence Software, Myongji University
b) Korea Electronics Technology Institute
‡ Corresponding Author : Sang-Kyun Kim
E-mail: goldmunt@gmail.com
Tel: +82-2-300-0637
ORCID: https://orcid.org/0000-0002-2359-8709
· Manuscript October 14, 2024; Revised November 21, 2024; Accepted November 25, 2024.

without restrictions. However, record labels have not been satisfied with Apple's song pricing, and musicians have not received sufficient royalties from Apple[1].

On-demand streaming platforms such as YouTube, Spotify, and Apple Music are the most used tools for accessing music today. Consumers have faster, simpler, and unlimited access to media. On the other hand, musicians still have little influence over their creations and continue to receive low rewards amid these turning points.

Blockchain enables establishing a transparent economic system through recording music ownership, transparent and rapid distribution of profits, and music management without intermediaries[2-4]. Music creators are guaranteed quick and transparent revenue by automating music registration and revenue distribution using smart contracts[5]. Studies are proposed for fast search and indexing of music using IPFS and blockchain[6-7]. This paper proposes a method to solve the problems of the existing music service system by introducing a smart contract for efficient music management.

The structure of this paper is as follows. Section 2 explains the need for blockchain-based music management (and smart contracts). Section 3 shows the architecture of the music platform based on the blockchain. Section 4 explains how the entities that make up the music management system perform music registration, purchase, and settlement. Section 5 describes the main functions of smart contracts for music management. Section 6 explains how gas consumption can be reduced. Section 7 explains the stability test results of the proposed smart contract. Section 8 discusses some limitations of the proposed method, and then section 9 concludes with some general observations and recommendations for ongoing work.

Ⅱ. Research Background

Music in the digital age is data. Metadata is data of data, that is, information about music. The metadata included with any music can include the copyright owner's terms of use and contact details, making it much easier to find the owner of the music and obtain permission to use it. Gradually deploying copyright data on the blockchain could create one comprehensive music copyright database[8].

Music contains at least two copyrights. One is the copyright of the recording itself related to the performer and the record label, and the other is the copyright to the lyrics and music related to the songwriter or composer and the music publisher. These copyrights can be stored on the blockchain via cryptographic hashes.

The average musician earns \$23.40 for every \$1,000 of their music sold[9]. Net profit is only 2%. Music labels, publishers, and streaming services perpetuate unequal contracts by exploiting the information asymmetry between themselves and content creators. Royalty payments for recordings and music are very slow, taking months or years to reach the bank accounts of copyright holders. Fees may have been deducted from multiple performing rights associations until the money reached the rights holders[10].

Blockchain technology has the potential to change this situation in many ways. In general, micropayments are possible due to the low transaction costs of cryptocurrencies with a wide range of amounts up to 8 decimal places. Also, content creators will be able to receive a very small amount of 'tip'[11].

The term smart contract refers to a blockchain-based contract with conditional conditions that can be executed automatically, such as settlements and asset transfers[12]. The application could be particularly innovative for the music industry, as it supports automatic royalty payments to artists and other actors holding stakes. Through smart contracts, payments made by consumers when they download, stream, or use music available on the blockchain network do not go through a third party. Instead, all information needed to deliver payments to each shareholder is on-chain and executed as specified in the contract. All contributors to the song will not only receive their stake immediately after the license to the song is granted but any information generated by the transaction will also be made available for data visualization and analysis techniques (e.g., data mining). In other words, smart contracts eliminate the need for content creators to navigate their way through expensive purchasing platforms and financial intermediaries, allowing them to sell their products directly to fans, consequently eliminating all transaction costs[13]. More specifically, through smart contracts, music creators can decide when/how/how much others can use their work at a precise price.

In effect, running a smart contract over a blockchain network eliminates the need for third-party intermediaries to review or verify transactions. With these self-executing contracts, all parties involved are bound by the rules and decisions of the underlying code. Thus, smart contracts could eliminate the need for lawyers and litigation in the not-too-distant future and save content creators money.

This improved business model is especially valuable to smaller content creators who can't make less money without sponsorship from a major record label. With easy trading in an accessible and uncrowded marketplace, new amateur content creators are expected to enter the new blockchain-enabled music industry. This influx of new content creators will lead to a quantitative influx of all published music, expected to benefit fans and content creators.

Ⅲ. Architecture for Music Platform Based on Blockchain

Figure 1 shows the overall architectural design from the point of view of platform users, including music owners, service providers, and end users. The architecture largely consists of a service gateway, data exchange handling, an intelligent data router, off-chain data, on-chain data, and an intelligent engine.

First, the service gateway unit comprises different management services such as DID, music, and access management. Each service type has at least one smart contract to enable different functions. All platform users must have at least one DID to identify themselves through DID management, and each DID is bound to a blockchain account to maintain uniqueness. Realized by a music service agreement and music registration music management, access management grants and confirms specific access rights.

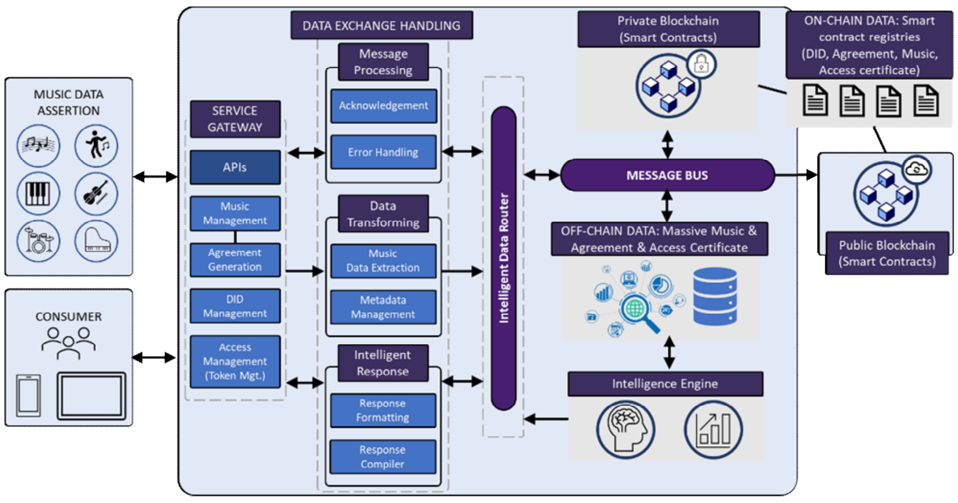

Fig. 1. Music supply and service architecture based on blockchain

Second, the data exchange handling unit helps to exchange messages or refined data between music owners, users, and content providers. The message processing unit is in charge of recognizing message delivery and processing errors. The data transforming unit is responsible for extracting feature values from input music, watermarking, and processing all music-related metadata. In the intelligent response unit, the intelligent engine formats the response from the blockchain to be presented to the user.

Third, the intelligent data distribution unit stores all data input from the user in the blockchain according to the characteristics or delivers the response from the intelligent engine unit to the user.

Fourth, the off-chain data processing unit stores original music files and documents and stores access certificates, including access rights for specific music.

Fifth, the on-chain data processing unit processes on-chain data and business logic by applying blockchain networks and smart contracts to the architecture design. At this time, the blockchain can be used by selecting a public or private blockchain depending on the nature of the system, and the on-chain data includes DID, Agreement, Music, and Access Certificate registries and smart contracts that manage them.

Finally, an intelligent engine unit intelligently determines the user's (music owner, consumer, content provider) requirements (e.g., search, recommendation) and provides an optimal response.

This paper mainly focuses on developing on- and off-chain data processing units.

Ⅳ. Smart Contract-Based Music Management Sequence

The main functions for transparent music purchase and settlement without an intermediary using the smart contract are music upload, music purchase, and music settlement. Each function's main entities are Buyer, Seller, SettlementContract, and Backend.

The Buyer is the music buyer, and the Seller is the music copyright holder or music seller. SettlementContract is a smart contract created by the Seller to distribute music revenue. The Backend stores music and its related information and verifies the validity of the smart contract. In addition, it confirms whether the transaction hash entered into the Backend is generated from a valid contract.

1. Music Upload Sequence

Figure 2 shows the sequence diagram for music registration. The Seller enters the necessary information for music upload and uploads the music file to the Backend. The Backend checks whether the uploaded music has already been registered. If it is not duplicate music, the Seller will verify that all data required to generate the metadata

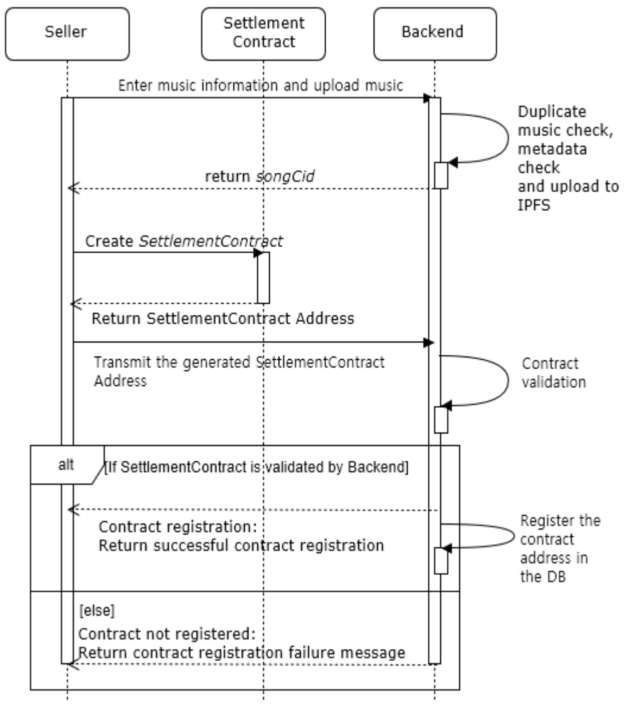
Seller
Settlement Contract
Backend
Enter music information and upload music
Duplicate music check, metadata check and upload to IPFS
return songCid
Create SettlementContract
Return SettlementContract Address
Transmit the generated SettlementContract Address
Contract validation
alt
[If SettlementContract is validated by Backend]
Contract registration: Return successful contract registration
Register the contract address in the DB
[else]
Contract not registered: Return contract registration failure message

Fig. 2. Sequence diagram for music registration

is entered correctly. After verification, the Backend uploads the music and its metadata to IPFS and returns songCid, the IPFS CID created after metadata upload, to the Seller.

The Seller creates a SettlementContract by inputting songCid and other necessary data. When the smart contract address (SettlementContract) is transmitted to the Backend, the Backend verifies the contract's validity using the values stored in the contract of the transmitted address.

As a result of verification, if it is a valid contract, the message "Contract registration successful" is returned, and the contract address is added to the database. The message "contract registration failed" is returned if it is invalid.

2. Music Purchase Sequence

Figure 3 shows the sequence diagram of music purchases. When the Buyer selects the music they want to purchase and makes a purchase request, the Backend returns the SettlementContract address of the music they want to buy and metadata containing the music settlement information. Here, the music settlement metadata includes the wallet address of the copyright holder, the revenue distribution ratio of the copyright holder, and the purchase amount per music. The Buyer checks whether the hash value (i.e., keccak256Hash) of the copyright holder's wallet address and distribution ratio with the keccak256Hash function matches the keccak256Hash value returned from SettlementContract. If the two match, the purchase proceeds with buy(). The reason for comparing the copyright holder's wallet address and revenue distribution ratio from IPFS with the corresponding information from SettlementContract is to ensure that the music requested by the Buyer

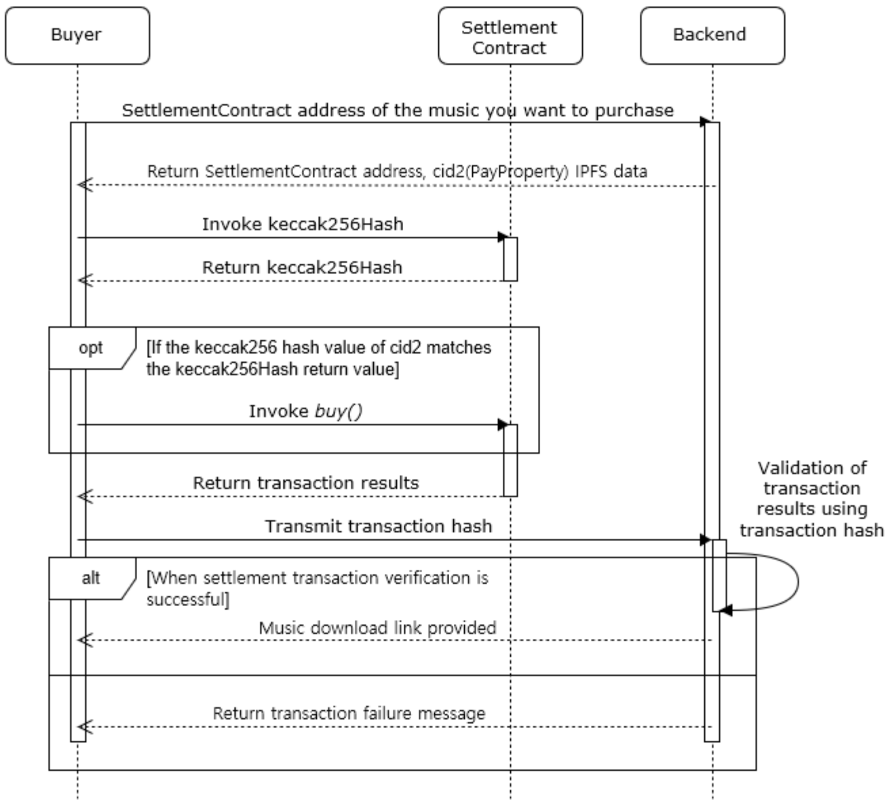
Buyer
Settlement Contract
Backend
SettlementContract address of the music you want to purchase
Return SettlementContract address, cid2(PayProperty) IPFS data
Invoke keccak256Hash
Return keccak256Hash
opt
[If the keccak256 hash value of cid2 matches the keccak256Hash return value]
Invoke buy()
Return transaction results
Transmit transaction hash
Validation of transaction results using transaction hash
alt
[When settlement transaction verification is successful]
Music download link provided
Return transaction failure message

Fig. 3. Sequence diagram for purchasing music

has accurate music settlement information. The Buyer records the transaction record for the purchase on the blockchain and sends the generated transaction hash to the Backend.

The Backend uses the transaction hash to track the transaction record in the blockchain and verifies that the transaction contains valid transaction information. If it is a valid transaction, a link to download the music is provided to the Buyer, and transaction failure is returned if it is invalid.

3. Music Revenue Settlement Sequence

Figure 4 shows the sequence of music revenue settlement. When desired, the Seller can call the settle() function, an internal function of SettlementContract, and receive a settling of revenue corresponding to the distribution ratio of copyright holders when registering music. If it is not the author, SettlementContract raises an error.

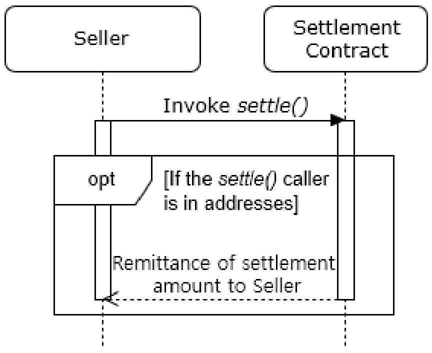
Seller
Settlement Contract
Invoke *settle()*
opt
[If the *settle()* caller is in addresses]
Remittance of settlement amount to Seller

Fig. 4. Sequence diagram for music revenue settlement

4. Music Destruction Sequence

Figure 5 shows the sequence of music destruction. Among Sellers, only the creator (owner) of the SettlementContract can destroy the contract through the destroy() function that destroys the contract. Even if the subject that executed the destroy() function is the owner of the SettlementContract, if the balance inside the SettlementContract is higher than the purchase price of the music, the contract cannot be destroyed. The reason is that when the destroy() function is executed, the contract balance is moved to the wallet of the caller of destroy(). By exploiting this, the contract owner can steal all the balance within the SettlementContract. This process is to prepare for the time when all balances in the SettlementContract cannot be settled due to unforeseen circumstances.

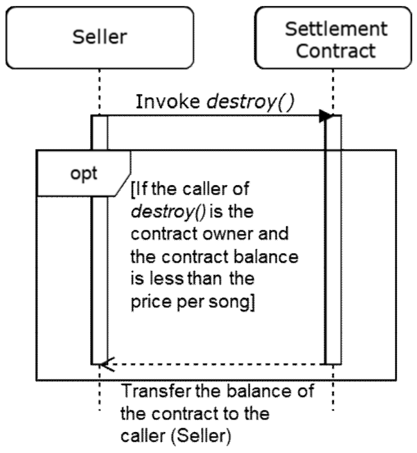
Seller
Settlement Contract
Invoke *destroy()*
opt
[If the caller of *destroy()* is the contract owner and the contract balance is less than the price per song]
Transfer the balance of the contract to the caller (Seller)

Fig. 5. Sequence diagram for music destruction

Ⅴ. Implementations of Smart Contract

Using SettlementContract, the purchaser can send the purchase amount of music to SettlementContract. The amount accumulated in the contract wallet is settled and paid at the distribution ratio promised when the registered copyright holder requests settlement.

The Buyer can send the amount to purchase music through the SettlementContract through the buy() (Figure 6②) function. The music copyright holder (i.e., Seller) settles with SettlementContract's settle() function (Figure 6③) when desired.

The smart contract creator owns the SettlementContract, and the smart contract can be destroyed if the balance of the contract wallet is less than the specified "music pur

```
  // SPDX-License-Identifier: AGPL-3.0
pragma solidity ^0.8.0.0;

contract SettlementContract {
  uint256 public price;
⑤··· uint256 public cumulativeSales = 0;
  address public owner;
  bytes32 public keccak256Hash;
  bytes32[2] public songCid;

  mapping (address => copyrightHolder) public
  copyrightHolders;
⑥··· struct copyrightHolder {
    uint256 proportion;
④···    uint256 count;
  }

  event logBuyerInfo(address buyer, bytes32[2]
  songCid, uint256 amount);
②··· function buy() public payable {
    require(
      price == msg.value &&
      cumulativeSales < type (uint256).max
    );
    cumulativeSales += 1;
    emit logBuyerInfo(msg.sender, songCid, price);
  }

  event logRecieverInfo(address receiver, bytes32[2]
  songCid, uint256 amount);
③··· function settle() public {
    copyrightHolder memory caller =
  copyrightHolders[msg.sender];
     require(
      caller.proportion > 0 &&
      cumulativeSales - caller.count > 0 &&
      cumulativeSales < type(uint256).max
    );

    uint amount = price / 10000 * caller.proportion
  * (cumulativeSales - caller.count);
    caller.count = cumulativeSales;
⑦···    payable(msg.sender).transfer(amount);
    emit logRecieverInfo(msg.sender, songCid,
  amount);
  }

⑧···  function destroy() public {
    require(msg.sender == owner &&
    address(this).balance < price);
    selfdestruct(payable(owner));
  }

①··· constructor(address[] memory _addresses,
  uint256[] memory _proportions, bytes32[2]
  memory _songCid, uint256 _price) {
    require(_price % 10000 == 0);
    owner = msg.sender;
    keccak256Hash = keccak256
  (abi.encode(_addresses, _proportions));
    songCid = _songCid;
    price = _price;
    for(uint i=0; i<_addresses.length; i++) {
      copyrightHolders[_addresses[i]] =
  copyrightHolder(_proportions[i], 0);
    }
  }
}
```

Fig. 6. Smart contract for music management

chase price". The remaining balance in the destroyed SettlementContract is transferred to the wallet address of the smart contract owner. This restriction to destroy the smart contract prevents an accident in which the contract owner kills the contract without the other registered copyright holder's consent and monopolizes the contract's balance.

1. Creation of Smart Contract

SettlementContract is created by the representative of the music copyright holder or the music registrant. The creator owns the contract and bears the distribution cost.

The constructor() (Figure 6①) is a constructor function and works only once when creating a contract. The input data of constructor() includes the wallet address of each copyright holder, the profit distribution ratio of each copyright holder, the title of the music to be registered, the IPFS CID containing music information such as lyrics, album, composer, and singer, and the price of the music. The cost of music shall be a multiple of 10,000. Also, the sum of the revenue share ratio of each copyright holder is always 10,000. There is still no way to safely store or use decimals in the solidity language (based on 0.8.0). Therefore, input the profit distribution ratio by converting it to a natural number value multiplied by 10,000 by the percentage (0.00%) that allows each distribution ratio to two decimal places. songCid receives a 46-character string that exceeds 32 bytes that the bytes32 data type can contain, and it must also include various kinds of IPFS hashes.

The wallet address of the input author and the profit distribution ratio are stored in a 1:1 mapping as a key and value, respectively, in a hash table data structure called solidity mapping through a loop. The mapping value is declared a copyrightHolder (Figure 6⑥) structure. The structure includes a variable count (Figure 6④) that stores the number of times it has been settled and the profit-sharing ratio.

2. Purchase

The purchaser can transfer the music price to the contract wallet through the buy() (Figure 6②) of the contract corresponding to the desired music. If the remittance amount does not match the music sales price or the cumulative sales amount is the maximum value of the data type, it is reverted (undone). The overflow of the corresponding variable can be prevented using this restriction.

If the above conditions are satisfied, the remittance to the contract wallet (CA) proceeds. When remittance proceeds, cumulativeSales increases by 1. After that, an event occurs, and the purchaser's wallet address, songCid of the purchased contract, and purchase amount are recorded in the event log of the transaction.

Since the maximum value of the data type of the cumulative sales variable, cumulativeSales is set, when the incremental sales reach the maximum value, no further purchases can be made. The overflow occurs when an operation that exceeds the maximum value of the data type is executed. If overflow occurs, the contract owner must discard the current contract and create a new contract with the same content.

3. Settlement

Suppose the music copyright holder intends to settle through the music contract. In that case, the copyright holder can execute the settle() (Figure 6③) function to receive the music revenue settled in their wallet according to a predetermined ratio of the amount sold. The calculation method of the settlement amount is as follows.

$$settlementrevenue = price \times (proportion \div 10000) \times (cumulativeSales - count) \tag{1}$$

, where the price is the music price and proportion/10,000 is the settlement ratio. At this time, the proportion must be less than or equal to 10,000, so the settlement ratio is calculated as a float number. The cumulativeSales is the cumulative sales volume, and the count is the number of settlements. The settlement ratio per piece of music is multiplied by the number of times it needs to be settled. Since caller.count (Figure 6④), which indicates the number of times settlement has been received at the address after settlement, is updated with the value of cumulativeSales (Figure 6⑤). The duplicate settlement does not occur even if sales and cumulative sales continue to increase. The calculated settlement revenue is transmitted to the caller of the settle() (Figure 6③) (the copyright holder) through the transfer() (Figure 6⑦) function. After settlement, an event occurs, and the address of the caller of the settlement function, IPFS CID of settlement music, and settlement revenue are recorded on the blockchain in the form of an event log.

4. Destroy of Smart Contract

Only the settlement contract owner of the Settlemet-Contract can destroy the contract, which is done through the destroy() function (Figure 6⑧). Also, the destroy() function can only be executed when the caller is the contract owner and the balance in the contract wallet is lower than the music price. This restriction prevents the forced destruction of smart contracts in the unsettled state. In other words, the contract owner cannot monopolize the revenue from music sales through the destruction of the contract when the respective copyright holders have not yet been settled.

The contract destruction is carried out due to a breakdown in an agreement between the copyright holders. Or it is necessary to recreate the contract after destruction due to contract defects (e.g., incorrect address input, incorrect distribution ratio input, reaching the maximum number of cumulative purchases of the contract).

Ⅵ. Gas Consumption Reduction

The cost of gas is one of the major problems in implementing blockchain systems or services adopting the smart contract. Therefore, developers need to design smart contracts to minimize gas consumption. Some points on how to save gas consumption are described as follows:

- Reduce Smart Contract's capacity by deleting the required statement's error message.
- Using struct and mapping can save gas costs compared to defining general state variables, so struct and mapping are used to store the wallet address of the copyright owner, distribution ratio, and settlement count.
- Refrained from using modifiers because using modifiers cost a lot of gas.
- If the require statement is used unnecessarily several times in a function, the number of executions of the require function may increase, which may result in additional gas costs.

Table 1 shows the gas consumption results before and after applying the gas reduction methods described above. The results show that about 30% gas reduction was achieved.

Ⅶ. Experiments

1. Objective

There are no exact evaluation criteria for testing smart contracts. Therefore, this experiment focused on testing the stability of the proposed smart contract. The experiment checks whether each function in the contract works precisely and whether there is no error even if the function is repeatedly executed.

2. Experimental setups

The experiment used Truffle.js, an Ethereum development framework, and mocha.js, an experimental framework. The experimental environment used 'Ganache', which creates a local Ethereum blockchain network environment.

The hardware setups for the experiment are as follows.

- OS: Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-57-generic x86_64)
- CPU: 11th Gen Intel(R) Core (TM) i7-11700K @ 3.60GHz
- GPU: NVIDIA Corporation GA106 [GeForce RTX 3060 Lite Hash Rate]

Table 1. Gas consumption report before and after applying the gas reduction methods

Tasks	Cost(unit: gas)	Before applying the gas reduction methods	After applying the gas reduction methods
Creation of contract	Transaction cost[1]	1,128,730	790,924
	Execution cost[2]	976,720	662,406
Execution of buy()	Transaction cost	53,349	36,249
	Execution cost	32,285	15,185
Execution of Settle()	Transaction cost	66,075	65,722
	Execution cost	45,011	44,658
Total		2,302,170	1,615,144

1) The total cost of all gas required to process a transaction.

2) The cost of running a virtual machine, which is the gas cost of running a function or action of a smart contract. Include virtual gas consumption for all compute operations and resource access that occur within the execution function.

- memory: 16GB
- storage: 1TB SSD

Other experimental setups include as follows.
- experimental Platform: Truffle Framework 5.8.0[3)]
- blockchain network: Ganache localNet

The gas setting value of the Ganache local blockchain is as follows.
- gas price[4)]: 20 Gwei
- gas limit[5)]: 6721975 Wei
- call gas limit[6)]: 9007199254740991 Wei

3. Experimental results

We created 10,000 wallets using Ganache for the experiment and initially allocated 10,000 ETH to each wallet. When a contract is created, six people are registered as copyright holders. One of the accounts created is assumed to be the Buyer's account. When a contract is created every time, the Buyer executes the purchase function in the created contract to transfer the amount corresponding to the price of the sound source to the contract.

Figure 7 shows the experimental process flow diagram described as follows.

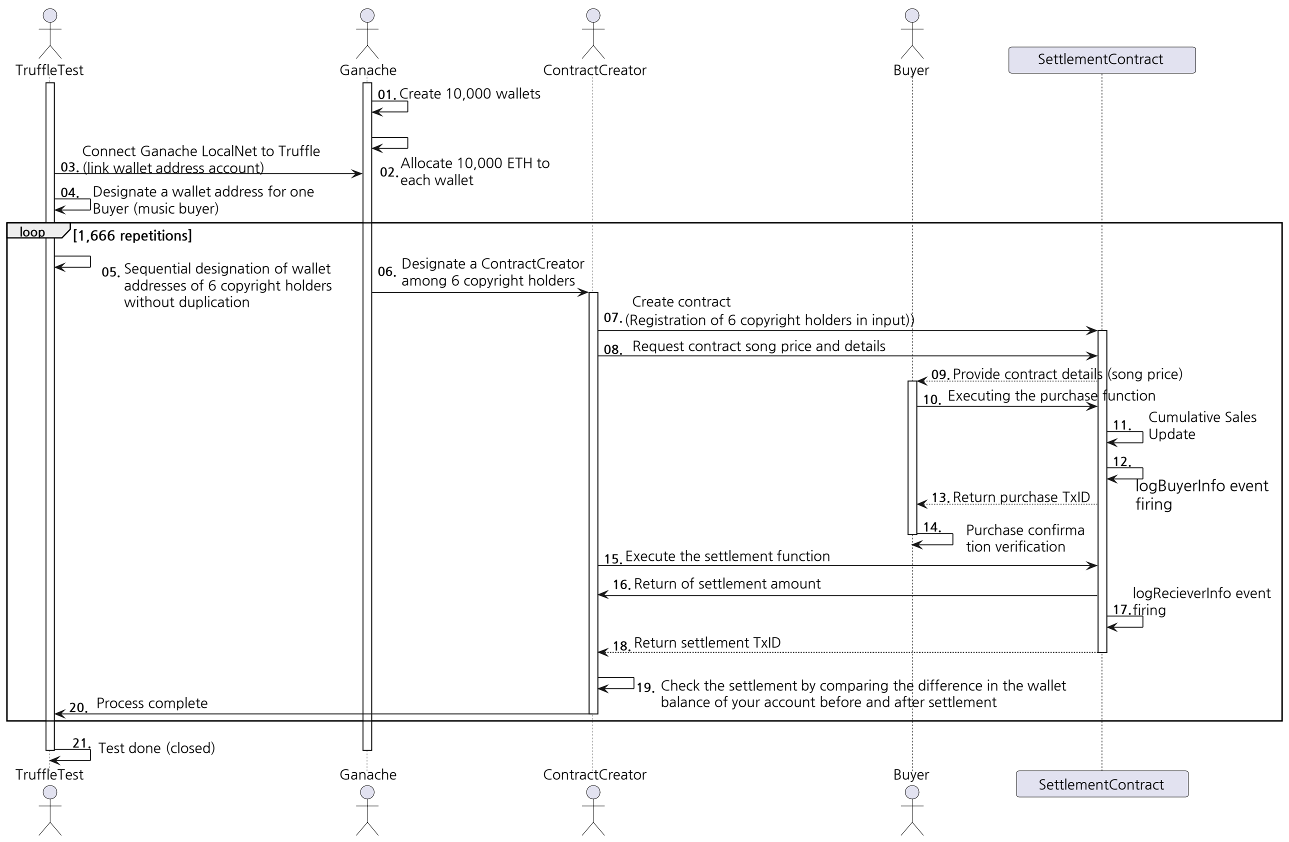
TruffleTest
Ganache
ContractCreator
Buyer
SettlementContract
01. Create 10,000 wallets
02. Allocate 10,000 ETH to each wallet
03. Connect Ganache LocalNet to Truffle (link wallet address account)
04. Designate a wallet address for one Buyer (music buyer)
loop [1,666 repetitions]
05. Sequential designation of wallet addresses of 6 copyright holders without duplication
06. Designate a ContractCreator among 6 copyright holders
07. Create contract (Registration of 6 copyright holders in input))
08. Request contract song price and details
09. Provide contract details (song price)
10. Executing the purchase function
11. Cumulative Sales Update
12. logBuyerInfo event firing
13. Return purchase TxID
14. Purchase confirma tion verification
15. Execute the settlement function
16. Return of settlement amount
17. logRecieverInfo event firing
18. Return settlement TxID
19. Check the settlement by comparing the difference in the wallet balance of your account before and after settlement
20. Process complete
21. Test done (closed)
TruffleTest
Ganache
ContractCreator
Buyer
SettlementContract

Fig. 7. Diagram of the experimental process

3) Truffle Framework: node.js-based smart contract production and experiment environment.
4) price per 1 gas set by Ganache.
5) Amount of gas that can be consumed in one transaction.
6) Limit of gas used when handling external calls such as external contract function calls in Ethereum.

1. Create 10,000 wallets using Ganache (local Ethereum blockchain environment).
2. Allocate 10,000 ETH per account to the created wallet.
3. Connect the Ganache local network to the Truffle test framework.
4. The Buyer (music buyer) designates the wallet address of one person.

[Repeat 5-20 1666 times.]

5. Designate the wallet addresses of 6 copyright holders (original authors) sequentially without duplication. For example, if wallets 1, 2, 3, 4, 5, and 6 served as copyright holders before, wallets 7, 8, 9, 10, 11, and 12 act as copyright holders next time.
6. Designate one ContractCreator (contract creator/distributor) among six copyright holders.
7. The ContractCreator creates a contract by inserting six people, including himself, into the list of copyright holders.
8. The Buyer requests the SettlemetContract for the contract's song price and music information to test the purchasing function.
9. The SettlemetContract returns the requested song price and music details to the Buyer.
10. The Buyer executes the purchase function with the price matched to the price of the song provided by the SettlemetContract.
11. When the purchase function is executed, the SettlemetContract increases the cumulative sales of the corresponding song in the contract by 1.
12. Execute logBuyerInfo() to generate an event log on the blockchain.
13. After the purchase function execution, the SettlemetContract returns TransactionID containing the transaction record of the buy() function execution.
14. The Buyer verifies transaction validity using TransactionID provided by the SettlemetContract. The Buyer is set to perform transaction validation logic, and the validation process follows the given validation procedure in the test code.
15. The ContractCreator executes the settlement function to request settlement of the proceeds in the contract.
16. The SettlemetContract checks whether the target of executing the settlement function is one of the copyright holders, settles the proceeds stored inside the contract by the settlement ratio entered when creating the contract, and returns them to the ContractCreator.
17. The SettlemetContract executes logRecieverInfo() to generate an event log for settlement.
18. The SettlemetContract returns TransactionID to ContractCreator after executing the settlement function.
19. The ContractCreator compares the balance before executing the settlement function with the balance after execution to check if the settlement amount has entered its wallet.
20. TruffleTest confirms that the testing process is complete. Repeat the test process from step 5.
21. End the test after completing 1,666 loops.

The contract creation, purchase, and settlement tests were repeated 1,666 times, and about 28 minutes were consumed. As a result of conducting repeated tests to confirm the stability of the contract, it was confirmed that the test was completed. In other words, no errors occurred within the test environment even after repeated use, purchase, and contract settlement.

Ⅷ. Discussion

There are several limitations exist in the proposed smart contract system. There is a limit to keeping the wallet address of the music copyright holder and the profit distribution ratio of the copyright holder completely private. The reason is that the copyright owner's wallet address and revenue distribution ratio are permanently stored in an immutable state in the contract. Due to the nature of Ethereum smart contracts, the information contained in the contract code is open to everyone. In the current Solidity language, there is no way to make the state variable included in the contract code completely private.

In addition, when the copyright owner of the music creates a settlement smart contract for his song, there is a limit that he can no longer receive a settlement for the song if he loses control and ownership of the wallet address he wrote down. Suppose the music copyright holder loses their wallet for any reason or changes their address and discards the previous wallet. In that case, the copyright holder cannot receive revenue from their song forever.

The currently proposed system has limitations in not achieving complete decentralization in providing songs after music settlement. Until the settlement of music revenue, reliable transactions are carried out through the blockchain network without intermediaries, so it can be considered decentralized. Still, valid transactions are determined through a centralized backend system after settlement. In addition, it was not a fully decentralized music trading platform in that it received a song download link from the backend system.

In addition, there is an unclear problem of whether the contract is legally valid. Currently, the validity period of copyright in the United States is 70 years after death, but theoretically, contracts can remain on the blockchain even after 70 years after death. In addition, smart contracts are not legally proven transaction systems, and smart contract codes on the blockchain appear the same regardless of country, so there are insufficient plans for dealing with different copyright laws in each country.

Ⅸ. Conclusions and Future Work

Music data management and fair and prompt revenue distribution to copyright holders remain long-standing problems in the music industry. Blockchain and smart contracts attract attention as effective technologies to solve these problems. This paper described a concrete implementation example of a smart contract combined with a blockchain for effective music management. Research on using NFT (Non-Fungible Token) for music ownership transfer should be continued further to vitalize more transparent revenue distribution and copyright protection.

References

[1] Baumeister, H., Koch, N. and Mandel, L., Towards a UML Extension for Hypermedia Design. in UML 1999, (1999), 614-629.

[2] Beck, K. Extreme Programming Explained. Addison-Wesley, 1999.

[3] Burdman, J. Collaborative Web Development. Addison-Wesley, 1999.

[4] Ceri, S., Fraternali, P. and Bongio, A., “Web Modeling Language (WebML): a modeling language for designing Web sites,” Proceedings of WWW9 Conference, (Amsterdam, 2000).

[5] Conallen, J. Building Web Applications with UML. Addison-Wesley, 1999.

[6] Kyoung-Sik Lee, Sang-Kyun Kim, “Music Source and Signature Storage Method using Blockchain and Distributed Storage System,” JOURNAL OF BROADCAST ENGINEERING, 24(6), 956-964, 2019.

[7] Sang-Kyun Kim, Kyoung-Sik Lee, “Music Source Signature Indexing Method for Quick Search,” JOURNAL OF BROADCAST ENGINEERING, 26(3), 321-326, 2021.

[8] O’Dair, Marcus, Zuleika Beaven, David Neilson, Richard Osborne, and Paul Pacifico. 2016. “Music On The Blockchain.” Blockchain for Creative Industries (BCI), Middlesex University.

[9] C. Byun, The Economics of the Popular Music Industry: Modelling from Microeconomic Theory and Industrial Organization 95 (2014).

[10] Rethink music. 2015, Fair Music: Transparency and Payment Flows in the Music Industry, Berklee Institute of Creative Entrepreneurship, Accessed 22.10.15: static1.squarespace.com/static/552c0535e4b0afcbed88dc53/t/55d0da1ae4b06bd4bea8c86c/1439750682446/rethink_music_fairness_transparency_final.pdf.

[11] Tapscott D, Tapscott A., “Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money,” Business and the World, Penguin Random House: New York, 2016.

[12] Iansiti, Marco, and Karim R. Lakhani. 2017. “The Truth About Blockchain.” Harvard Business Review.

[13] Yessi Bello Perez, Imogen Heap: Decentralizing the Music Industry with Blockchain, Mycelia For Music, http://myceliaformusic.org/2016/05/14/imogen-heap-decentralising-the-music-industry-with-blockchain/, [https://perma.cc/LF2T-GYBK] (last visited Nov. 11, 2022) (on file with the Harvard Law School Library).

Introduction Authors

Gi Woong Chae

- 2020. 02. ~ Current : Student, Department of Convergence Software, Myongji University
- 2022. 07. ~ 2024. 07. : Undergraduate Research Assistant, UX Media Lab, Myongji University
- ORCID : https://orcid.org/0009-0001-0530-3508
- Research interests : Blockchain, Smart Contract, Blockchain-Based Decentralized Identity and Data Management, Artificial Intelligence, VR/XR

Youngmin Kim

- 2005. : B.S. in School of Electrical Engineering, Seoul National University
- 2011. : Ph.D in School of Electrical Engineering, Seoul National University (Integrated M.S and Ph.D)
- 2011. 3. ~ 2011. 10. : Postdoctoral Research Fellow, Seoul National University
- 2011. 10. ~ Current : Principal Research Engineer, Korea Electronics Technology Institute
- ORCID : https://orcid.org/0009-0002-2941-4197
- Research interests : Three-dimensiona display including hologrpahy, visual fatigue associated with three-dimensional display, deep learning technology based on massive data, copyright protection (sound source, xr glass), and multimodal AI applications (language and image)

Sang-Kyun Kim

- 1997. : Computer Science, Univ. of Iowa, B.S.(1991), M.S.(1995), PhD(1997)
- 1997. 03. ~ 2007. 02. : Professional Researcher, Multimedia Lab. of Samsung Advanced Institute of Technology
- 2007. 03. ~ 2016. 02. : Professor of Computer Engineering, Myongji University
- 2016. 03. ~ Current : Professor of Data Technology, Myongji University
- ORCID : https://orcid.org/0000-0002-2359-8709
- Research interests : Digital Content(image, video and audio) analysis and management, 4D media, Blockchain, VR, Internet of Things and multimedia standardization