NFT Minting System on Cardano Blockchain

M. Tran[1] and P. Dzurenda[1]

[1]Department of Telecommunications, Brno University of Technology, Brno, Czech Republic

E-mail: xtranm00@vutbr.cz, dzurenda@vut.cz

Abstract—Nowadays almost everything is digitalized and online. This can cause many security issues for our data. They can be leaked, misused, or even compromised. Therefore, we need technology, which can secure digital assets. The solution to this problem can be blockchain technology and Non-Fungible Tokens (NFT). This article presents our open-source implementation of an automated NFT minting system on the Cardano blockchain. We research the capabilities of NFTs on Cardano and possibilities to create an NFT minting system similar to other successful but proprietary NFT minting systems. Our system is built on Cardano command-line interface commands in combination with Python language for easy handling of the minting process.

Keywords—NFT, Non-Fungible Tokens, Minting, Cardano, Blockchain, Python, Cryptocurrency

1. INTRODUCTION

In the present world, we need technology that combines safety, transparency, decentralization, and undestroyable proof of ownership of digital assets. Today every digital asset is either saved on just one server or multiple servers. Data of the assets can be changed, deleted, or multiplied by one entity. The owner of digital assets has no guarantee of asset's safety or immutability of their characteristics or uniqueness. This problem can be resolved by blockchain technology and Non-Fungible Tokens (NFT) [1]. A decentralized blockchain with a public digital ledger ensures that the transactions and its data that has been written to a block cannot be changed or deleted. NFT ensures that the asset is unique and irreplaceable. The ownership and characteristics of an asset, for example, digital art or collectibles, can be safely and easily checked on the ledger itself.

In this paper, we implemented an NFT minting system that is fully compatible with the Cardano blockchain and is open-source. This system allows us to set our own policy for NFT minting, define NFTs properties via metadata, and finally, mint and send an NFT to the blockchain. The implemented system is compatible with both testnet and mainnet. The only thing needed to change between them is to adjust one parameter in the minting transaction. Diagram of our system can be seen in Figure 1.

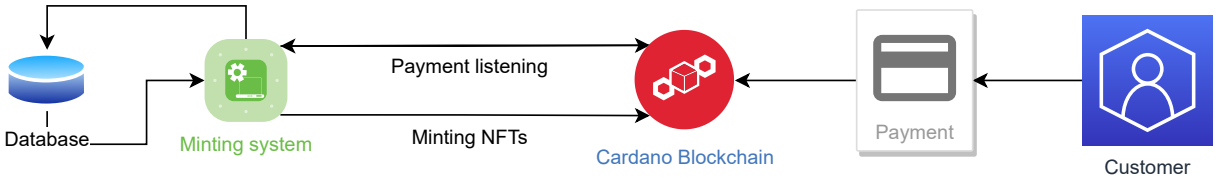

Figure 1: General diagram of our NFT minting system.

2. NFT MINTING SYSTEM

In Cardano, every interaction with the chain is being done with a transaction through a node. Therefore, the most important prerequisite is a functional Cardano node. Furthermore, we need two Cardano wallets, one cardano-cli type wallet with some ADA to pay for transaction fees, and one Cardano Wallet type wallet for payment listening.

We use Cardano Command Line Interface (CLI) on Cardano node with a combination of Python programming language to create an automated NFT minting system which includes functions for creating policyID, creating metadata from NFT metadata database, the configuration of transactions, payment listening, and refunding change. The system listens for incoming payments and corresponds with creating and sending NFT or refunding a customer. The flow diagram of the system core can be seen in Figure 2.

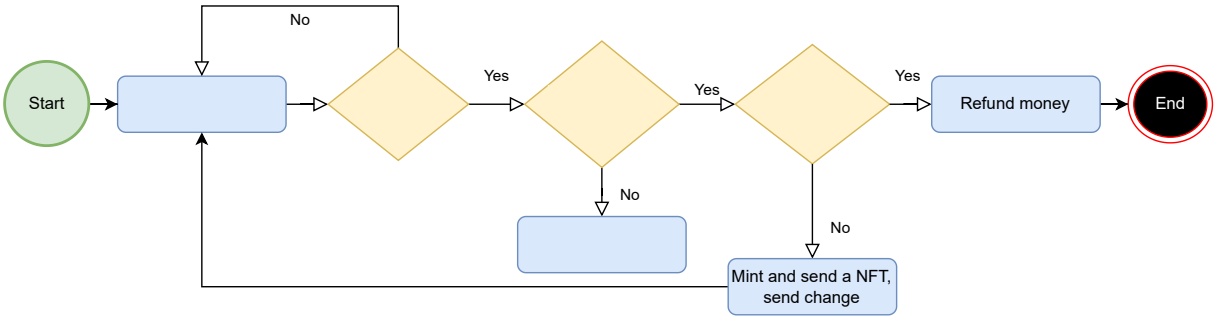

Figure 2: Flow diagram of NFT minting system core.

2.1. PolicyID and policy script

PolicyID serves to distinguish NFTs from each other. Value of policyID is derived from a policy script, which is a simple JavaScript Object Notation (JSON) file predefined by Cardano and is used for defining who can mint NFT under the policyID and it also defines a time window when these NFTs can be minted. At the end of the time window, the policy is locked and nobody can mint any NFT under the same policyID ever again. This is the main principle to guarantee the scarcity of an asset. To create a policy script, we firstly generate a policy verification key and signing key. This is done via Cardano node CLI. Next, we generate a hash of the verification key. After that, we define the time window. This is done by getting the current blockchain slot number and adding a number to it. Approximately every second one slot passes [2]. In our case, the time window includes 604800 slots, which corresponds to one week of the minting period. Now, we have everything we need to build a policy script. Our script file is depicted in Listing 1. Finally, we compute the policyID by hashing the policy script. We used Python library subprocess to execute all CLI commands from our Python application itself.

2.2. NFT Metadata

Metadata stores policyID and other information such as names of asset and InterPlanetary File System (IPFS) link to the image that will be displayed into an NFT in JSON format. We defined the metadata file as it can be seen in Listing 2.

```
policyScriptRaw={
        "type": "all",
        "scripts":
        [{
            "type": "before",
            "slot": int(slot)
            },
            {
            "type": "sig",
            "keyHash": keyHash
        }]
}
```

Listing 1: Policy script.

```
metadataRaw={
    "721": {
        policyID: {
            assetName: {
            "description": description,
            "name": name,
            "id": int(id),
            "image": "ipfs://"+ipfs
            }
        }
    }
}
```

Listing 2: NFT Metadata.

Because of the IPFS link and custom attributes, we populate the metadata database with those data before starting the payment listening and minting process. We used Comma-Separated Values (CSV) format to do so. From the database, metadata files can be generated and used to mint NFT. We use JSON and pandas library to create metadata files.

2.3. Minting and sending NFT

All the communication with the Cardano blockchain is being done by predefined transactions. Therefore, we simply configure the parameters of these transactions through our application. The following actions are being executed with Cardano type wallet. The life cycle of any transaction begins with building it. A building of a minting transaction and its parameters are shown in Listing 3. These parameters can be divided into 4 groups.

The first group defines what net and era we are using. Here, Cardano testnet with the newest era is used. Cardano is using the UTXO model [3], therefore, the transaction input and output have to be there.

This is defined in the second group. These parameters are TXIN (transaction input), TXOUT (transaction output), and change address. Thanks to the usage of the Alonzo era the minting system does not have to mint an NFT to minting wallet and afterward send it to a customer. It can directly mint an NFT to the customer's wallet. It saves time and transaction fees. UTXO hash and index of minting wallet are stored in TXIN. The address where an NFT will be sent, the minimum amount of ADA that has to be included in the transaction, token amount, policyID, and asset name are stored in TXOUT. The remaining assets in TXIN are sent to change address, i.e. the minting wallet address. The third group contains details about minting itself. Parameter mint indicates how many tokens with a particular policyID and asset name will be minted. The next parameter defines what policy script will be used. The last parameter of this group is the metadata JSON file. The fourth group contains information on how many private keys we need to sign the transaction. In the case of our NFT minting system, there have to be two private keys. The last parameter indicates where the built transaction file will be saved.

```
cmdTxRaw="cardano-cli transaction build"+\
    " --testnet-magic 1097911063"+\
    " --alonzo-era"+\
    " --tx-in "+xUTXOH+"#"+xUTXOI+\
    " --tx-out "+adress+"+"+str(txOutput)+'+"'+str(tokenAmount)+"
    "+policyID+"."+assetName+'"'+\
    " --change-address "+addressMint+\
    " --mint="+'"'+str(tokenAmount)+" "+policyID+"."+assetName+'"'+\
    " --minting-script-file "+policyScriptFile+\
    " --metadata-json-file "+metadataFile+\
    " --invalid-hereafter "+str(slotNumber)+\
    " --witness-override 2"+\
    " --out-file "+txMintRawFile
```

Listing 3: Minting build transaction.

The minting transaction needs to be multisigned with two separate private keys (i.e., two wallets). It is because we need permission for using ADA to pay transaction fees, permission for deploying the policy script, and being able to mint under the policyID. Finally, we submit the transaction to the blockchain. The actual mint of NFT happens when the block, where the submitted transaction was placed, gets created on the blockchain.

2.4. Payment listening

The most suitable tool for listening for incoming payments is Cardano Wallet. That is because this type of wallet focuses only on wallet functionality thus it provides a higher level of features. For example, this wallet saves easy to read history of all transactions of a particular wallet. For this reason, this wallet is used as a public receiving wallet for the customer to pay for goods. To retrieve the history of payments just a simple cardano-wallet GET request through the command line is needed. The NFT minting system repeatedly fetches a list of transactions and adds new incoming transactions to a database of payments. The system then reads every transaction one by one and determines if the amount of ADA received meets the price of NFT or not.

2.5. Sending ADA

ADA is being sent from Cardano Wallet. Here only one POST request command with three parameters is necessary: 1) sender address, 2) header parameter is always the same and defines the content type, and 3) data consist of a passphrase which acts as a security element, address of the receiver, and amount of ADA.

3. EXPERIMENTAL RESULTS

The NFT minting system was tested in an environment that simulates real-world usage. There are three possibilities of the outcome. If the amount that the customer sends meets the price of an NFT or is greater then: 1) the system mints, 2) sends NFT, and 3) sends back the change to the customer. Otherwise, the customer is refunded. All of them were tested with expected outcomes. For testing purposes, the customer used a third-party wallet called Nami wallet. In Figure 3, there is a screenshot of an easily readable minting transaction taken from cardanoscan.io, from Nami wallet, and the screenshot of minted NFT from Nami wallet. We can see that NFT parameters: 1) transaction ID, 2) date, 3) asset name, 4) policyID, and 5) asset fingerprint match.

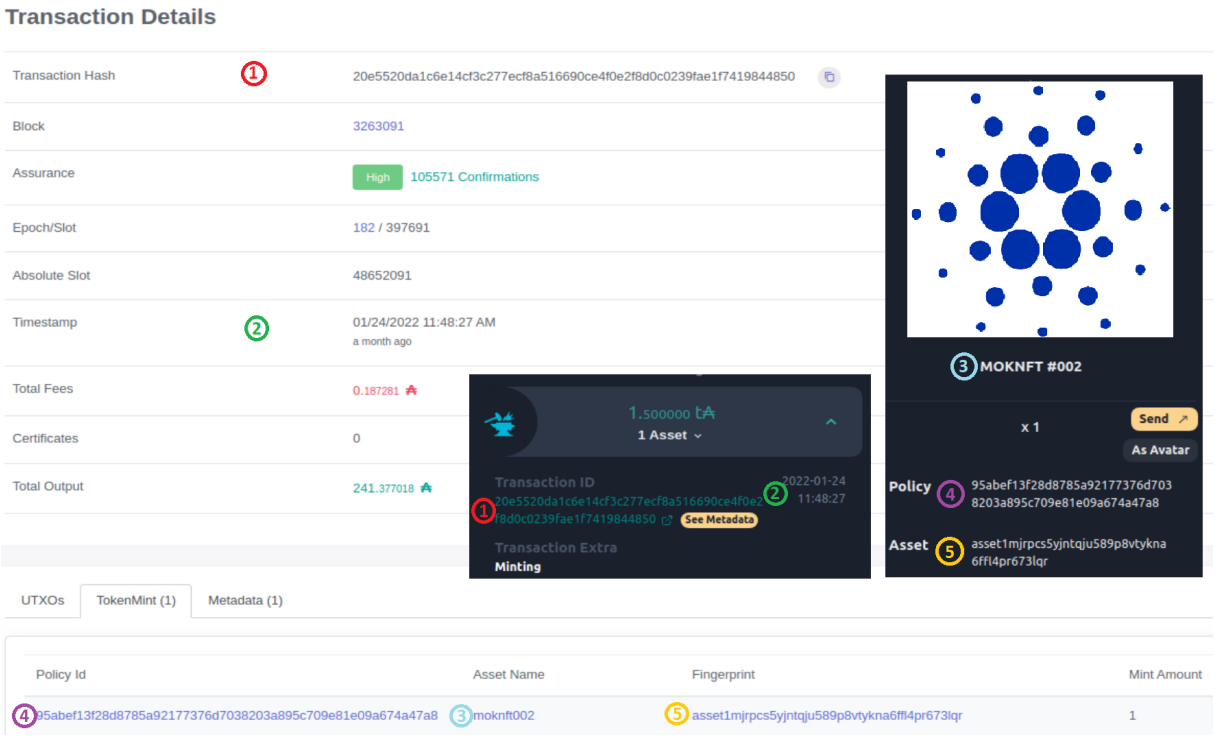

Figure 3: Minting transaction from cardanoscan.io and Nami wallet and minted NFT displayed in Nami.

There were minimum delays (ms) between incoming transactions from customers and minting transactions from the NFT minting system at the time of testing. However, the time between sending money and receiving NFT depends on the blockchain´s load. It could be milliseconds or even hours. Because of single thread usage, there were no collisions in databases and transactions.

4. CONCLUSION

In this paper, we present our NFT minting system. We demonstrate how the Cardano command-line interface in combination with Python language can be used for handling the minting process. The NFT minting system is fully compatible with the Cardano blockchain. It supports settings of policyID for NFT minting, setting of NFT metadata, minting, and sending NFT to the blockchain. Experimental results show that our minting system does work and automatically handles all the possible outcomes with minimum time delay.

REFERENCES

[1] PARHAM, Arsalan a Corinna BREITINGER. *Non-fungible Tokens: Promise or Peril?* [online]. 13 Feb 2022, 7 [cit. 2022-03-11]. Available from: doi: https://doi.org/10.48550/arXiv.2202.06354

[2] *Cardano Developer Portal* [online]. Zug, Switzerland: Cardano Foundation, 2022 [cit. 2022-03-11]. Available from: https://developers.cardano.org/

[3] CHAKRAVARTY, Manuel, James CHAPMAN, Kenneth MACKENZIE, Orestis MELKONIAN, Jann MÜLLER, Michael PEYTON JONES, Polina VINOGRADOVA a Philip WADLER. *Native Custom Tokens in the Extended UTXO Model* [online]. October 2020, 20 [cit. 2022-03-11]. Available from: https://iohk.io/en/research/library/papers/native-custom-tokens-in-the-extended-utxo-model/