*Applied Computer Science, vol. 17, no. 2, pp. 100–126*
*doi: 10.23743/acs-2021-16*

*Submitted: 2021-05-01 | Revised: 2021-06-17 | Accepted: 2021-06-23*

*Blockchain, Ethereum, Smart Contract,*
*ERC-20 Tokens, Loan Default*

*Saha RENO* [0000-0003-1897-9002][*],
*Sheikh Surfuddin Reza Ali CHOWDHURY* [0000-0001-6101-9977][**],
*Iqramuzzaman SADI* [0000-0003-1739-0896][***]

# MITIGATING LOAN ASSOCIATED FINANCIAL RISK USING BLOCKCHAIN BASED LENDING SYSTEM

**Abstract**

*Lending systems in real world are not much secure and reliable as the borrower and third parties involved in this aspect may create various deceitful situations. Blockchain is a secure system where the utilization of smart contract can avoid deceptive phenomena involved in lending but the decline in exchange rate of cryptocurrency can create the opportunity to pay back less than the borrowed amount in terms of fiat money. In this paper, a blockchain and smart contract-based lending framework is designed which requires the borrower to provide Ethereum Request for Comments (ERC)-20 standard tokens as collateral to mitigate the associated risks. The smart contract feature is utilized to automate the system without any third-party management. Besides, transaction stored in the blocks creates transparency among the users of the system. To tackle the aforementioned issues, ERC-20 token value is increased periodically and the instability of the exchange rate is surveilled by the system. By the end of this paper, some test cases and charts relevant to the data set are evaluated to assess the effectiveness of the system.*

## 1. INTRODUCTION

Blockchain refers to a chain of time-stamped and immutable blocks which are linked with each other using cryptography. Each block of the chain records Bitcoin or other cyrptocurrencies related transaction data (Swan, 2015). After a block gets filled with several transactional information, another block is created and gets linked with the previous one by a cryptographic hash (Narayanan, Bonneau, Felten, Miller & Goldfeder, 2016). To alter any transaction residing in a block, more than 50% of the hashing/computational power is required which is still a hypothetical thought known as 51% Attack (Yang, Chen & Chen, 2017). That is why, once a transaction gets stored in a block, it is almost impossible to delete or modify it. This phenomenon has made the blockchain as one of the most secured and reliable technologies ever existed. There are several crypto-currency based systems which use the concept of blockchain to provide security to its users. Ethereum is an open source, decentralized

[*] Bangladesh Army International University of Science and Technology, Dept. of CSE, Cumilla, Bangladesh, reno.saha39@gmail.com, me@sheikhsurfuddin.com, sadivictorian@gmail.com

blockchain based platform which has several benefits over other cryptocurrency based public ledgers. What makes this Ethereum so special is one of its unique features called Smart Contract (Buterin, 2014). Smart Contract is a piece of code which manages the overall functionality of a system and can be thought of as the back-end of any Ethereum based decentralized application (Mohanta, Panda & Jena, 2018).

Blockchain can be used to implement a digital collateral-based lending system which can remove all the shortcomings of the classical loan providing systems (Firdaus & Nugraha, 2019). Instead of any third-party managing the loan, a decentralized platform can be developed using Ethereum where smart contract is a computer code running on top of Ethereum network, which is intended to preserve all the rules and regulations of a platform. If an Ethereum based collateral loan system is developed, the smart contract feature will play the role of the manager for the supervision of the whole system and will make sure of the fact that both the borrower and lender utilize the system as per the rules (Okoye & Clark, 2018). As there will be no intervention of any human managing the system and only a piece of computer code will be given the responsibility, all the terms and conditions will be maintained precisely. Moreover, smart contracts involved in this system will ensure that the money borrowed from the lender must be repaid in time and if the debtor fails to repay the borrowed amount, smart contract will make lender the owner of the collateral. As a result, the collateral provided at the time of receiving the loans will permanently be seized by the lender. Each and every transaction regarding loans will be stored in the blocks and in no means these transactions can be altered or omitted. Moreover, if the value of collateral can be increased periodically by a certain amount, more people will be interested to join the system and lend money as it will economically benefit them. Keeping the financial benefit of the lenders in mind, all the calculations will be based on paper-money and not on the cryptocurrency used by the system. This is because, the exchange rate of the cryptocurrency fluctuates over time. If the exchange rate decreases in a certain time, then the lender will actually receive less amount of paper money at the time of repaying the loan. If all the accounting cost are calculated on the verge of fiat currency, then decrement in exchange rate will result in getting more cryptocurrency and the original amount can be retrieved. Reasonable mortgage cost, increment of collateral's pricing and generating loan interest will create keenness in people to join the system for buying tokens and providing loans as the system serves both as an income source and feasible lending system.

In this paper, we have proposed a smart contract based secure lending system by implementing the above-mentioned techniques. The system mitigates the financial risk by removing the deficiencies of the classical lending system of the real world and guarantees that the lenders is benefited economically each time he/she lends money to a borrower.

## 2. PRELIMINARIES

### 2.1. Blockchain

Blockchain refers to a chain of blocks which stores transactional information in such a way that the digital data inside the blocks are immutable (Narayanan, Bonneau, Felten, Miller & Goldfeder, 2016). It can be thought of as a public ledger which is distributed and decentralized. These blocks have their own unique hash which distinguishes them from each other (Yuan & Wang, 2016). Using a mechanism named 'Consensus', blockchain ensures that no fraud transactions can occur or the transactions inside these blocks cannot be omitted or changed (Singh & Singh, 2016; Heilman, Alshenibr, Baldimtsi, Scafuro & Goldberg, 2017). Proof of Work is one of the many Consensus mechanisms; in which a miner mines a block by making sure that no fraud transactions are being stored in the blocks or no intruder can modify the information residing in the blockchain (Yang, Chen & Chen, 2017). To temper a blockchain, 51% attack must occur, which refers to an attack made by a large number of miners or just by one single block validator taking over 51% of the system's computing power, which is still hypothetical and almost impossible (Cao, Husbands, Zhang, Binns, Kassam & Lan, 2018). This decentralized ledger is also destitute of double spending and thus making it a robust and trustworthy protocol (Watanabe, Fujimura, Nakadaira, Miyazaki, Akutsu & Kishigami, 2015; Christidis & Devetsikiotis, 2016).

### 2.2. Ethereum

Ethereum is an open source, decentralized blockchain based system which has a unique functionality named Smart Contract (Buterin, 2014). Developers can use this platform to build decentralized applications and what makes this system distinguishable from other blockchains is that it is programmable (Heilman, Alshenibr, Baldimtsi, Scafuro & Goldberg, 2017; Cao, Husbands, Zhang, Binns, Kassam & Lan, 2018; Ali, Nelson, Shea & Freedman, 2016). Ethereum supports many standard tokens which are not crypto-currencies but essential for Initial Coin Offerings (ICO) to raise fund from public for a future project or new cryptocurrencies to be developed by Li et al. (Li, Fung, Tang & Song, 2018). Moreover, there are various test networks for Ethereum besides the main network, all of which can be used to test a newly developed decentralized application whether it has any kinds of flaw and working according to the desired functionality (Li, Fung, Tang & Song, 2018).

### 2.3. Smart Contract

Smart Contract is a piece of code which controls the whole decentralized app built by a developer (Mohanta, Panda & Jena, 2018). It manages all the transactions being executed without third parties. Smart contract is described as an autonomous agent residing in blockchain by Luu et al. (Luu, Chu, Olickel, Saxena & Hobor, 2016). This contract includes the terms and conditions of the agreement between buyer and seller; necessary for a transaction to be executed, which are written into lines of code (Swan, 2015). Like each of the blocks in a blockchain, smart contract also has its own address so that the contract can easily be traced (Luu, Chu, Olickel, Saxena & Hobor, 2016). After writing the smart contract, it needs to be deployed to Ethereum's main network or test network (Buterin, 2014). Jeuls, Kosba and Shi stated that smart contract has several advantages over the traditional cryptocurrencies (Juels, Kosba & Shi, 2016).

## 2.4. Initial Coin Offering

Initial Coin Offering is a process to launch one's own cryptocurrency in the market by raising fund from the people who are interested in buying the newly arrived digital currency (Burns & Moro, 2018). To raise a fund, a person needs to create a document which essentially explains the system where this cryptocurrency can be used to pay for the desired services and also the benefits of using this new currency (Hrga, Benčić & Žarko, 2019). Anyone can set their own standards to create these tokens but ERC-20 Standard is by far the most popular and widely used standard in Ethereum Blockchain platform among others (Fenu, Marchesi, Marchesi & Tonelli, 2018). Though the creation and distribution of tokens are widely accessible to everyone, many legal considerations must be taken into account before holding an ICO (Zetzsche, Buckley, Arner & Föhr, 2017). Platforms which support the smart contract feature like Ethereum and Neo allow the developers to develop their own decentralized applications in which the smart contact facility governs the trading of ICO tokens (Rosic, 2017).

## 2.5. ERC-20 Standard

ERC-20 is a technical standard, widely utilized for specifying certain standards and rules for issuing tokens on Ethereum based systems. The figure 20 is used to assign this particular standard a unique identification number, which distinguishes this protocol from other variants of ERC standard tokens (Fröwis, Fuchs & Böhme, 2019). Prior to the arrival of ERC-20, if anyone wanted their token to be available on an exchange, they had to write custom code to allow others to trade and make the exchange possible (Somin, Gordon, Pentland, Shmueli & Altshuler, 2020). ERC-20 standard brings the uniformity among various Ethereum based tokens, reduces the complexity of understanding different types of token implementation and enhances the liquidity of the tokens (Victor & Lüders, 2019). Also, risk of breaking the token contract is minimized.

## 3. LITERATURE REVIEW

### 3.1. PTPTN Study Loan using Blockchain

National Higher Education Fund Corporation (PTPTN) is an organization managed by Malaysian Government which provides study loans for their citizen but is accused of poor management the filing system is not satisfactory. Moreover, 1.25 million students failed to payback their debts and thus people are taking this PTPTN loans as granted. Failure to repay the educational loans is drastically affecting the PTPTN as providing funds to the new students are becoming much more difficult. To eliminate these problems, a blockchain and smart contract based solution is proposed to allow the PTPTN and other agencies to monitor the current status of the borrowers. The loan adjustment among the system actors are maintained by three types of smart contracts: (i) Registrar Contract (RC) which assigns an Ethereum address or a public key to the individual taking loan to track his/her status, (ii) Customer-Provider Relationship (CPR) stores and manages all the records of education loan, (iii) Summary Contract (SC) is used to store borrower's transaction history in the system. Though the system is re-invented using blockchain, it is not fully decentralized as PTPTN and other government agencies are the centralized actors in charge of issuing the loans. Moreover, as this loan is for educational purpose and is refrained to the citizens and students of that country, only Malaysian students are able to be benefited from this system (Gazali, Hassan, Nor & Rahman, 2017).

### 3.2. Poverty Alleviation Loan Management using Smart Contract

To mollify the poverty in China, Poverty Alleviation Loan is introduced to the impoverished citizens of that country. As the current system is managed by a single service node, transparency and tractability are difficult to maintain. As a result, implementing this poverty loan management system using blockchain and smart contract is proposed. In this model, digital signature and oracle are used to ensure the data privacy. Transactions of this proposed system relies on: (i) Chaincode, which is a smart contract specifying all the terms and condition which must be met to execute certain operations and (ii) Unlocking Code, which is a piece of code used to satisfy the conditions mentioned in chaincode so that any specific transaction is allowed to be executed. Only needful citizens of China are given the permission to become a user of this system and are allowed to receive this poverty alleviation loan (Guo, Ma, Wang, Cheng & Wang, 2018).

### 3.3. Dharma

Founded by Nathan Hollander, Dharma is a popular platform which allows two individuals to get engaged in lending purpose. To mitigate the risk, a trusted third-party named Underwriter investigates on whether a borrower is faithful or not. This underwriter calculates the probability that a borrower will completely repay his loan by investigating the past transactions made by the borrower. But this mechanism surely does not lessen the risk to a great extent. Later, Dharma launched collateralized loans where the borrower must provide 1.5 times the money, he/she wants to borrow as collateral to ensure the payback of the borrowed amount along with the 8% interest (14% if the amount is in DAI) if the borrower fails to repay. This is certainly not feasible for the borrowers and as a result, people are being attracted to use the system as a lender and not as a borrower (Hollander, 2017).

### 3.4. Impact of BI Rate, Exchange Rate, Inflation, Third Party Fund (DFK) on Credit Distribution and Non-Performing Loan (NPL)

A quantitative research was conducted about the effect of BI Rate, Exchange Rate, Inflation, and Third-Party Fund (DFK) on Credit Distribution and Non-Performing Loan (NPL) where the researchers used the data of Bank in Indonesia. They have found that the BI Rate along with Exchange Rate and DFK of a loan is proportional with Credit Distribution. On the other hand, BI Rate has a negative impact on NPL resulting in lesser NPL cases. Also, Credit may create a negative and small impact on NPL which is also similar to research by Al-Wesabi (2020), which stated that NPL shows a positive and negligible impact on credit (Sinaga, Muda & Silalahi, 2020).

### 3.5. VaR Assessment and Selecting Portfolio for Western Balkan Countries

Llesh and Alban analyzed the stock quotations for five Western Balkan countries on a daily basis. Their analysis was based on capital distribution return and risk level was also considered. VaR and CVaR estimators were utilized for determining loss or profit intervals. Lagrange multipliers and interpolation were used for the methodological aspect. The evaluation showed that the Western Balkans Region's stock exchange indexes are not profitable. These indexes lack economic efficiency and simultaneous investing in the indexes of every exchange market turned out to be risky for investors. This research also found that the investing strategy in passive way is a better option for financial investment than the active strategy (Lleshaj & Korbi, 2020).

## 4. PROPOSED ARCHITECTURE OF LOAN DEFAULT PREVENTION PROTOCOL

Our proposed Ethereum based lending framework is defined using several procedures intended to issue, administrate, trade and fund cryptocurrency as loan; which is managed by the system's smart contracts, structured interfaces and ERC-20 tokens being utilized as collaterals. Our protocol is based on EVM based blockchain, but can be further extended to support permissioned and private blockchain like Hyperledger using chaincode functionality. Role of the participants and smart contracts, loan issuance and repayment process along with the system's terms and conditions are described in the following subsections:

### 4.1. Participants

The end-users of the system are defined as active participants of our framework -- clients willing to lend money to the debtor or, requesting loan from the creditor. Therefore, these participants fall under two categories:

1. **Borrower**

   The party who requests loan from the system and as requirement provides some tokens as collateral. They owe the lender some agreed upon amount. The debt order from a borrower must satisfy all terms and conditions of the proposed system. The credit score determined by risk assessment indicates a debtor's trustworthiness.

2. **Lender**

   The party which lends cryptocurrency as a loan asset to the borrower and takes tokens from them as collateral. Loan requests are passed to the lenders from the relayer's side. Lenders can accept or reject a debt request according to their preferences. Upon agreement, the requested amount is transferred from lender's wallet to the borrower's one. Lenders can also issue a loan as borrower and the same is applicable for the debtors.

   The interaction among the participant and system keepers (described in the succeeding subsection) is graphically represented using Fig. 1.

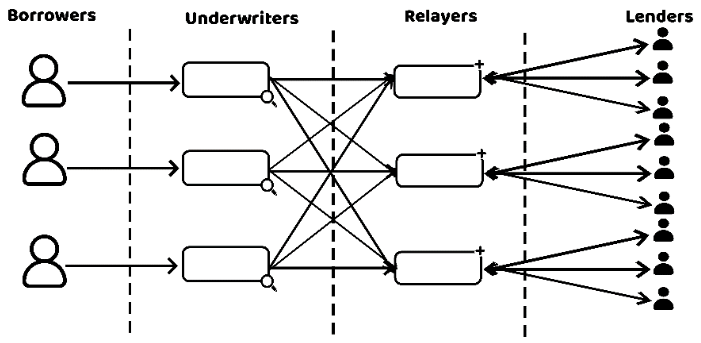


**Fig. 1. Interaction Among Different Participant Types**

## 4.2. System Keepers

We define system keepers as the managers of the overall architecture of our protocol. Keepers ensure the proper maintenance and accurate transitions of messages and information all over the system:

1. **Underwriter**

   Underwriter is one of the smart contracts of our system, certifying valid loan requests from the borrowers. It checks whether the loan request complies with the system's terms and conditions and forward the request to the relayer. The functions performed by the underwriter are as follows:

   a. Generating a loan request message to be passed from the borrower.
   b. Negotiating the debt to check the validity of the loan request.
   c. Ensuring in-time repayment in accordance with the terms agreed upon.
   d. Collecting collaterals from the borrowers and passing it to the relayer to inhibit loan defaulting.

2. **Relayer**

   Relayer receives valid loan requests from the underwriter and stores them in the memory of EVM named loan records. Relayer broadcasts these loan requests to the lenders and maps a lender to a specific loan request. Like underwriter, relayer is also a piece of smart contract where necessary instructions are provided for smooth maintenance of every loan case. Other duties of relayer includes:

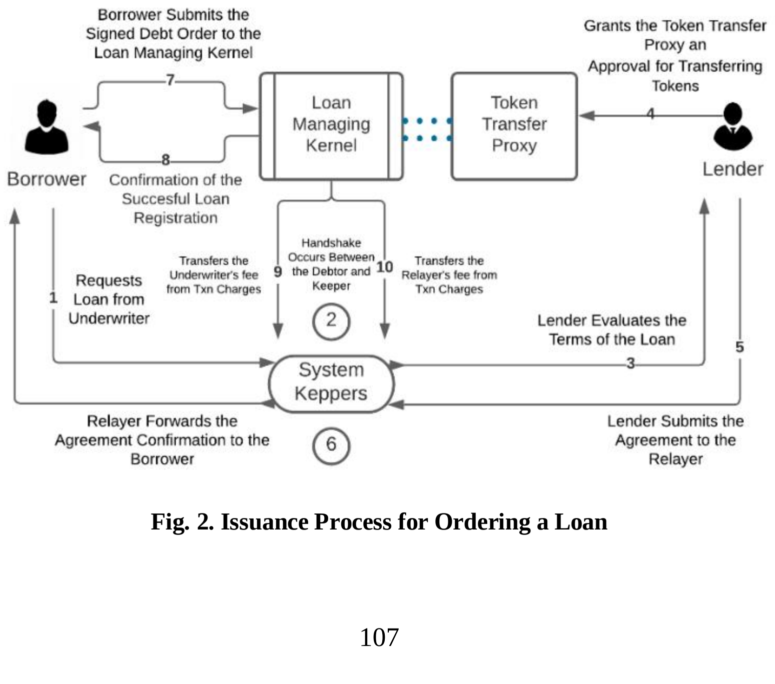
   a. Keeping track of the deadlines mentioned by the borrowers.
   b. Incrementing the price of the tokens provided as collateral and make it stop after the compensation amount is reached.
   c. If the borrower fails to repay the loan, relayer takes the collaterals from the underwriter, updates its value as per the calculations and transfers them to the lender's account.
   d. Compute the risk factor for a particular end-user of the system by evaluating the transaction records and loan histories.

## 4.3. Loan Issuance Process

The order in which a loan gets issued is described as follows:

1. Borrower requests a loan by filling up necessary and required information. This information include: loan amount, deadline to repay the loan, the agreement on terms and conditions of the protocol etc.
2. The request is passed to the underwriter. Underwriter sends the requestor's user ID to the relayer to fetch the credit score for checking the eligibility of taking loans. Also, the underwriter checks whether the borrower has enough ERC-20 tokens in their wallet. It also verifies if the requested amount surpasses the maximum limit a debtor can borrow money as per his/her credit score. If the deadline mentioned by the borrower does not comply with the system's stipulation, the underwriter declines the loan request. If the request follows all the system's rules and regulations, it is forwarded to the relayer.

3. Relayer broadcasts the loan request to a number of lenders. The lender willing to provide the solicited amount forwards the acceptance message to the relayer. Relayer then records all of the relevant information about that particular loan in EVM's memory. This relevant message includes Ether's exchange rate on loan receiving date.
4. Relayer also warns the creditor if the requestor is having a bad credit score. It provides suggestions about whether lending money to that particular borrower is considered risky or not.
5. Relayer informs the underwriter about the successful issuance of loan and requests it to fetch the required number of tokens from the borrower's account.
6. After successful retrieval of the required amount of collateral, relayer starts the countdown and also starts calculating the interest. The price of the collateral taken from the debtor gets incremented on a daily basis, until the loan is repaid or the threshold value is reached.
7. While the borrower is willing to repay the loan, the relayer checks the current exchange rate of Ether and calculates how much extra cryptocurrency the borrower needs to add to the repayment if the market price of Ether decreases.
8. If the borrower fails to repay the loan by the deadline, the relayer marks it as a failed repayment, updates the borrower's credit score and transfers the tokens to the lender's account after updating its price.
9. If the exchange rate of Ether increases, the debtors are not allowed to pay less cryptocurrency than the amount he/she borrowed. Instead, the full amount must be repaid.
10. On the other hand, if the debtor successfully repays the loan in time, the increments get cancelled and the tokens are returned back to the borrower's account.
11. The details of a specific loan issuance in EVM memory gets updated by relayer after the successful repayment/failure to payback. This information is required for the purpose of risk assessment.
12. All the transaction details of our proposed system are immutable in nature, meaning that they cannot be updated, altered or deleted by any means. Thus, transparency is maintained.



Borrower Submits the Signed Debt Order to the Loan Managing Kernel
7
Loan Managing Kernel
Token Transfer Proxy
Grants the Token Transfer Proxy an Approval for Transferring Tokens
4
Borrower
8
Confirmation of the Succesful Loan Registration
Lender
1
Requests Loan from Underwriter
Transfers the Underwriter's fee from Txn Charges
9
Handshake Occurs Between the Debtor and Keeper
10
Transfers the Relayer's fee from Txn Charges
2
Lender Evaluates the Terms of the Loan
5
3
System Keppers
Relayer Forwards the Agreement Confirmation to the Borrower
6
Lender Submits the Agreement to the Relayer

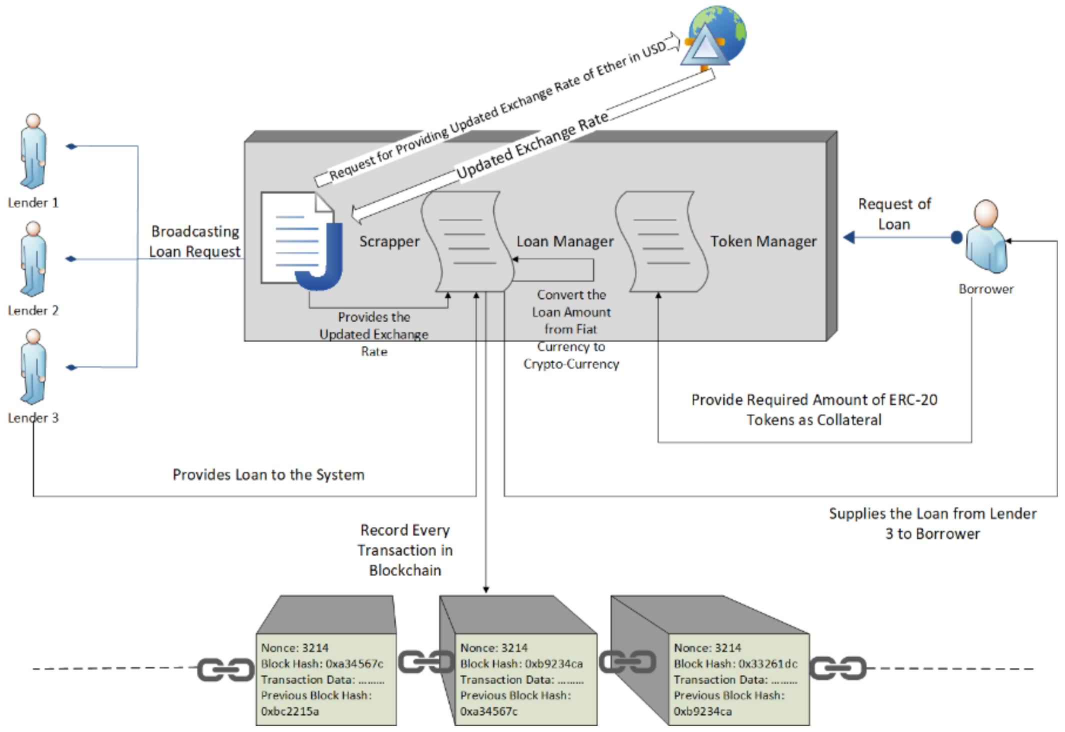
**Fig. 2. Issuance Process for Ordering a Loan**

The overall loan issuance is depicted using Fig. 2.

### 4.4. Terms and Conditions of the Protocol

To prevent loan defaulting and restrain lender's economical loss from debtor's failure to payback, the borrower must acknowledge and agree to the following terms and conditions of the proposed lending protocol:

1. Borrower must provide ERC-20 tokens worth 90% of the loan amount to request for loan issuance.
2. 2% daily interest is applicable on the lent amount. This rate can be changed and the periodical pattern can be switched from daily basis to monthly using the system's smart contract.
3. The escalation in collateral price also gets increased by 2% on a daily basis. Also, this rate and pattern of increment are adjustable.
4. The repayment deadline cannot exceed 90 days.



Request for Providing Updated Exchange Rate of Ether in USD
Updated Exchange Rate
Lender 1
Lender 2
Lender 3
Broadcasting Loan Request
Scrapper
Loan Manager
Token Manager
Request of Loan
Borrower
Provides the Updated Exchange Rate
Convert the Loan Amount from Fiat Currency to Crypto-Currency
Provide Required Amount of ERC-20 Tokens as Collateral
Provides Loan to the System
Supplies the Loan from Lender 3 to Borrower
Record Every Transaction in Blockchain
Nonce: 3214
Block Hash: 0xa34567c
Transaction Data: ........
Previous Block Hash: 0xbc2215a
Nonce: 3214
Block Hash: 0xb9234ca
Transaction Data: ........
Previous Block Hash: 0xa34567c
Nonce: 3214
Block Hash: 0x33261dc
Transaction Data: ........
Previous Block Hash: 0xb9234ca

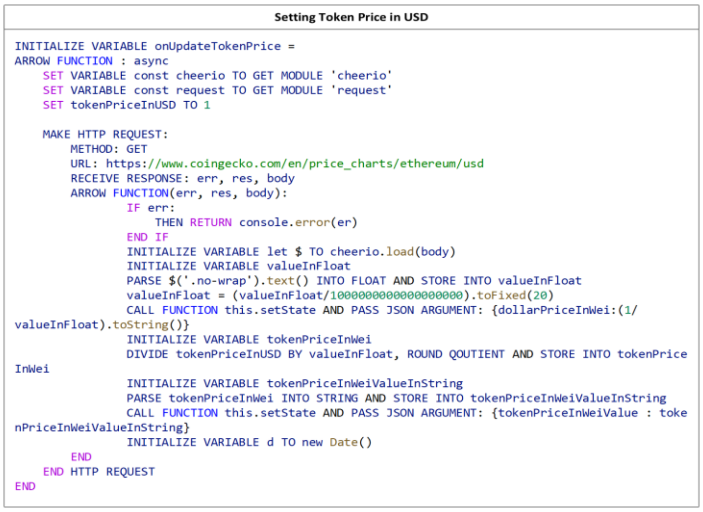
**Fig. 3. Overview of the Lending Management System**

## 5. METHODOLOGY

Although every financial transaction of the system utilizes cryptocurrency to be executed, all kinds of calculations are performed on the basis of fiat money to avoid the unwanted circumstances caused by the increase or decrease in the exchange rate of Ether. The system has its own modified ERC-20 standard tokens which are used as collateral for lending purpose. Users can buy these tokens from the system and can sell these either to the system or other users. Two smart contracts are in charge of managing the system. One of the smart contracts creates the tokens and governs the trading of these tokens among the users. The second one administrates the lending and borrowing aspects, along with generating interest, elevating the collateral price and liquidating the lender. Fig. 3 demonstrates the basic concepts of the proposed system and the working principle and basic operations of the system can be sectioned as follows:

### 5.1. Fixing the Price of Tokens in Fiat Currency

The price of these tokens is fixed in fiat currency and not in Ether, as fluctuations in exchange rate of cryptocurrencies create different pricing of these tokens at different times and decrement in Ether's exchange rate will bring financial loss to the holder of the tokens while selling them or using them to take loans. The system keeps checking the current exchange rate of Ether and updating the price of tokens continuously. For this purpose, two JavaScript modules 'cheerio' and 'request' are used to scrap necessary data from (CoinGecko, 2021). 'request' sends a request to (CoinGecko, 2021) for fetching the current exchange rate of Ether in USD. After the request is granted, 'cheerio' fetches the equivalent USD of 1 Ether from the aforementioned website. After fetching the equivalent USD of per Ether using 'cheerio', this value is stored inside the smart contract and then converted to Wei (1 Ether = $10^{18}$ Wei). CoinGecko does not directly provide how much Ether is achievable from 1 USD, so this value is inversed to get the amount of Ether receivable per dollar, which keeps getting updated due to the fluctuating price of cryptocurrencies. This amount is then multiplied by the price of per token (in USD), which gets us each token price in Wei.



**Setting Token Price in USD**

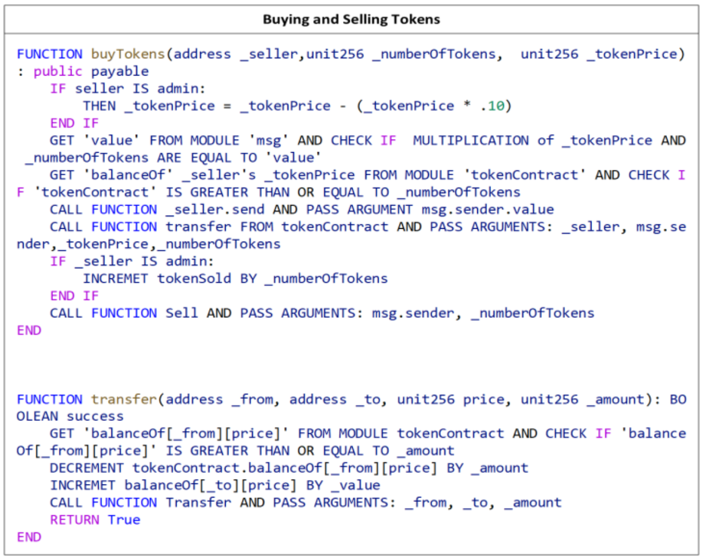
```
INITIALIZE VARIABLE onUpdateTokenPrice =
ARROW FUNCTION : async
    SET VARIABLE const cheerio TO GET MODULE 'cheerio'
    SET VARIABLE const request TO GET MODULE 'request'
    SET tokenPriceInUSD TO 1

    MAKE HTTP REQUEST:
        METHOD: GET
        URL: https://www.coingecko.com/en/price_charts/ethereum/usd
        RECEIVE RESPONSE: err, res, body
        ARROW FUNCTION(err, res, body):
                IF err:
                    THEN RETURN console.error(er)
                END IF
                INITIALIZE VARIABLE let $ TO cheerio.load(body)
                INITIALIZE VARIABLE valueInFloat
                PARSE $('.no-wrap').text() INTO FLOAT AND STORE INTO valueInFloat
                valueInFloat = (valueInFloat/1000000000000000000).toFixed(20)
                CALL FUNCTION this.setState AND PASS JSON ARGUMENT: {dollarPriceInWei:(1/
valueInFloat).toString()}
                INITIALIZE VARIABLE tokenPriceInWei
                DIVIDE tokenPriceInUSD BY valueInFloat, ROUND QOUTIENT AND STORE INTO tokenPrice
InWei
                INITIALIZE VARIABLE tokenPriceInWeiValueInString
                PARSE tokenPriceInWei INTO STRING AND STORE INTO tokenPriceInWeiValueInString
                CALL FUNCTION this.setState AND PASS JSON ARGUMENT: {tokenPriceInWeiValue : toke
nPriceInWeiValueInString}
                INITIALIZE VARIABLE d TO new Date()
        END
    END HTTP REQUEST
END
```

**Fig. 4. Setting Token Price in Fiat Money Using React.js**

As the equivalent Wei of per USD gets updated periodically inside the smart contract, the variable nature of exchange rate is thus tackled by requiring extra or less Ether from the client to buy a token from the system or its owner, ensuring that the price of the tokens does not deteriorate with the downfall of Ether's exchange rate. The code snippet to fix the price of ERC-20 tokens in fiat currency is demonstrated using Fig. 4.

## 5.2. Buying and Selling Tokens

Tokens can be purchased both from the system and other users. To buy tokens from the system, users simply need to mention the number of tokens they are willing to buy and provide the required amount of Ether to the system. Also, the users can sell their tokens to other interested users.



**Buying and Selling Tokens**

```
FUNCTION buyTokens(address _seller,unit256 _numberOfTokens,  unit256 _tokenPrice)
: public payable
    IF seller IS admin:
        THEN _tokenPrice = _tokenPrice - (_tokenPrice * .10)
    END IF
    GET 'value' FROM MODULE 'msg' AND CHECK IF  MULTIPLICATION of _tokenPrice AND
 _numberOfTokens ARE EQUAL TO 'value'
    GET 'balanceOf' _seller's _tokenPrice FROM MODULE 'tokenContract' AND CHECK I
F 'tokenContract' IS GREATER THAN OR EQUAL TO _numberOfTokens
    CALL FUNCTION _seller.send AND PASS ARGUMENT msg.sender.value
    CALL FUNCTION transfer FROM tokenContract AND PASS ARGUMENTS: _seller, msg.se
nder,_tokenPrice,_numberOfTokens
    IF _seller IS admin:
        INCREMET tokenSold BY _numberOfTokens
    END IF
    CALL FUNCTION Sell AND PASS ARGUMENTS: msg.sender, _numberOfTokens
END



FUNCTION transfer(address _from, address _to, unit256 price, unit256 _amount): BO
OLEAN success
    GET 'balanceOf[_from][price]' FROM MODULE tokenContract AND CHECK IF 'balance
Of[_from][price]' IS GREATER THAN OR EQUAL TO _amount
    DECREMENT tokenContract.balanceOf[_from][price] BY _amount
    INCREMET balanceOf[_to][price] BY _value
    CALL FUNCTION Transfer AND PASS ARGUMENTS: _from, _to, _amount
    RETURN True
END
```

**Fig. 5. Smart Contract Segment for Buying and Selling Tokens as Collateral**

To handle the trading of tokens, the smart contract is utilized to take the Ethereum wallet address of the buyer and transfer the tokens into that address. Two conditional statements ensure that (i) the buyer is providing enough Ether and (ii) the seller has enough Tokens in his/her wallet. Upon fulfilling the conditions, the Ether provided by the buyer are transferred to the seller's account. After that, the tokens are transferred from seller's wallet to the buyer's one.

As the price of some tokens which were seized as collateral during the period of a loan gets incremented, the system as well as token holders will carry tokens of different costs. As a result, there will exist both the tokens of regular price and the tokens of higher price than the original cost. To maintain tokens of different pricing (several price variations and the number of tokens under a specific price), mapping data structure from Solidity language is used, which takes the user/contract address and price variations as arguments. After providing an Ethereum address and a specific price variation, this mapping structure returns how much tokens the address currently holds for that specific price. It is up to the buyer which price-variant tokens he/she will buy from the user or from the smart contract itself (the system). System may or may not allow the smart contract to buy tokens from the users, depending on the availability of the fund raised by selling its tokens. To maintain the fund, system provides 10% less Ether than the actual price to its users in case of buying tokens from them. A conditional statement is evaluated to check whether the seller is the contract itself or not, and if the seller is the token selling contract, then the price of token is reduced to 10%. Smart contract pseudo code for buying and selling tokens is stated in Fig. 5 while Fig. 6 illuminates the basic token trading concept of the proposed lending system.

## 5.1. Lending Money to the Borrower

Before broadcasting a request of loan, with the help of the smart contact which manages the trading of tokens, the loan managing contract verifies whether the requester can provide required number of tokens as collateral. According to the current management, ERC-20 standard tokens of various price will exist in the system as the price of the tokens which are seized as collateral get increased by a certain percentage to mitigate the risk of lending money, which is described elaborately in the next subsection. As a result, tokens of different price variations can be noticed in the system. How many price-variant tokens a user owns in his wallet can be checked from the system. While requesting a loan or selling tokens to other users, a chart is shown to the users where the number of different priced tokens along with their rates can be examined. To borrow money from the creditor, the debtor must hold a certain number of tokens in his wallet, the price of which must be equal to 90% of the borrowed amount.

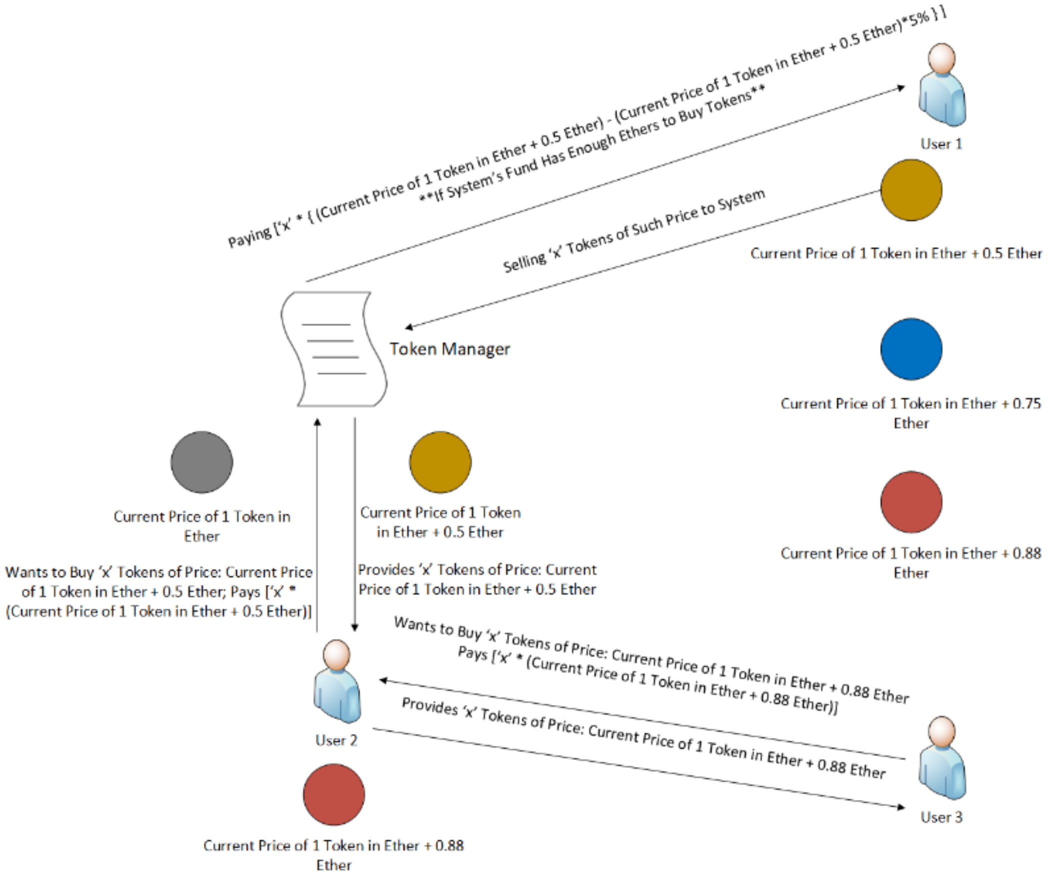

Paying ['x' * { (Current Price of 1 Token in Ether + 0.5 Ether) - (Current Price of 1 Token in Ether + 0.5 Ether)*5% } ]
**If System's Fund Has Enough Ethers to Buy Tokens**
User 1
Selling 'x' Tokens of Such Price to System
Current Price of 1 Token in Ether + 0.5 Ether
Token Manager
Current Price of 1 Token in Ether + 0.75 Ether
Current Price of 1 Token in Ether
Current Price of 1 Token in Ether + 0.5 Ether
Current Price of 1 Token in Ether + 0.88 Ether
Wants to Buy 'x' Tokens of Price: Current Price of 1 Token in Ether + 0.5 Ether; Pays ['x' * (Current Price of 1 Token in Ether + 0.5 Ether)]
Provides 'x' Tokens of Price: Current Price of 1 Token in Ether + 0.5 Ether
Wants to Buy 'x' Tokens of Price: Current Price of 1 Token in Ether + 0.88 Ether
Pays ['x' * (Current Price of 1 Token in Ether + 0.88 Ether)]
Provides 'x' Tokens of Price: Current Price of 1 Token in Ether + 0.88 Ether
User 2
User 3
Current Price of 1 Token in Ether + 0.88 Ether

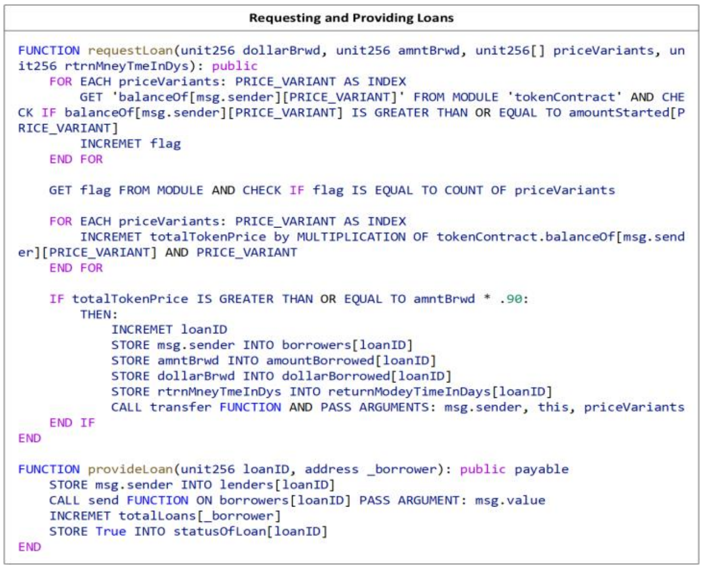
**Fig. 6. Trading of Different Price Variant Tokens Among Users and System**

Among tokens of various pricing, the borrower must select a proper combination of tokens which must satisfy the following condition:

$$\sum_{i=1}^{n}(\mathrm{TN_i} \times \mathrm{P_i}) \geq 0.9E \tag{1}$$

Where, TN refers to the number of tokens, P refers to the price of tokens and ***i*** denotes how many variants of token a user has and $E$ denotes to the amount of Ethers borrowed. In short, the borrower must provide a certain amount of collateral and the price of which must be greater than/equal to the 90% of the requested amount to be borrowed.

If the borrower is able to provide the required number of tokens as collateral, then the system marks the request as valid and broadcasts it to the lenders. After a lender accepts a request, the system collects the tokens and transfers these tokens to the smart contract serving as the back-end. The lender's money is then sent to the borrower's account. It is important to note that, to restrain the overflow of token variants caused by different pricing, the system may limit the users to hold a certain amount of price variant tokens. For example, a user cannot carry more than 10 different price variant tokens in his wallet. All of the aforementioned techniques managed by the loan managing contract is elaborately described below:



**Requesting and Providing Loans**

```
FUNCTION requestLoan(unit256 dollarBrwd, unit256 amntBrwd, unit256[] priceVariants, un
it256 rtrnMneyTmeInDys): public
    FOR EACH priceVariants: PRICE_VARIANT AS INDEX
        GET 'balanceOf[msg.sender][PRICE_VARIANT]' FROM MODULE 'tokenContract' AND CHE
CK IF balanceOf[msg.sender][PRICE_VARIANT] IS GREATER THAN OR EQUAL TO amountStarted[P
RICE_VARIANT]
        INCREMET flag
    END FOR

    GET flag FROM MODULE AND CHECK IF flag IS EQUAL TO COUNT OF priceVariants

    FOR EACH priceVariants: PRICE_VARIANT AS INDEX
        INCREMET totalTokenPrice by MULTIPLICATION OF tokenContract.balanceOf[msg.send
er][PRICE_VARIANT] AND PRICE_VARIANT
    END FOR

    IF totalTokenPrice IS GREATER THAN OR EQUAL TO amntBrwd * .90:
        THEN:
            INCREMET loanID
            STORE msg.sender INTO borrowers[loanID]
            STORE amntBrwd INTO amountBorrowed[loanID]
            STORE dollarBrwd INTO dollarBorrowed[loanID]
            STORE rtrnMneyTmeInDys INTO returnModeyTimeInDays[loanID]
            CALL transfer FUNCTION AND PASS ARGUMENTS: msg.sender, this, priceVariants
    END IF
END

FUNCTION provideLoan(unit256 loanID, address _borrower): public payable
    STORE msg.sender INTO lenders[loanID]
    CALL send FUNCTION ON borrowers[loanID] PASS ARGUMENT: msg.value
    INCREMET totalLoans[_borrower]
    STORE True INTO statusOfLoan[loanID]
END
```

**Fig. 7. Solidity Methods for Management of Requesting and Providing Loans**

The smart contract handles all the requests from the borrowers to check their eligibility of taking loans. To perform this operation, 4 parameters are required, among which the 'Amount of dollar borrowed' is required to record the requested amount in terms of USD, so that when the borrower is repaying the lender, the system can check how much extra Ether the borrower needs to pay if the exchange rate decreases. Another parameter contains the number of tokens the borrower is willing to provide against each price variants. At first, the contract determines whether the borrower's wallet actually contains the number of tokens he/she stated to be provided as collateral. A flag is used to check that the borrower's collateral combination is legit. If the combination of the total amount of different price variant tokens turns out to be valid, then the overall price of the borrower's collateral value is to be calculated as total token price. After calculation, the price of all the tokens is compared against (Borrowed Amount X 0.9). If the total price of these tokens is enough to grant the borrower the loan he/she requested, a loan ID will be generated to track all the loans managed by the system. The smart contract algorithm for providing loan is stated in Fig. 7.

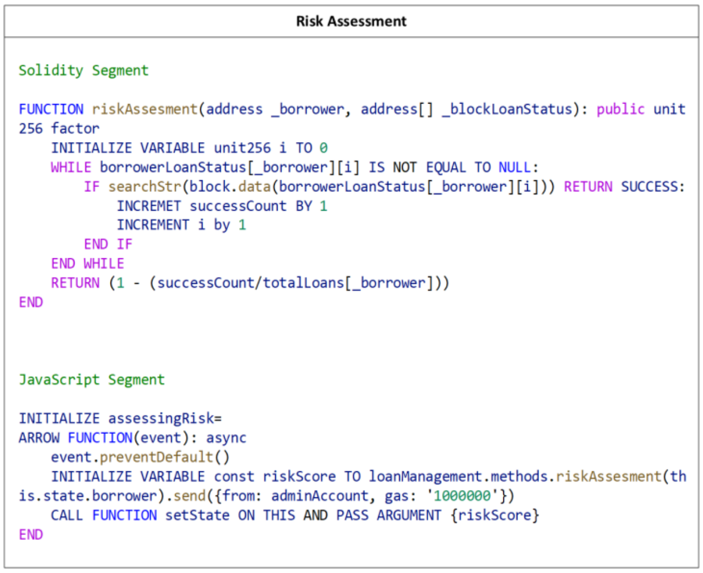
Borrower's address, the amount of loan in both Ether and USD and payback time are also recorded as the required loan information to be handled by the system later. Smart contract then locks all the collateral provided by the borrower in its own wallet. The lender then invokes the smart contract to provide the requested amount. The lender's address is recorded against the particular loan ID and smart contract transfer the loan to the borrower's wallet address. This contract constantly checks whether a loan is still to be repaid or not. Again, a mapping structure is used to track how many loans the borrower has taken so far, which is utilized for Risk Assessment purpose described in the next subsection.

## 5.2. Risk Assessment

All of the system's transactions are recorded in the immutable ledger of Ethereum network. Any transaction regarding successful or unsuccessful payback is stored in the blockchain and thus can be used for risk factor assessment. Each user will have their own risk factor that indicates how much risky it is to provide them a certain amount of loan. Risk factor of a user can be calculated using the following formula:

$$1 - \frac{Total\ Number\ of\ Succesful\ Paybacks}{Total\ Number\ of\ Loans\ Taken} \tag{2}$$

Fig. 8 contains the smart contract snippet to assess the Risk Factor. The assessment procedure of is described below:



Risk Assessment

```
Solidity Segment

FUNCTION riskAssesment(address _borrower, address[] _blockLoanStatus): public unit
256 factor
    INITIALIZE VARIABLE unit256 i TO 0
    WHILE borrowerLoanStatus[_borrower][i] IS NOT EQUAL TO NULL:
        IF searchStr(block.data(borrowerLoanStatus[_borrower][i])) RETURN SUCCESS:
            INCREMET successCount BY 1
            INCREMENT i by 1
        END IF
    END WHILE
    RETURN (1 - (successCount/totalLoans[_borrower]))
END



JavaScript Segment

INITIALIZE assessingRisk=
ARROW FUNCTION(event): async
    event.preventDefault()
    INITIALIZE VARIABLE const riskScore TO loanManagement.methods.riskAssesment(th
is.state.borrower).send({from: adminAccount, gas: '1000000'})
    CALL FUNCTION setState ON THIS AND PASS ARGUMENT {riskScore}
END
```

**Fig. 8. Risk Factor Calculation Using Smart Contract**

Loan managing contract stores the block address in its memory against each borrower of the system. Each time before broadcasting a borrower's loan request, this contract assesses the block data using the address and calculates the risk factor. The contract utilizes borrower's address and the addresses of the block containing the loan status against each borrower. The system then fetches the information from the specific blocks. This smart contract searches for the transactions containing the term 'success' from the fetched data and once it finds all the blocks containing desired transaction status, it will count the number of blocks. Finally, the risk factor against a particular borrower is calculated by ***1 – (Number of Success Counts / Total Number of Loans Taken to Date)***. The risk factor status of a borrower is displayed when the system broadcasts his/her loan request. For example, if a borrower takes 10 loans from different lenders and successfully paid 5 of them, then the risk factor 50% will be displayed beside his/her loan request. The schematic diagram for understanding the system's risk factor assessment procedure is illustrated by Fig. 9.

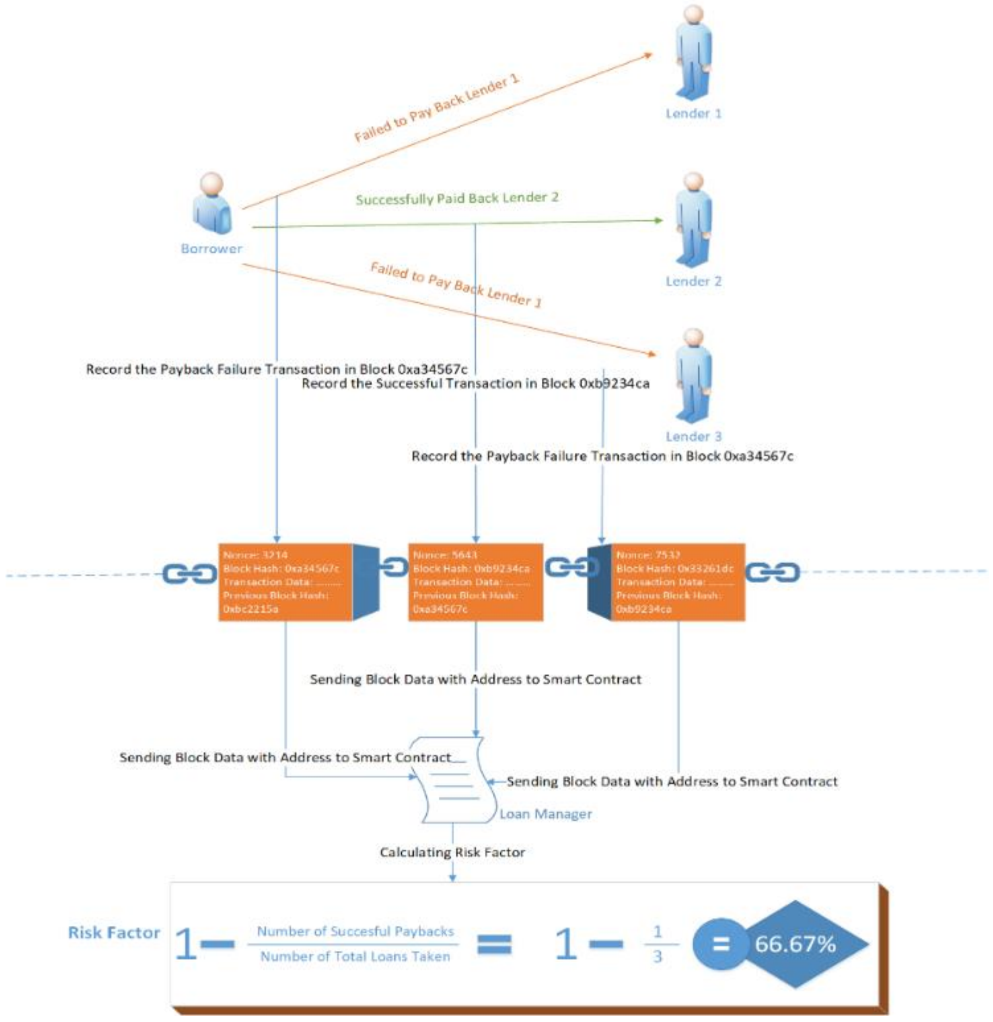

Lender 1
Failed to Pay Back Lender 1
Borrower
Successfully Paid Back Lender 2
Lender 2
Failed to Pay Back Lender 3
Lender 3
Record the Payback Failure Transaction in Block 0xa34567c
Record the Successful Transaction in Block 0xb9234ca
Record the Payback Failure Transaction in Block 0xa34567c
Nonce: 3214
Block Hash: 0xa34567c
Transaction Data: ........
Previous Block Hash: 0xbc2215a
Nonce: 5643
Block Hash: 0xb9234ca
Transaction Data: ........
Previous Block Hash: 0xa34567c
Nonce: 7532
Block Hash: 0x33261dc
Transaction Data: ........
Previous Block Hash: 0xb9234ca
Sending Block Data with Address to Smart Contract
Sending Block Data with Address to Smart Contract
Sending Block Data with Address to Smart Contract
Loan Manager
Calculating Risk Factor
Risk Factor
1 − Number of Succesful Paybacks / Number of Total Loans Taken = 1 − 1/3 = 66.67%

**Fig. 9. Accessing Block Data to Calculate Risk Assessment Factor**

## 5.3. Mitigating Financial Risk by Incrementing Token Price and Generating Interest

To mitigate the risk of financial loss of the lender and also to provide profit, the price of the tokens seized as collateral is increased periodically as follows: if the token price increases by the rate of 2% per day, the price of each token in Ether as per the exchange rate of loan issuance date is ***TP*** and the number of days between the date on which the loan was provided and the repayment date is ***R***, then the increment of per token price in each day will be:

$$\sum_{i=1}^{R} \mathrm{TP_i} \times 2\% \tag{3}$$

This increment is added to the actual cost of the token. The interest on the borrowed amount is be calculated as follows: If the interest is based on daily basis and the daily interest is 2%, the number of Ethers borrowed is ***E*** and ***M*** is the minimum interest to be paid (irrespective of the numbers of days passed since the loan issuance date), then the total amount of interest to be paid is:

$$\sum_{i=1}^{R} (E \times 2\%) \geq M \tag{4}$$

The steps mentioned above are implemented using Solidity and React.js by the following means: A Solidity mapping structure takes the loan ID and price of the tokens as its index values to keep track of different token pricing, as periodical price increment of specific token occurs continuously. Loan ID is used here to refer to the loans of which the collateral price is increased and pricing denotes all the pricing variants of the collateral provided by the borrower. From the front-end section, a Javascript segment invokes the smart contract to increment the token value once in a day, providing the loan ID and the chosen combination of different price variant tokens (which will be seized as collateral) as the arguments. Meanwhile, loan managing contracts also assess the expiration of deadline. If the loan is active and deadline is not crossed, then it increments each of price variants of the tokens by 2%, belonging to a specific loan ID. This increment continues until the case of the loan is closed or the deadline is expired. Upon the expiration of deadline, all the seized collateral whose price is incremented are then transferred from the contract's address to the lender's address. The contract also calculates the interest until the status of a loan ID is still active. Values of the incremented price variants under a particular user wallet are stored in the smart contract's memory, which will later be used to create any collateral combination for taking loans in the future. Javascript front-end segment again invokes the contract side, passing the status of loan, deadline and number of days elapsed as necessary arguments to calculate interest for a particular loan. If the loan is still active, deadline is not crossed and the number of passed days is 1, then system will calculate interest for 15 days (as per the rule of the system described below), the interest value will be stored against a particular loanID and the increment will stop for 15 days. If more than 15 days have passed, then again, the increment will be started on 2% daily basis. The formalization of the above techniques is provided in Fig. 10.

**Increment Token Price**

```
FUNCTION incrementTokenPrice(unit256 loanID, unit256[] pricing): public
    IF statusOfLoan[loanID] IS True AND deadline[loanID] IS False:
        THEN:
            WHILE prcing[i] IS NOT NULL:
                INCREMENT tokenPriceVariants[loanID][pricing[i]] BY tokenPriceVa
riants[loanID][pricing[i]] * 0.02
            END WHILE
    ELSE IF statusOfLoan[loanID] IS True AND deadline[loanID] IS True:
        THEN:
            WHILE pricing[i] IS NOT NULL:
                SET tokenContract.balanceOf[lender[loanID]][tokenPriceVariants[l
oanID][pricing][i]] TO tokenContract.balanceOf[this][tokenPriceVariants[loanID][
pricing[i]]]
                SET tokenContract.balanceOf[this][tokenPriceVariants[loanID][pri
cing[i]]] TO NULL
                CALL FUNCTION push ON userTokenPriceVariants[lender[loanID]] and
 PASS ARGUMENT: tokenPriceVariants[loanID][pricing[i]]
            END WHILE
    END IF
END
```

**Calculating Interest**

```
FUNCTION generateInterest(unit256 baseAmount, unit256 loanID): public
    IF statusOfLoan[loanID] IS True AND deadline[loanID] IS FALSE AND firstDay
 EQUAL TO daysPassed EQUAL TO 1:
        THEN:
            SET interest[loanID] TO amountBorrowed[loanID] * 0.02 * 15
            IF statusOfLoan[loanID] IS True AND deadline[loanID] IS False AND
daysPassed IS LESS THAN 15:
                THEN:
                    CONTINUE
            ELSE:
                INCREMENT interest[loanID] BY interest[loanID] * 0.02
            END IF
    END IF
END
```

**Fig. 10. Smart Contract Management for Mitigating Loan Risk**

Javascript front-end segment again invokes the contract side, passing the status of loan, deadline and number of days elapsed as necessary arguments to calculate interest for a particular loan. If the loan is still active, deadline is not crossed and the number of passed days is 1, then system will calculate interest for 15 days (as per the rule of the system described below), the interest value will be stored against a particular loanID and the increment will stop for 15 days. If more than 15 days have passed, then again, the increment will be started on 2% daily basis. The formalization of the above techniques is provided in Fig. 10.

## 5.4. Repaying the Borrowed Amount

The loan managing contract is to be triggered by the borrower to repay the loan. The increment of both the collateral price and interest continues until the payback date. While the borrower is willing to repay the money, the system prepares to check how much the exchange rate of Ether on the repayment date varies from the exchange rate on loan issuance date. If the exchange rate of Ether decreases, the system then calculates the extra amount Ether which the borrower needs to add to the original payback amount. On the other hand, if the exchange rate increases, then the borrower needs to pay the original amount. System's payback and liquidation management are graphically represented using Fig. 11.

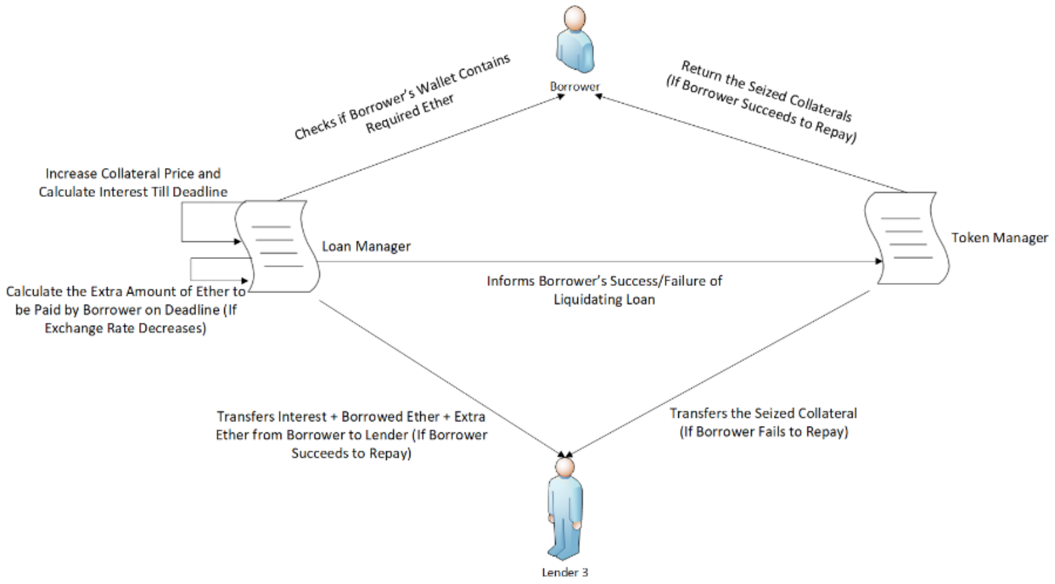

Borrower
Checks if Borrower's Wallet Contains Required Ether
Return the Seized Collaterals (If Borrower Succeeds to Repay)
Increase Collateral Price and Calculate Interest Till Deadline
Loan Manager
Token Manager
Calculate the Extra Amount of Ether to be Paid by Borrower on Deadline (If Exchange Rate Decreases)
Informs Borrower's Success/Failure of Liquidating Loan
Transfers Interest + Borrowed Ether + Extra Ether from Borrower to Lender (If Borrower Succeeds to Repay)
Transfers the Seized Collateral (If Borrower Fails to Repay)
Lender 3

**Fig. 11. Payback and Liquidation Management Using Smart Contracts**

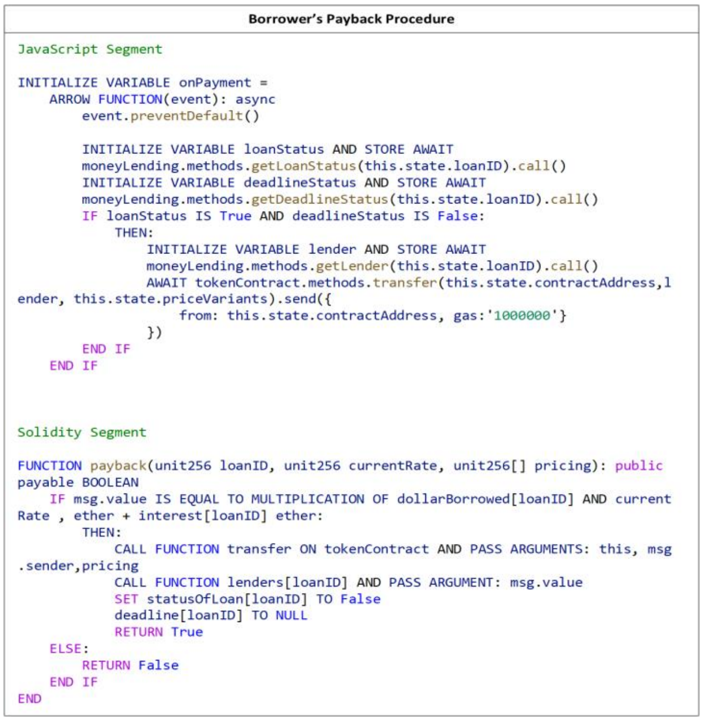
Suppose, at the time of borrowing, the exchange rate of 1 Ether was equal to ***D*** dollars and the borrower wanted ***B*** dollars from the lender. Then, the borrower will get ***(B/D)*** Ether. When repaying the amount, the exchange rate of 1 Ether decreases to $\boldsymbol{D'}$ dollars. Then the extra amount of Ether the borrower must provide to the lender will be according to the following equation:

$$\left({}^{D}/_{D'} - {}^{B}/_{D}\right) Ether \tag{5}$$

This is due to the fact that, the system refrains the borrower's opportunity to provide less Ether if exchange rate decreases as this phenomenon reduces the equivalent Ether of a certain amount of fiat money. According to the above discussion, the total payback amount a borrower must return by the deadline will follow the equation stated below:

$$E + \left({}^{D}/_{D'} - {}^{B}/_{D}\right) + (E \times 2\% \times N)\ Ether \tag{6}$$

where ***E***, ***[(D/D') - (B/D)]*** and ***(E × 2% × N)*** denote to the original borrowed amount, the extra Ether to be paid if exchange rate decreases and the amount of interest respectively (***N*** refers to the number of days between the loan issuance date and the payback date). If the borrower is able to provide the amount calculated using the above formula, only then the request to repay the borrowed amount is accepted and all the tokens are returned to his/her wallet. If the debtor does not repay the borrowed amount, all tokens whose prices were being increased on a regular basis to compensate for the unpaid amount are transferred from the smart contract's account to the lender's wallet. It is to be noted that, upon successful payback, the increment of the collateral price is reverted before being returned to the borrower's account to avoid the overflow of variances of token prices. The aforementioned techniques to mitigate is implemented in the loan managing smart contract by the following means: From the Javascript end, the system first fetches the loan status and deadline status of a particular loan ID to check the list of unpaid amounts. If the loan status is active but the deadline is expired, then system will directly trigger the transfer to send all the seized collateral to the lender's account. In case of payment imitative taken by the borrower, the payback functionality of smart contact is triggered by the borrower to repay the money. This payback requires the current exchange rate to calculate the updated price of payback amount and also the loan ID. These parameters are passed by the React.js front-end code snippet.



**Borrower's Payback Procedure**

```
JavaScript Segment

INITIALIZE VARIABLE onPayment =
    ARROW FUNCTION(event): async
        event.preventDefault()

        INITIALIZE VARIABLE loanStatus AND STORE AWAIT
        moneyLending.methods.getLoanStatus(this.state.loanID).call()
        INITIALIZE VARIABLE deadlineStatus AND STORE AWAIT
        moneyLending.methods.getDeadlineStatus(this.state.loanID).call()
        IF loanStatus IS True AND deadlineStatus IS False:
            THEN:
                INITIALIZE VARIABLE lender AND STORE AWAIT
                moneyLending.methods.getLender(this.state.loanID).call()
                AWAIT tokenContract.methods.transfer(this.state.contractAddress,l
ender, this.state.priceVariants).send({
                    from: this.state.contractAddress, gas:'1000000'}
                })
        END IF
    END IF



Solidity Segment

FUNCTION payback(unit256 loanID, unit256 currentRate, unit256[] pricing): public
payable BOOLEAN
    IF msg.value IS EQUAL TO MULTIPLICATION OF dollarBorrowed[loanID] AND current
Rate , ether + interest[loanID] ether:
        THEN:
            CALL FUNCTION transfer ON tokenContract AND PASS ARGUMENTS: this, msg
.sender,pricing
            CALL FUNCTION lenders[loanID] AND PASS ARGUMENT: msg.value
            SET statusOfLoan[loanID] TO False
            deadline[loanID] TO NULL
            RETURN True
    ELSE:
        RETURN False
    END IF
END
```

**Fig. 12. Reimbursement Technique to Payback Lender**

To evaluate the current exchange rate, the code segment to fetch the required information using 'cheerio' and 'request' modules are described in Fig. 4. Also, it needs the amount of generated interest under a specific loan ID, which can be found in the contract memory. After calculating the returnable amount, smart contract checks if the payback amount equals the calculated amount plus the amount of interest generated. If it matches, the contract will execute the transfer to liquidate lender, returns all the seized collateral from the contract address to the borrower's one, and mark the status of loan as closed. If the borrower's wallet does not have the required amount of money, he/she will be warned about this issue by the front-end section. The overall lending management is depicted using the flowchart in Fig. 13.

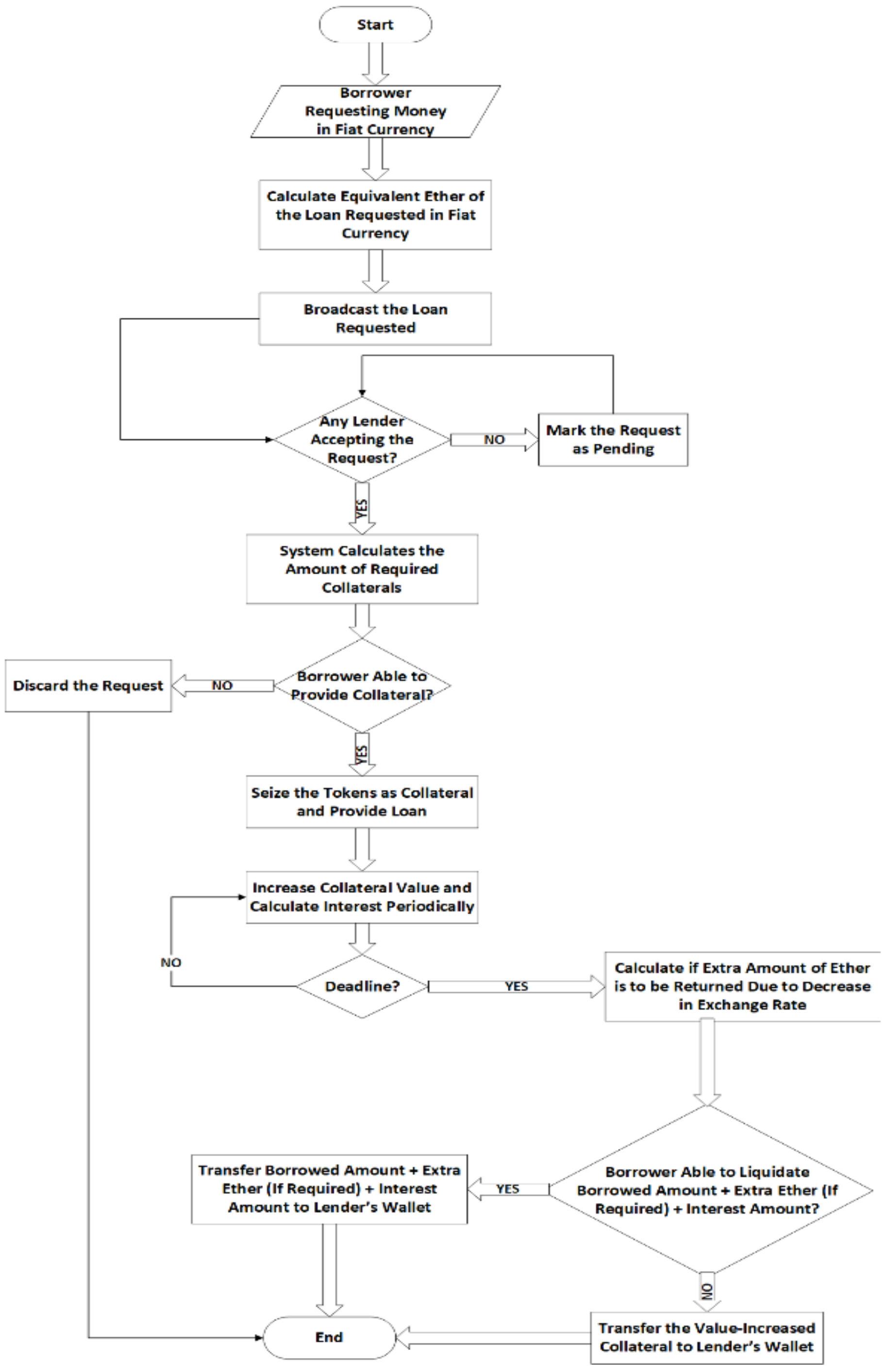

Start
Borrower Requesting Money in Fiat Currency
Calculate Equivalent Ether of the Loan Requested in Fiat Currency
Broadcast the Loan Requested
Any Lender Accepting the Request?
NO
Mark the Request as Pending
YES
System Calculates the Amount of Required Collaterals
Borrower Able to Provide Collateral?
NO
Discard the Request
YES
Seize the Tokens as Collateral and Provide Loan
Increase Collateral Value and Calculate Interest Periodically
NO
Deadline?
YES
Calculate if Extra Amount of Ether is to be Returned Due to Decrease in Exchange Rate
Borrower Able to Liquidate Borrowed Amount + Extra Ether (If Required) + Interest Amount?
YES
Transfer Borrowed Amount + Extra Ether (If Required) + Interest Amount to Lender's Wallet
NO
Transfer the Value-Increased Collateral to Lender's Wallet
End

**Fig. 13. Flowchart of the Overall Lending Management System**

## 6. RESULT ANALYSIS

React, a JavaScript library is used to build the user interface for the proposed architecture by which the borrowers and lenders will interact with each other and the system. Infura is a back-end infrastructure where all the smart contracts are uploaded to connect this decentralized lending system app to the Ethereum network. The front-end communicates with these smart contracts via web3 and Metamask. Fig. 14 depicts the aforementioned decentralized lending system structure. To assess the effectiveness of our system, some test cases are evaluated using the system which can be found in the Table 1.

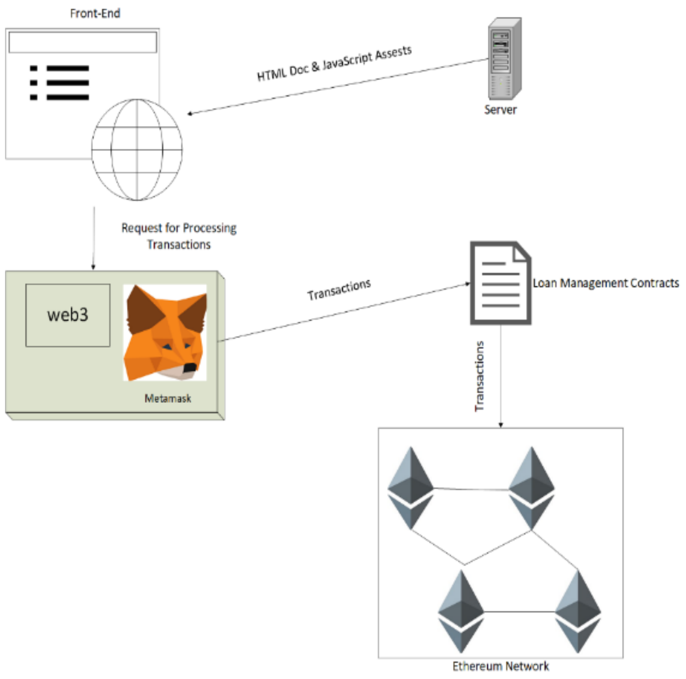

Front-End
HTML Doc & JavaScript Assests
Server
Request for Processing Transactions
web3
Metamask
Transactions
Loan Management Contracts
Transactions
Ethereum Network

**Fig. 14. Architecture of the Decentralized Lending System**

From the instances of Table 1, it is noticeable that in case of Test 1, 2, 3, 4, 5 and 8; the exchange rates of Ether were decreased on the loan repayment date compared to loan issuance date. If the borrower would return the same base amount which the lender lent on the loan issuance date, lender would actually receive less fiat currency because of the downfall of the exchange rate. This can cause financial loss to the lender, in terms of fiat money. For example, in the first case, lender provided 500 USD to the borrower which was equivalent to 1.865 Ether according to the exchange rate of 17th Feb, 2020. After 39 days, when the borrower returned the money on 26th March, 2020, the exchange rate fell from 268.03 USD to 139.00 USD and according to the updated exchange rate, 1.865 Ether would actually be equivalent to 259.235 USD. So, if the borrower returned exactly 1.865 Ether, the lender would receive 240.765 USD less, but according to our system, the payback must be according to the fiat currency the borrower had taken (500 USD) and the exchange rate of the reimbursement date. For this reason, the borrower must provide 3.595 Ether which is equal to 500 USD on the loan repayment date.

For this reason, the borrower must provide 3.595 Ether which is equal to 500 USD on the loan repayment date. In cases of 6, 7 and 9; the exchange rate increases and it seems that the borrower may provide less Ether to the lenders, but to make the system profitable for its users, the protocol requires the borrower to pay the same base amount of Ether he/she borrowed.

**Tab. 1. Test Case Evaluation Using the Proposed System: Incrementing Token Price to Mitigate Risk (Source: www.coingecko.com, Author Generated Test Cases)**

| Test Case No. | Loan Issuance Date | Loan Repayment Date | Amount Borrowed (USD/Ether) | Exchange Rate of Ether on Loan Issuance Date | Exchange Rate of Ether on Loan Repayment Date | Amount to be Repaid (In Ether) | Extra Amount of Ether Paid |
|---|---|---|---|---|---|---|---|
| 01 | 17 February, 2020 | 26 March, 2020 | $500/1.865 ETH | 268.03 USD | 139.00 USD (-) | 3.595 ETH | + 1.73 ETH |
| 02 | 18 February, 2020 | 11 March, 2020 | $300/1.064 ETH | 281.95 USD | 194.22 USD (-) | 1.545 ETH | + 0.481 ETH |
| 03 | 23 February, 2020 | 18 March, 2020 | $350/1.274 ETH | 274.63 USD | 117.70 USD (-) | 2.974 ETH | + 1.699 ETH |
| 04 | 2 May, 2020 | 11 May, 2020 | $400/1.871 ETH | 213.82 USD | 184.85 USD (-) | 2.164 ETH | + 0.293 ETH |
| 05 | 10 March, 2020 | 16 March, 2020 | $700/3.487 ETH | 200.75 USD | 110.99 USD (-) | 6.307 ETH | + 2.82 ETH |
| 06 | 23 May, 2020 | 31 May, 2020 | $820/3.965 ETH | 206.80 USD | 231.54 USD (+) | 3.542 ETH | N/A |
| 07 | 13 May, 2020 | 30 May, 2020 | $960/4.799 ETH | 200.01 USD | 243.28 USD (+) | 3.946 ETH | N/A |
| 08 | 14 Feb, 2020 | 12 March, 2020 | $1100/3.870 ETH | 284.23 USD | 110.60 USD (-) | 9.945 ETH | + 6.075 ETH |
| 09 | 15 April, 2020 | 29 April, 2020 | $990/6.461 ETH | 153.22 USD | 217.32 USD (+) | 4.555 ETH | N/A |

For example, in the last case, the exchange rate decreased on 29th April and according to that borrower should have paid 4.555 Ether which is still equivalent to 900 USD. But, the protocol of our system forced the borrower to pay 6.461 Ether and, in that case, the lender received profit of extra 414 USD. The following chart compares our protocol with existing Blockchain based lending protocols (which require only the base borrowed amount to be repaid, besides the interest) corresponding to the extra payment made for the compensation of the loss that could occur due to the downfall of exchange rate. Here, from the chart represented below by Fig. 15, it is clearly visible that our system (orange colored line) forces the borrower to mitigate the financial loss of lender by providing extra Ether in 6 out of 9 cases as the exchange rate descended on the repayment date. Also, in case of the rest 3 cases (where exchange rate increased), the system performs the same as compared to other Blockchain based lending Protocol (e.g., Dharma). Table 2 demonstrates the history of the collateral seized at the time of taking loans in all the 9 test cases. It also illustrates how the periodic increment of collateral price can compensate the loss which could happen due to the borrower's failure to payback.

To understand how the increment of collateral price mitigates the financial risk of lender, we explain the scenario of the first test case in which the borrower took 500 USD as loan on 17 February, 2020; which is equivalent to 1.865 Ether according to the exchange rate of that loan issuance date. The borrower chose a suitable combination of tokens as collateral which were seized temporarily by the system. Borrower provided 100 tokens of price 0.00373 Ether, 75 tokens of price 0.00799 Ether and 45 tokens of 0.01632 Ether, totaling 1.70260 Ether, which is about 90% of the borrowed amount. Until the repayment date arrived, the price of these tokens increased daily by 2%.

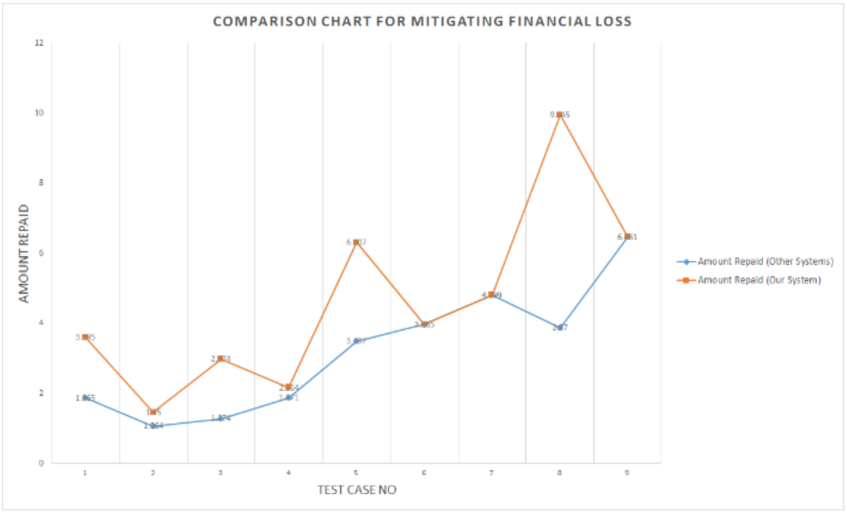


**Fig. 15. System's Efficiency to Tackle Low Exchange Rate Compared to the Basic Protocols**

Assuming the loan repayment date as deadline day, if the borrower failed to repay the loan, then all the seized tokens equivalent to 3.03055 Ether would be transferred to the lender's account, which is greater than the base amount 1.865 Ether. The column chart in Fig. 16 depicts how the periodic increment of the token price compensates the base amount upon failure of payback. From this chart, it is clearly reflected that in all of the 9 test cases, if the borrower failed to payback the borrowed amount, the lender would not face any loss as the total updated price of the seized collateral price is greater than the base amount.

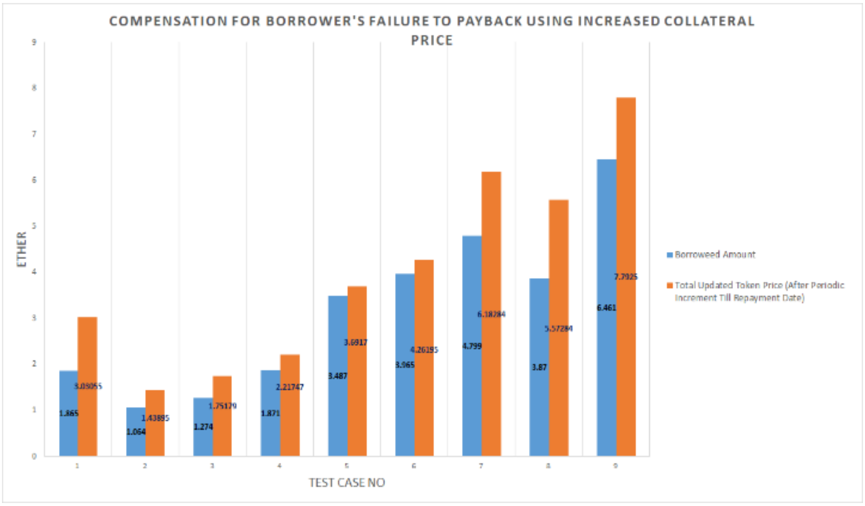


**Fig. 16. Visual Demonstration of Compensating Borrower's Payback-Failure with Increasing Collateral Price**

Finally, the summarization of the cases tested using the lending system is represented by Table 3 (includes the amount of interest and total repayable balance).

**Tab. 2. Test Case Evaluation Using the Proposed System: Incrementing Token Price to Mitigate Risk (Source: www.coingecko.com, Author Generated Test Cases)**

| Test Case No. | Loan Issuance Date | Loan Repayment Date | Amount Borrowed (USD/ Ether) | Tokens Combination with Price (In Ether) | Token Price Incremented (Incremented Periodically Till Repayment Date) | Total Price of Seized Collateral |
|---|---|---|---|---|---|---|
| 01 | 17 February, 2020 | 26 March, 2020 | $500/1.865 ETH | (i) 0.00373 ETH: 100 Tokens (ii) 0.00799 ETH: 75 Tokens (iii) 0.01623 ETH: 45 Tokens | (i) +0.00291 ETH: 100 Tokens (ii) +0.00623 ETH: 75 Tokens (iii) +0.01266 ETH: 45 Tokens | (i) 0.66400 ETH + (ii) 1.06650 ETH + (iii) 1.30005 ETH = 3.03055 ETH |
| 02 | 18 February, 2020 | 11 March, 2020 | $300/1.064 ETH | (i) 0.00354 ETH: 60 Tokens (ii) 0.00417 ETH: 45 Tokens (iii) 0.01953 ETH: 30 Tokens | (i) +0.00163 ETH: 60 Tokens (ii) +0.00192 ETH: 45 Tokens (iii) +0.00896 ETH: 30 Tokens | (i) 0.31020 ETH + (ii) 0.27405 ETH + (iii) 0.85470 ETH = 1.43895 ETH |
| 03 | 23 February, 2020 | 18 March, 2020 | $350/1.274 ETH | (i) 0.00364 ETH: 70 Tokens (ii) 0.00458 ETH: 35 Tokens (iii) 0.0511 ETH: 15 Tokens | (i) +0.00182 ETH: 70 Tokens (ii) +0.00229 ETH: 35 Tokens (iii) +0.02555 ETH: 15 Tokens | (i) 0.38220 ETH + (ii) 0.21984 ETH + (iii) 1.14975 ETH = 1.75179 ETH |
| 04 | 2 May, 2020 | 11 May, 2020 | $400/1.871 ETH | (i) 0.00468 ETH: 80 Tokens (ii) 0.00679 ETH: 45 Tokens (iii) 0.06487 ETH: 18 Tokens | (i) +0.00094 ETH: 80 Tokens (ii) +0.00136 ETH: 45 Tokens (iii) +0.01297 ETH: 18 Tokens | (i) 0.44960 ETH + (ii) 0.36675 ETH + (iii) 1.40112 ETH = 2.21747 ETH |
| 05 | 10 March, 2020 | 16 March, 2020 | $700/3.487 ETH | (i) 0.00498 ETH: 70 Tokens (ii) 0.00725 ETH: 50 Tokens (iii) 0.08423 ETH: 30 Tokens | (i) +0.00070 ETH: 70 Tokens (ii) +0.00102 ETH: 50 Tokens (iii) +0.01179 ETH: 30 Tokens | (i) 0.39760 ETH + (ii) 0.41350 ETH + (iii) 2.88060 ETH = 3.69170 ETH |
| 06 | 23 May, 2020 | 31 May, 2020 | $820/3.965 ETH | (i) 0.004835 ETH: 90 Tokens (ii) 0.04236 ETH: 75 Tokens | (i) +0.00087 ETH: 90 Tokens (ii) +0.00762 ETH: 75 Tokens | (i) 0.51345 ETH + (ii) 3.74850 ETH = 4.26195 ETH |
| 07 | 13 May, 2020 | 30 May, 2020 | $960/4.799 ETH | (i) 0.004999 ETH: 60 Tokens (ii) 0.09875 ETH: 43 Tokens | (i) +0.00180 ETH: 60 Tokens (ii) +0.03555 ETH: 43 Tokens | (i) 0.40794 ETH + (ii) 5.77490 ETH = 6.18284 ETH |
| 08 | 14 Feb, 2020 | 12 March, 2020 | $1100/ 3.870ETH | (i) 0.003518 ETH: 55 Tokens (ii) 0.08456 ETH: 40 Tokens | (i) +0.00197 ETH: 55 Tokens (ii) +0.04735 ETH: 40 Tokens | (i) 0.30184 ETH + (ii) 5.27640 ETH = 5.57284 ETH |
| 09 | 15 April, 2020 | 29 April, 2020 | $990/6.461 ETH | (i) 0.00652 ETH: 75 Tokens (ii) 0.08423 ETH: 65 Tokens | (i) +0.00196 ETH: 75 Tokens (ii) +0.02527 ETH: 65 Tokens | (i) 0.63600 ETH + (ii) 7.15650 ETH = 7.79250 ETH |

**Tab. 3. Test Case Evaluation Using the Proposed System: Providing Extra Ether for Exchange Rate Fall Down (Source: System's Evaluation Results)**

| Test Case No. | Loan Issuance Date | Loan Repayment Date | Amount Borrowed (USD/Ether) | Exchange Rate of Ether on Loan Issuance Date | Exchange Rate of Ether on Loan Repayment Date | Tokens Combination with Price (In Ether) | Total Price of Seized Collateral (Incremented Periodically Till Repayment Date) | Amount of Interest Generated (2% Daily Basis, Min. 15 Days) | Amount to be Repaid (In Ether) | Total Amount of Ether to be Paid (Interest + Base Amount) | Loss |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 01 | 17 February, 2020 | 26 March, 2020 | $500/1.865 ETH | 268.03 USD | 139.00 USD (-) | (i) 0.00373 ETH: 100 Tokens<br>(ii) 0.00799 ETH: 75 Tokens<br>(iii) 0.01623 ETH: 45 Tokens | 3.03055 ETH | 1.4547 ETH | 3.595 ETH | 5.0494 ETH | NO |
| 02 | 18 February, 2020 | 11 March, 2020 | $300/1.064 ETH | 281.95 USD | 194.22 USD (-) | (i) +0.00163 ETH: 60 Tokens<br>(ii)+0.00192 ETH: 45 Tokens<br>(iii)+0.00896 ETH: 30 Tokens | 1.43895 ETH | 0.48944 ETH | 1.545 ETH | 2.03444 ETH | NO |
| 03 | 23 February, 2020 | 18 March, 2020 | $350/1.274 ETH | 274.63 USD | 117.70 USD (-) | (i) +0.00182 ETH: 70 Tokens<br>(ii) +0.00229 ETH: 35 Tokens<br>(iii) +0.02555 ETH: 15 Tokens | 1.75179 ETH | 0.63700 ETH | 2.974 ETH | 3.61100 ETH | NO |
| 04 | 2 May, 2020 | 11 May, 2020 | $400/1.871 ETH | 213.82 USD | 184.85 USD (-) | (i) +0.00094 ETH: 80 Tokens<br>(ii) +0.00136 ETH: 45 Tokens<br>(iii) +0.01297 ETH: 18 Tokens | 2.21747 ETH | 0.56130 ETH | 2.164 ETH | 2.53820 ETH | NO |
| 05 | 10 March, 2020 | 16 March, 2020 | $700/3.487 ETH | 200.75 USD | 110.99 USD (-) | (i) +0.00070 ETH: 70 Tokens<br>(ii) +0.00102 ETH: 50 Tokens<br>(iii) +0.01179 ETH: 30 Tokens | 3.69170 ETH | 1.04610 ETH | 6.307 ETH | 6.9518 ETH | NO |
| 06 | 23 May, 2020 | 31 May, 2020 | $820/3.965 ETH | 206.80 USD | 231.54 USD (+) | (i) +0.00087 ETH: 90 Tokens<br>(ii) +0.00762 ETH: 75 Tokens | 4.26195 ETH | 1.18950 ETH | 3.965 ETH | 4.67870 ETH | NO |
| 07 | 13 May, 2020 | 30 May, 2020 | $960/4.799 ETH | 200.01 USD | 243.28 USD (+) | (i) +0.00180 ETH: 60 Tokens<br>(ii) +0.03555 ETH: 43 Tokens | 6.18284 ETH | 1.72764 ETH | 4.799 ETH | 6.52664 ETH | NO |
| 08 | 14 Feb, 2020 | 12 March, 2020 | $1100/3.870ETH | 284.23 USD | 110.60 USD (-) | (i) +0.00197 ETH: 55 Tokens<br>(ii) +0.04735 ETH: 40 Tokens | 5.57284 ETH | 2.16720 ETH | 9.945 ETH | 12.11220 ETH | NO |
| 09 | 15 April, 2020 | 29 April, 2020 | $990/6.461 ETH | 153.22 USD | 217.32 USD (+) | (i) +0.00196 ETH: 75 Tokens<br>(ii) +0.02527 ETH: 65 Tokens | 7.79250 ETH | 1.93830 ETH | 6.461 ETH | 8.39930 ETH | NO |

## 7. CONCLUSION AND FUTURE RECOMMENDATION

Our proposed collateral-based lending system supports the periodical increment of collateral price, compensating the borrower's intentional or unintentional failure to pay back the debt. As token fare raises on a daily basis, the collateral price becomes almost the same as the lent amount or surpasses it, completely eliminating the lender's financial loss. Classical loan management systems are unable to calculate the accurate credit score of their clients, as it is easy to temper any financial/bank statement of a certain period of time. As a result, loan might be provided to the defaulters. In case of our proposed blockchain based lending system, the transactions are immutable, which means that they are unchangeable. For this reason, precise risk assessment is guaranteed in our system and transparency is maintained. Furthermore, the fall of exchange rate of Ether is tackled by considering the fiat amount while returning the money, instead of the cryptocurrency amount. In conclusion, we can say that the proper utilization of this suggested protocol can definitely lessen the loan defaulting as it compensates the lender using the escalating price of tokens and its exchange rate fluctuation tackling mechanism. Moreover, our system can prove to be a better substitution to the classical insecure lending system which is vulnerable to the loan defaulters.

However, the users will carry different price-variant tokens which may cause a disarray for the system as the collateral price needs to be increased to return lender's lent amount along with the interest. Hopefully, this issue will be solved by optimizing the principle of the system in near future and is surely an open challenge for the researchers.

## REFERENCES

Ali, M., Nelson, J., Shea, R., & Freedman, M.J. (2016). Blockstack: A global naming and storage system secured by blockchains. In *2016 {USENIX} annual technical conference ({USENIX}{ATC} 16* (pp. 181–194). USENIX Association.

Burns, L., & Moro, A. (2018). What makes an ico successful? an investigation of the role of ico characteristics, team quality and market sentiment. *An Investigation of the Role of ICO Characteristics, Team Quality and Market Sentiment (September 27, 2018)*. https://dx.doi.org/10.2139/ssrn.3256512

Buterin, V. (2014). *Ethereum: A next-generation smart contract and decentralized application platform*. https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_ application_platform-vitalik-buterin.pdf

Cao, D., Husbands, K., Zhang, E., Binns, B., Kassam, A., & Lan, A. (2018). *Oneledger: Public blockchain whitepaper*. www.oneledger.io/hubfs/Website/Whitepaper/oneledger-whitepaper.en.pdf

Christidis, K., & Devetsikiotis, M. (2016). Blockchains and smart con- tracts for the internet of things. *IEEE Access, 4*, 2292– 2303. https://doi.org/10.1109/ACCESS.2016.2566339

CoinGecko. (2021). *Cryptocurrency prices & market capitalization*. https://www.coingecko.com/en/pricecharts/ethereum/usd.

Fenu, G., Marchesi, L., Marchesi, M., & Tonelli, R. (2018). The ico phenomenon and its relationships with ethereum smart contract environment. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)* (pp. 26–32). IEEE. https://doi.org/10.1109/IWBOSE.2018.8327568

Firdaus, A.H., & Nugraha, I.G.B.B. (2019). Saving and loan transaction system in cooperative using blockchain. In *2019 Inter- national Conference on ICT for Smart Society (ICISS)* (vol. 7, pp. 1–4). IEEE. https://doi.org/10.1109/ICISS48059.2019.8969847

Fröwis, M., Fuchs, A., & Böhme, R. (2019). Detecting token systems on ethereum. In *International conference on financial cryptography and data security* (pp. 93–112). Springer. https://doi.org/10.1007/978-3-030-32101-7_7

Gazali, H.M., Hassan, R., Nor, R.M., & Rahman, H.M. (2017). Re-inventing ptptn study loan with blockchain and smart contracts. In *2017 8th International Conference on Information Technology (ICIT)* (pp. 751–754). IEEE. https://doi.org/10.1109/ICITECH.2017.8079940

Guo, C., Ma, S., Wang, H., Cheng, S., & Wang, T. (2018). Loc: Poverty alleviation loan management system based on smart contracts. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)* (pp. 1527–1532). IEEE. https://doi.org/10.1109/Cybermatics_2018.2018.00257

Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., & Goldberg, S. (2017). Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*. http://dx.doi.org/10.14722/NDSS.2017.23086

Hollander, N. (2017). *Dharma: A generic protocol for tokenized debt issuance*. http://whitepaper.dharma.io.

Hrga, A., Benčić, F.M., & Žarko, I.P. (2019). Technical analysis of an initial coin offering. In *2019 15th International Conference on Telecommunications (ConTEL)* (pp. 1–8). IEEE. https://doi.org/10.1109/ConTEL.2019.8848532

Juels, A., Kosba, A., & Shi, E. (2016). The ring of gyges: Investigating the future of criminal smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 283–295). ACM Digital Library. https://doi.org/10.1145/2976749.2978362

Li, D., Fung, C., Tang, J., & Song, C. (2018). *Mixin: A free and lightning-fast peer-to-peer transactional network for digital assets*. https://mixin.one/assets/Mixin-Draft-2018-07-01.pdf

Lleshaj, L., & Korbi, A. (2020). Portfolio selection and var estimation: Evidence from western balkan countries. *Universal Journal of Accounting and Finance*, *8*(4), 92–102. https://doi.org/10.13189/ujaf.2020.080402

Luu, L., Chu, D.-H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (pp. 254–269). ACM Digital Library. https://doi.org/10.1145/2976749.2978309

Mohanta, B.K., Panda, S.S., & Jena, D. (2018). An overview of smart contract and use cases in blockchain technology. In *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)* (p. 14). IEEE. https://doi.org/10.1109/ICCCNT.2018.8494045

Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press.

Okoye, M.C. & Clark, J. (2018). Toward cryptocurrency lending. In *International Conference on Financial Cryptography and Data Security* (pp. 367–380). Springer. https://doi.org/10.1007/978-3-662-58820-8_25

Rosic, A. (2017). What is an initial coin offering? raising millions in seconds. https://blockgeeks.com/guides/initial-coin-offering/

Sinaga, J.S., Muda, I., & Silalahi, A.S. (2020). The effect of bi rate, exchange rate, inflation and third party fund (dpk) on credit distribution and its impact on non performing loan (npl) on xyz commercial segment bank. *Universal Journal of Accounting and Finance*, *8*(3), 55–64. https://doi.org/10.13189/ujaf.2020.080301

Singh, S., & Singh, N. (2016). Blockchain: Future of financial and cyber security. In *2016 2nd international conference on contemporary computing and informatics (IC3I)* (pp. 463–467). IEEE. https://doi.org/10.1109/IC3I.2016.7918009

Somin, S., Gordon, G., Pentland, A., Shmueli, E., & Altshuler, Y. (2020). Erc20 transactions over ethereum blockchain: Network analysis and predictions. *arXiv preprint arXiv:2004.08201*. https://arxiv.org/abs/2004.08201

Swan, M. (2015). *Blockchain: Blueprint for a new economy*. O'Reilly Media, Inc.

Victor, F., & Lüders, B.K. (2019). Measuring ethereum-based erc20 token networks. In *International Conference on Financial Cryptography and Data Security* (pp. 113–129). Springer. https://doi.org/10.1007/978-3-030-32101-7_8

Watanabe, H., Fujimura, S., Nakadaira, A., Miyazaki, Y., Akutsu, A., & Kishigami, J.J. (2015). Blockchain contract: A complete consensus using blockchain. In *2015 IEEE 4th global conference on consumer electronics (GCCE)* (pp. 577–578). IEEE. https://doi.org/10.1109/GCCE.2015.7398721

Yang, X., Chen, Y., & Chen, X. (2019). Effective scheme against 51% attack on proof-of-work blockchain with history weighted information. In *2019 IEEE International Conference on Blockchain (Blockchain)* (pp. 261–265). IEEE. https://doi.org/10.1109/Blockchain.2019.00041

Yuan, Y., & Wang, F. (2016). Development status and prospect of blockchain technology. *Journal of automation*, *42*(4), 481–494.

Zetzsche, D.A., Buckley, R.P., Arner, D.W., & Föhr, L. (2017). The ico gold rush: It's a scam, it's a bubble, it's a super challenge for regulators. *University of Luxembourg Law Working Paper*, *11*, 17–83. https://dx.doi.org/10.2139/ssrn.3072298