OPTIMIZATION OF E-COMMERCE PLATFORM STRUCTURE BASED ON BLOCKCHAIN TECHNOLOGY

A PROJECT REPORT

Submitted by

ARUNA. M [REGISTER NO:211417104024]

HARINI.T.M [REGISTER NO:211417104080]

KRITHIKAA.K [REGISTER NO:211417104124]

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE, CHENNAI-600123.

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2021

BONAFIDE CERTIFICATE

Certified that this project report **"… .....Structure Optimization of e-Commerce Platform Based on Artificial Intelligence and Blockchain Technology....."** is the bonafide work of " **…Aruna.M (211417104024) ,Harini.T.M (211417104080) Krithikaa.K(211417104124)**..." who carried out the project work under my supervision.

SIGNATURE

Dr.S.MURUGAVALLI,M.E.,Ph.D.,
HEAD OF THE DEPARTMENT

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

Mrs.C.Vijayalakshmi
SUPERVISOR

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the Anna University Project Viva-Voce Examination held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like express our heartfelt and sincere thanks to our Directors **Tmt.C.VIJAYARAJESWARI**, **Dr.C.SAKTHIKUMAR, M.E., Ph.D.,** and **Tmt. SARANYASREE SAKTHIKUMAR B.E.,M.B.A.,** for providing us with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K.Mani, M.E., Ph.D.** for his timely concern and encouragement provided to us throughout the course.

We thank the HOD of CSE Department, **Dr. S.MURUGAVALLI , M.E.,Ph.D.,** for the support extended throughout the project.

We would like to thank my **Project Guide** ……**Mrs.C.Vijayalakshmi**…… and all the faculty members of the Department of CSE for their advice and suggestions for the successfulcompletion of the project.

NAME OF THE STUDENTS

Aruna.M

Harini.T.M

Krithikaa.K

ABSTRACT

E-commerce has become more and more popular because of rich products, fast transactions, and free from time, locations, stores and so on. However, the disclosure of users' personal information such as 1) identities, 2) addresses, and 3) phone numbers has become a big concern of online activity. Existing E-commerce models are trapped in a dilemma between the proof of ownership and privacy protection. To address this issue, in this paper we design a privacy-preserving business protocol by employing private smart contracts in the negotiation phase. The protocol allows counterparties make deals without the disclosure of private information such as identities, addresses, and phone numbers. Moreover, we employ the zero- knowledge proof to guarantee the ownership.

TABLE OF CONTENTS

CHAPTER NO. **TITLE** **PAGE NO.**

1. INTRODUCTION

1.1 Overview

E-commerce has become more and more popular because of rich products, fast transactions, and free from time, locations, stores and so on. However, the disclosure of users' personal information such as 1) identities, 2) addresses, and 3) phone numbers has become a big concern of online activity . Factually, it has formed a huge "gray industry" that seriously endangers users' safety and privacy. It is not uncommon for sellers to threaten and enforce customers to make, modify or delete product reviews which are against their wills. Meanwhile, online shopping websites are also suffering from malicious bad reviews or spurious praise reviews , which reduce the Quality of Experience (QoE). There are already some researches and practices addressing these issues. Janice Tsai et al. show that consumers are more willing to pay a premium for privacy protection. Berendt et al. propose PET which provides more timely privacy protection and trust tools for Web applications. Li et al. present a coherent adaptive trust model based on a transaction feedback system to quantify and compare the credibility of users in peer-to-peer (P2P) e-commerce communities. Vandervort proposes three different models by which a reputation system could be implemented based on the blockchain technology. However, none of them consider the issue of privacy disclosure of buyers during the delivery of products. In order to address privacy issues in product delivery, "Privacy Waybill" is invented to hide consumer's information so that private information will not appear in the waybill. Only couriers can obtain recipients' information including identities, addresses, and phone numbers through authorized devices. Although it prevents the disclosure of personal information in some extent, this technology cannot hide the addresses, phone numbers and other users' information from the sellers. As far as we know, there are no existing models protecting private information from beginning to end. In this paper, we address this problem by focusing on the trade and certification of collections such as artworks, luxuries and limited edition products. These goods are much easier to encounter privacy issues in E-commerce since buyers are usually more desired to protect their private information such as identities or addresses in these transactions. We would like to build a unified platform for collectible

collections and items to exchange, evaluate and certificate their ownership. Just like the CryptoKitties.co does that on the Fancy Cat which is maintained on blockchain, our platform aims to give, certificate and protect the value of these “collections”. We protect users’ private information by employing the blockchain technologies including private smart contracts and zero-knowledge proof. Specifically, each trade is represented by a private smart contract, which defines the business logics, types of trade, counterparties, underlying assets, price, and any other relevant information.

1.2 Problem Definition

E-Commerce has become more and more popular because of rich products, fast transactions, and free from time, locations, stores, and so on. However, the disclosure of personal data such as their IDs, addresses, and phone numbers has become a major concern for online activities. The current e-commerce model is at the crossroads of ownership and privacy. To address this, this article creates an enterprise protocol that uses smart personal contracts to protect privacy during the negotiation phase. This protocol allows contracting parties to conduct business without disclosing personal information such as identity, address, and phone number. Furthermore, we employ the zero-knowledge proof to ensure ownership.

2. LITERATURE SURVEY

2.1 A Reputation-Based Trust Model for Peer-to-Peer eCommerce Communities

Peer-to-Peer eCommerce communities are commonly perceived as an environment offering both opportunities and threats. One way to minimize threats in such an open community is to use community-based reputations to help evaluating the trustworthiness and predicting the future behavior of peers. This paper presents PeerTrust a coherent adaptive trust model for quantifying and comparing the trustworthiness of peers based on a transaction-based feed-back system. There are two main features of our model. First, we introduce three basic trust parameters in computing trustworthiness of peers. In addition to feedback a peer receives through its transactions with other peers, we in- corporate the total number of transactions a peer performs, and the credibility of the feedback sources into the model for evaluating the trustworthiness of peers. We argue that the trust models based solely on feedback from other peers in the community is inaccurate and ineffective. Second, we introduce two adaptive trust factors, the transaction context factor and the community context factor, to allow the basic trust metric to incorporate different contexts (situations) and to address common problems encountered in a variety of online eCommerce communities. We present a concrete method to validate the proposed trust model and report the set of simulation-based experiments, showing the feasibility and benefit of the PeerTrust model.

Drawbacks:

- There is an extensive amount of research focused on building trust for electronic markets through trusted third parties or intermediaries. However, it is not applicable to P2P eCommerce communities where peers are equal in their roles and are independent entities, thus no peers can serve as trusted third parties or intermediaries.

Author: Li Xiong, Ling Liu.

Published In: 2003

2.2 Challenges and Opportunities Associated with a Bitcoin-Based Transaction Rating System

It has been shown that seller ratings given by previous buyers give new customers useful information when making purchasing decisions. Bitcoin, however, is designed to obfuscate the link between buyer and seller with a layer of limited anonymity, thus preventing buyers from finding or validating this information. While this level of anonymity is valued by the Bitcoin community, as Bitcoin moves toward greater adoption there will be pressure from buyers who wish to know more about who they are doing business with, and sellers who consider their reputation a strong selling point, to allow greater transparency. We consider three different models by which a reputation/rating system could be implemented in conjunction with Bitcoin transactions and consider pros and cons of each. We find that each presents challenges on both the technological and social fronts.

Drawbacks:

- The Bitcoin community points to this anonymity, often referred to as pseudonymity because it is not absolute, as an asset, a way of circumventing surveillance and cumbersome regulatory regimes. It is also considered a defense against the user profiling/data mining practiced by large merchants such as K-Mart.

Author: David Vandervort

Published In: 2014

2.3 Zerocash: Decentralized Anonymous Payments from Bitcoin

Bitcoin is the first digital currency to see widespread adoption. While payments are conducted between pseudonyms, Bitcoin cannot offer strong privacy guarantees: payment transactions are recorded in a public decentralized ledger, from which much information can be deduced. Zerocoin (Miers et al., IEEE S&P 2013) tackles some of these privacy issues by unlinking transactions from the payment's origin. Yet, it still reveals payments' destinations and amounts, and is limited in functionality. In this paper, we construct a full-fledged ledger-based digital currency with strong privacy guarantees. Our results leverage recent

advances in zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) First, we formulate and construct decentralized anonymous payment schemes (DAP schemes). A DAP scheme enables users to directly pay each other privately: the corresponding transaction hides the payment's origin, destination, and transferred amount. We provide formal definitions and proofs of the construction's security. Second, we build Zerocash, a practical instantiation of our DAP scheme construction. In Zerocash, transactions are less than 1 kB and take under 6 ms to verify — orders of magnitude more efficient than the less-anonymous Zerocoin and competitive with plain Bitcoin.

Drawback:

- Zerocoin does not rely on digital signatures to validate coins, nor does it require a central bank to prevent double spending. Instead, Zerocoin authenticates coins by proving, in zero- knowledge, that they belong to a public list of valid coins (which can be maintained on the block chain).

Author: Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, Madars Virza.

Published In: 2014

2.4 The Effect of Online Privacy Information on Purchasing Behavior: An Experimental Study

Although online retailers detail their privacy practices in online privacy policies, this information often remains invisible to consumers, who seldom make the effort to read and understand those policies. This paper reports on research undertaken to determine whether a more prominent display of privacy information will cause consumers to incorporate privacy considerations into their online purchasing decisions. We designed an experiment in which a shopping search engine interface clearly and compactly displays privacy policy information. When such information is made available, consumers tend to purchase from online retailers who better protect their privacy. In fact, our study indicates that when privacy information is made more salient and accessible, some consumers are willing to pay a premium to purchase from privacy protective websites. This result suggests that businesses may be able to

leverage privacy protection

Drawbacks:

- After the purchase, the consumer may not know how the merchant will use the personal information she revealed as part of the transaction (Acquisti and Grossklags 2005b). This lack of information arguably affects individual behavior in different ways.

Author: Janice Y. Tsai, Serge Egelman, Lorrie Cranor and Alessandro Acquisti

Publlished In: 2016

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

Green et al. show how to construct anonymous payment channels between two mutually distrustful parties. Their protocols are instantiated using efficient cryptographic primitives with no trusted third parties. Payments of arbitrary value are conducted directly between parties, or via an intermediate connection who learns neither the participants identities nor the amount involved. Coupled with a decentralized anonymous payment scheme for funding the channels, they provide for private instantaneous anonymous payments without a trusted bank. They use the idea of lightning networks and address the anonymity problem of payment channels based on the zero-knowledge technology. Gallay et al. design a novel platform for decentralized logistics, the aim of which is to magnify and accelerate the impact offered by the integration of the most recent advances in Information and Communication Technologies (ICTs) to multi-modal freight operations. Besides, the platform allows for an implementation that is not affected by scalability issues and is not limited by geographical borders. They also employ the IDS and blockchain technology to construct a decentralized logistics platform.

DISADVANTAGES OF EXISTING SYSTEM:

- However, they did not consider the payment problem in the event of a dispute.
- The platform allows for an implementation that is not affected by scalability issues and is not limited by geographical borders.

3.2 PROPOSED SYSTEM:

We make the following contributions:

We design a privacy-preserving model for E-commerce based on the private smart contract technology. Our model can provide the proof of ownership while protect users' private information from other participants. Besides, we build implementations of model using two existing blockchain application platforms. We also evaluate the performance and validate its effectiveness and efficiency of the model with experiments. Performance analysis of the blockchain platforms provides further considerations for deploying a usable implementation.

ADVANTAGES OF PROPOSED SYSTEM:

- We employ a escrow protocol to address the problem of dispute. In addition, our model is based on a permissioned blockchain which in nature has superior performance efficiency to public chains.
- We further tap the potential of blockchain and expand it to E-commerce systems based on permissioned blockchain. Additionally, we enhance the anonymity of the logistics model to better protect users' privacy.

3.3 Requirements Analysis and Specification

3.3.1 Input Requirements:

3.3.2 Output Requirements:

3.3.3 Functional Requirements

Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatibility and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements

Hardware requirements for present project:

- System : Intel Pentium.

- Hard Disk : 120 GB.
- Monitor : 15'' LED
- Input Devices : Keyboard, Mouse
- Ram : 2 GB

Software Requirements:

Software Requirements deal with defining software resource requirements and pre-requisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or pre-requisites are generally not included in the software installation package and need to be installed separately before the software is installed.

Software requirements for present project:

- Operating system : Windows 7 or more
- Coding Language : Python
- Backend : Anaconda

3.4 Technology Stack

PYTHON:

Python is an interpreted, high-level, general-purpose programming language. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

4.5 Software Description:

PYTHON:

Python is a multi-paradigm programming language. Object-oriented

programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming. Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter map and `reduce` functions; list comprehensions, dictionaries, sets and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more

than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it" design philosophy. Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture."

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name a tribute to the British comedy group Monty Python and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonistas.

Libraries

Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating

pseudorandom numbers, arithmetic with arbitrary precision decimals, manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications, but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of March 2018, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 130,000 packages with a wide range of functionality, including:

- Graphical user interfaces
- Web frameworks
- Multimedia
- Databases
- Networking
- Test frameworks
- Automation
- Web scraping
- Documentation
- System administration
- Scientific computing
- Text processing
- Image processing

Development

Python's development is conducted largely through the Python Enhancement Proposal (PEP) process, the primary mechanism for proposing major new features, collecting community input on issues and documenting Python design decisions. Python coding style is covered in PEP 8. Outstanding PEPs are reviewed

and commented on by the Python community and the steering council.

Enhancement of the language corresponds with development of the CPython reference implementation. The mailing list python-dev is the primary forum for the language's development. Specific issues are discussed in the Roundup bug tracker maintained at python.org. Development originally took place on a self-hosted source-code repository running Mercurial, until Python moved to GitHub in January 2017.

CPython's public releases come in three types, distinguished by which part of the version number is incremented:

- Backward-incompatible versions, where code is expected to break and need to be manually ported. The first part of the version number is incremented. These releases happen infrequently for example, version 3.0 was released 8 years after 2.0.
- Major or "feature" releases, about every 18 months, are largely compatible but introduce new features. The second part of the version number is incremented. Each major version is supported by bugfixes for several years after its release.
- Bugfix releases, which introduce no new features, occur about every 3 months and are made when a sufficient number of bugs have been fixed upstream since the last release. Security vulnerabilities are also patched in these releases. The third and final part of the version number is incremented.

Many alpha, beta, and release-candidates are also released as previews and for testing before final releases. Although there is a rough schedule for each release, they are often delayed if the code is not ready. Python's development team monitors the state of the code by running the large unit test suite during development, and using the BuildBot continuous integration system.

The community of Python developers has also contributed over 86,000 software modules (as of 20 August 2016) to the Python Package Index (PyPI),

the official repository of third-party Python libraries.

The major academic conference on Python is PyCon. There are also special Python mentoring programmes, such as Pyladies.

4. System Design

4.1. ER diagram
4.2 Data dictionary
4.3 Table Normalization
4.4 UML Diagrams

UML stands for **U**nified **M**odeling **L**anguage. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

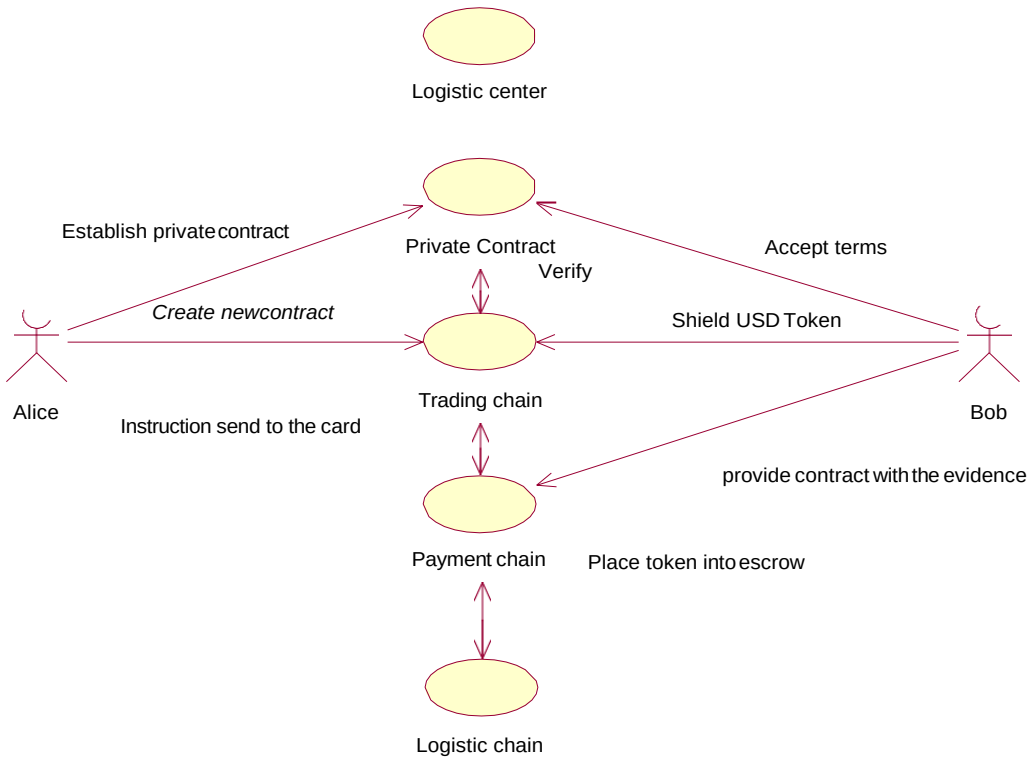
1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.

5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

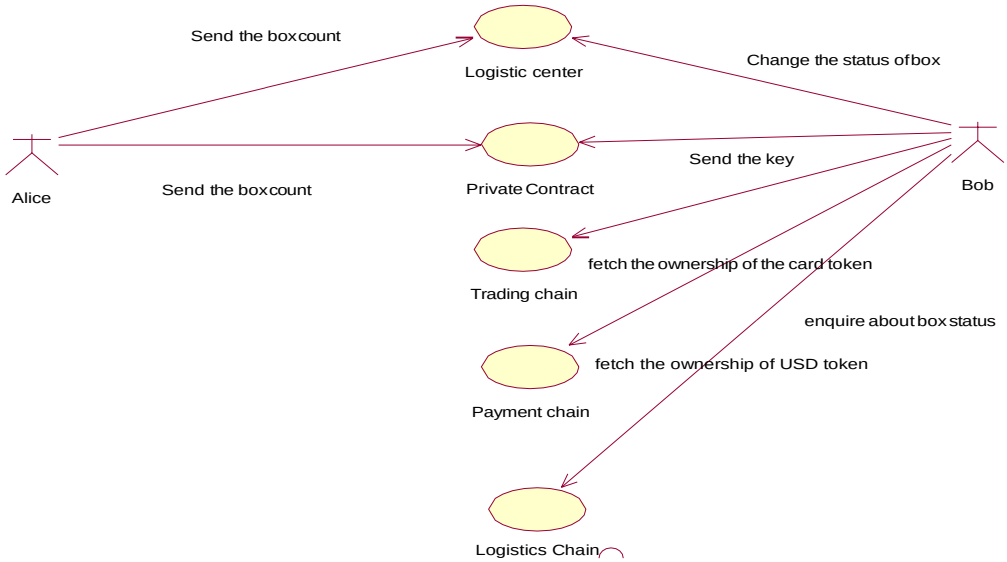
Use Case Diagram:

A Use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

Negotiation phase

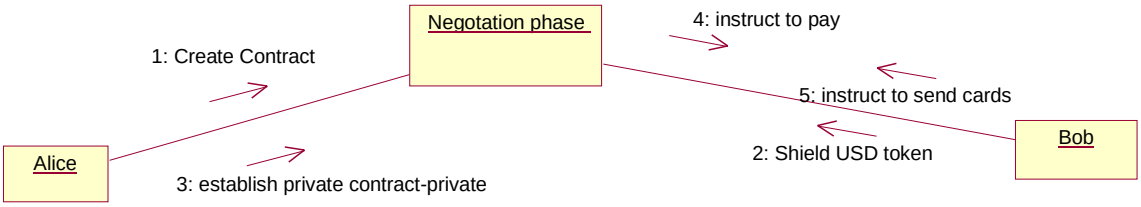


Delivery Phase

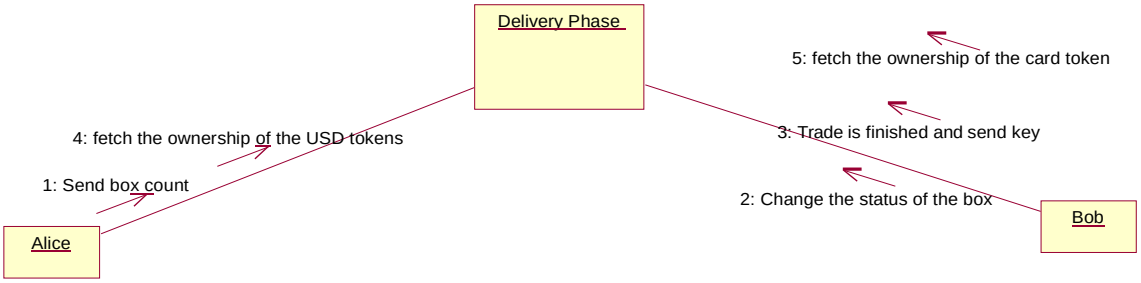


COLLABORATION DIAGRAM

Negotiation Phase



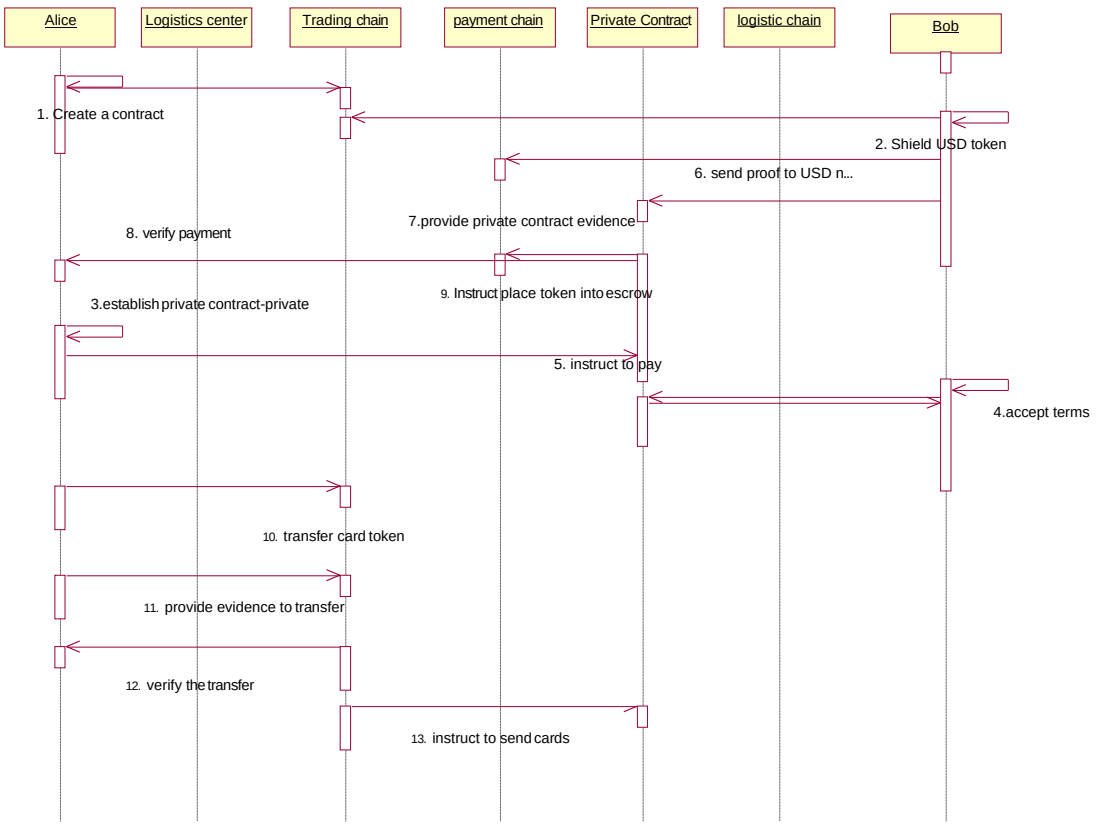
Delivery Phase



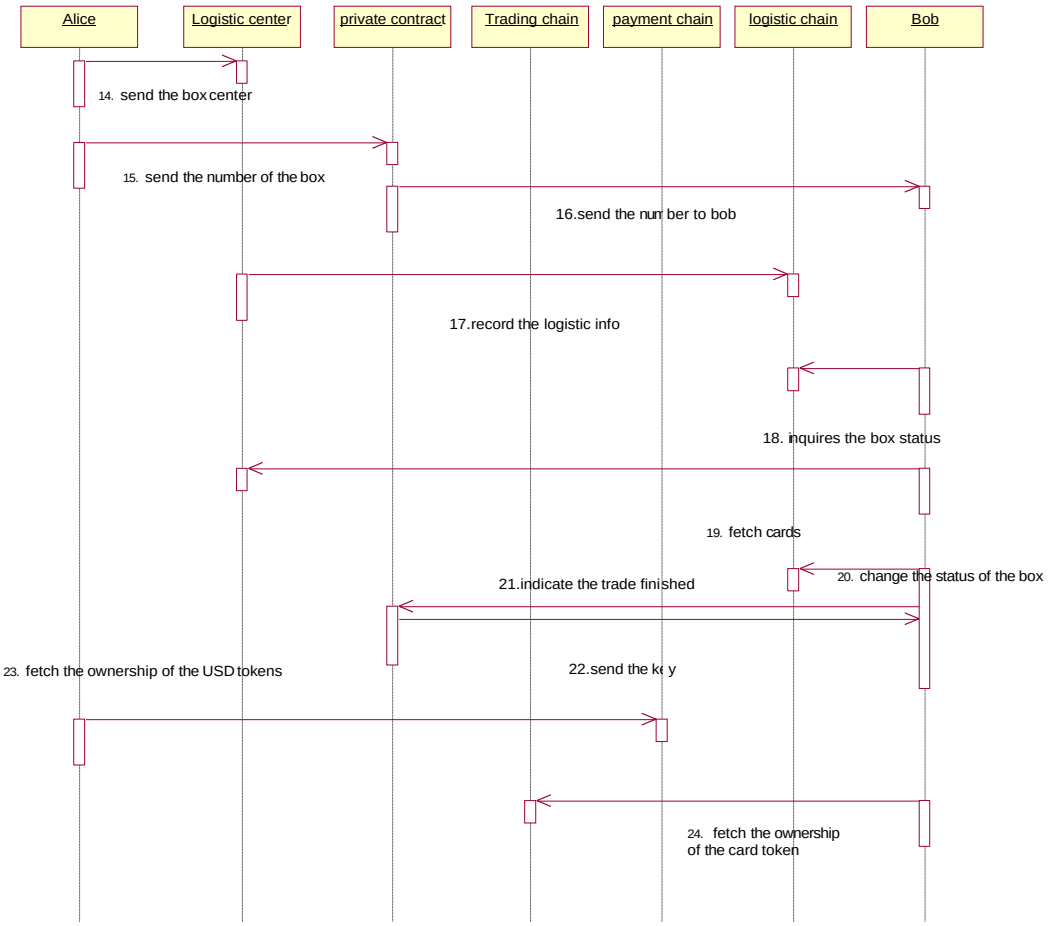
SEQUENCE DIAGRAM:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

Negotiation Phase



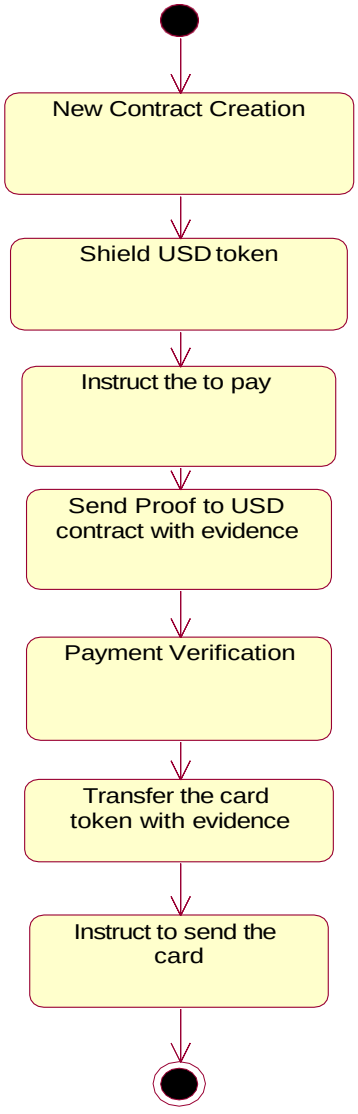
Delivery Phase



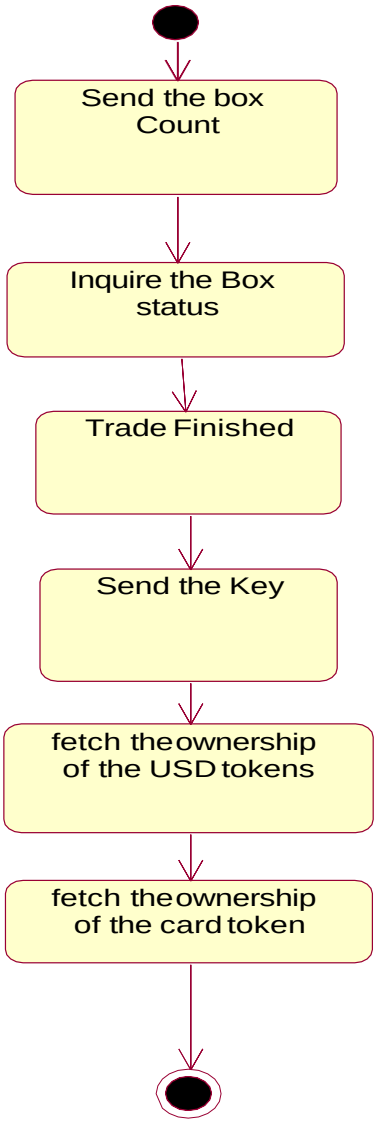
ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

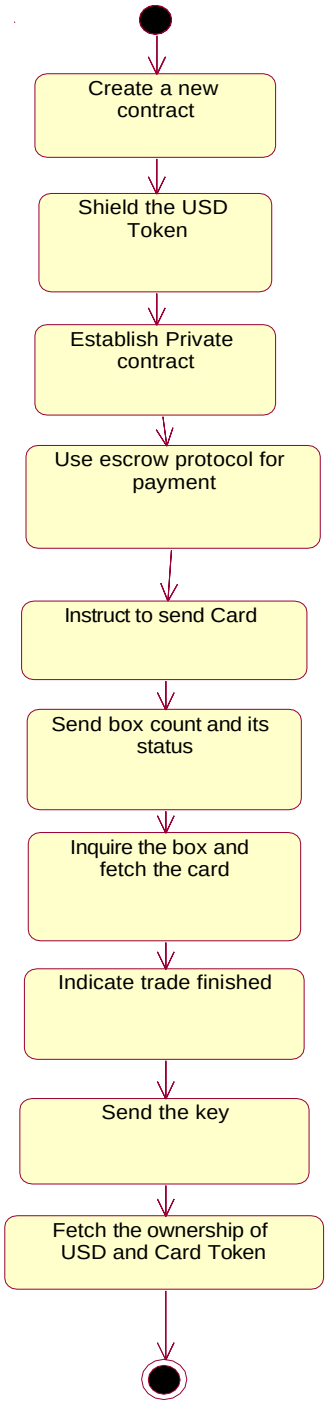
Negotiation Phase



Delivery Phase

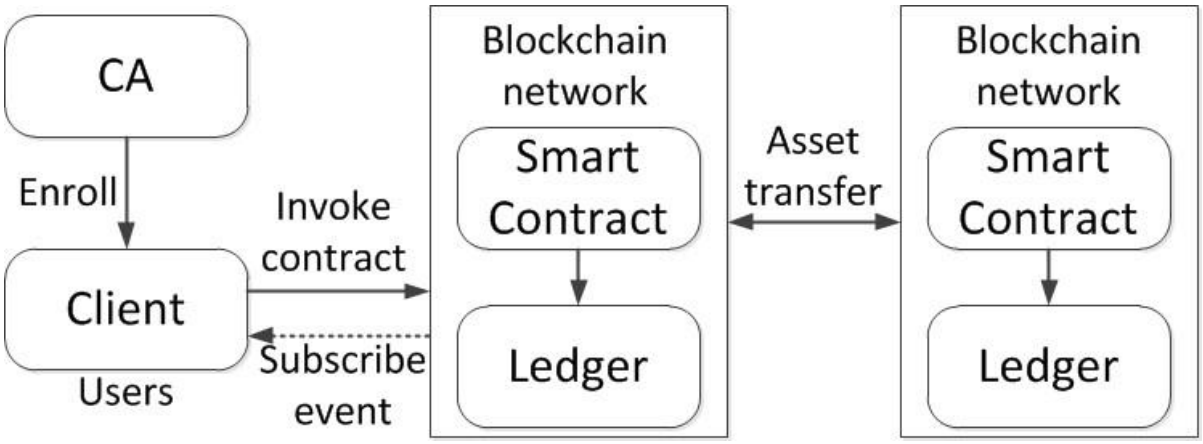


STATE CHART DIAGRAM



5. SYSTEM IMPLEMENTATION

5.1 Architecture Overview:



Modules in the project

The entire project mainly consists of 4 modules, which are

- ❖ Central Authority
- ❖ Token Creation
- ❖ Negotiation phase
- ❖ Delivery phase

Central Authority:

The CA component is in charge of issuing PKI-based certificates to organization members and their users. A root certificate (rootCert) is issued to each member and one enrollment certificate (ECert) to each authorized user. The Client interacts with the blockchain network and smart contracts. It has to obtain a valid identity certificate from CA before joining the application channel/chains in the network. Both users and intelligent logistics centers act as clients.

Token Creation:

Thus, Alice has to create a note-contract for the set of collector cards and issue a shielded token for herself . Now Alice owns a CARD token. At a same time, Bob shields some USD tokens by the USD note- contract.

Negotiation phase:

Alice establishes a private contract with Bob in a private channel . The private contract specifies the trade of the cards at a specific price in USD between Alice and Bob. The private contract also refers to the cards and USD note- contracts. Besides, the private contract also receives the rel- evant public keys and payment addresses of the two parties (including the Hash of physical addresses). When Alice initializes the contract, Bob can send to the private contract a transaction indicating acceptance of the terms We assume that the USD must be paid first. After the private contract receives the confirmation transaction, the private contract issues an instruction to Bob to pay the relevant amount of USD to Alice (Bob places the USD tokens into the escrow.USD tokens to a mediator's payment address by generating the necessary zk-SNARK proof and sends it to the USD note-contract(Miner).

Delivery phase:

Alice places the cards into a delivery box which has a unique number.Then Alice sends the delivery box to the intelligent logistics center (suppose the intelligent center is reliable). Transport companies are responsible for inspecting and monitoring the legitimacy of the items. Alice's client sends the Hash of Bob's address to this box and sends the number of the box to the private contract. The private contract then sends the number of the delivery box to Bob's client.

5.2 Program Design Language:

6.SYSTEM IMPLEMENTATION

6.1 Client-side coding

Blockchain.py

```
import OrderedDict
import binascii
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
```

```
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5

import hashlib
import json
from time import time
from urllib.parse import urlparse
from uuid import uuid4

import requests
from flask import Flask, jsonify, request, render_template
from flask_cors import CORS

MINING_SENDER = "THE BLOCKCHAIN"
MINING_REWARD = 1
MINING_DIFFICULTY = 2

class Blockchain:

    def __init__(self):

        self.transactions = []
        self.chain = []
        self.nodes = set()
        #Generate random number to be used as node_id
        self.node_id = str(uuid4()).replace('-', '')
        #Create genesis block
```

```
    self.create_block(0, '00')


def register_node(self, node_url):
    """
    Add a new node to the list of nodes
    """
    #Checking node_url has valid format
    parsed_url = urlparse(node_url)
    if parsed_url.netloc:
        self.nodes.add(parsed_url.netloc)
    elif parsed_url.path:
        # Accepts an URL without scheme like '192.168.0.5:5000'.
        self.nodes.add(parsed_url.path)
    else:
        raise ValueError('Invalid URL')


def verify_transaction_signature(self, sender_address, signature, transaction):
    """
    Check that the provided signature corresponds to transaction
    signed by the public key (sender_address)
    """
    public_key = RSA.importKey(binascii.unhexlify(sender_address))
    verifier = PKCS1_v1_5.new(public_key)
    h = SHA.new(str(transaction).encode('utf8'))
    return verifier.verify(h, binascii.unhexlify(signature))
```

```
def submit_transaction(self, sender_address, recipient_address, value, signature):
    """
    Add a transaction to transactions array if the signature verified
    """
    transaction = OrderedDict({'sender_address': sender_address,
                               'recipient_address': recipient_address,
                               'value': value})

    #Reward for mining a block
    if sender_address == MINING_SENDER:
        self.transactions.append(transaction)
        return len(self.chain) + 1
    #Manages transactions from wallet to another wallet
    else:
        transaction_verification = self.verify_transaction_signature(sender_address, signature,
 transaction)
        if transaction_verification:
            self.transactions.append(transaction)
            return len(self.chain) + 1
        else:
            return False



def create_block(self, nonce, previous_hash):
    """
```

```
    Add a block of transactions to the blockchain
    """
    block = {'block_number': len(self.chain) + 1,
             'timestamp': time(),
             'transactions': self.transactions,
             'nonce': nonce,
             'previous_hash': previous_hash}

    # Reset the current list of transactions
    self.transactions = []

    self.chain.append(block)
    return block


def hash(self, block):
    """
    Create a SHA-256 hash of a block
    """
    # We must make sure that the Dictionary is Ordered, or we'll have inconsistent hashes
    block_string = json.dumps(block, sort_keys=True).encode()

    return hashlib.sha256(block_string).hexdigest()


def proof_of_work(self):
    """
```

```
    Proof of work algorithm
    """
    last_block = self.chain[-1]
    last_hash = self.hash(last_block)

    nonce = 0
    while self.valid_proof(self.transactions, last_hash, nonce) is False:
        nonce += 1

    return nonce


def valid_proof(self, transactions, last_hash, nonce, difficulty=MINING_DIFFICULTY):
    """
    Check if a hash value satisfies the mining conditions. This function is used within the
 proof_of_work function.
    """
    guess = (str(transactions)+str(last_hash)+str(nonce)).encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    return guess_hash[:difficulty] == '0'*difficulty


def valid_chain(self, chain):
    """
    check if a bockchain is valid
    """
    last_block = chain[0]
```

```
    current_index = 1

    while current_index < len(chain):
        block = chain[current_index]
        #print(last_block)
        #print(block)
        #print("\n-----------\n")
        # Check that the hash of the block is correct
        if block['previous_hash'] != self.hash(last_block):
            return False

        # Check that the Proof of Work is correct
        #Delete the reward transaction
        transactions = block['transactions'][:-1]
        # Need to make sure that the dictionary is ordered. Otherwise we'll get a different hash
        transaction_elements = ['sender_address', 'recipient_address', 'value']
        transactions = [OrderedDict((k, transaction[k]) for k in transaction_elements) for transaction in
transactions]

        if not self.valid_proof(transactions, block['previous_hash'], block['nonce'],
MINING_DIFFICULTY):
            return False

        last_block = block
        current_index += 1

    return True
```

```
def resolve_conflicts(self):
    """
    Resolve conflicts between blockchain's nodes
    by replacing our chain with the longest one in the network.
    """
    neighbours = self.nodes
    new_chain = None

    # We're only looking for chains longer than ours
    max_length = len(self.chain)

    # Grab and verify the chains from all the nodes in our network
    for node in neighbours:
        print('http://' + node + '/chain')
        response = requests.get('http://' + node + '/chain')

        if response.status_code == 200:
            length = response.json()['length']
            chain = response.json()['chain']

            # Check if the length is longer and the chain is valid
            if length > max_length and self.valid_chain(chain):
                max_length = length
                new_chain = chain

    # Replace our chain if we discovered a new, valid chain longer than ours
    if new_chain:
```

```
            self.chain = new_chain
            return True

        return False

# Instantiate the Node
app = Flask(__name__)
CORS(app)

# Instantiate the Blockchain
blockchain = Blockchain()

@app.route('/')
def index():
    return render_template('./index.html')

@app.route('/configure')
def configure():
    return render_template('./configure.html')



@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.form

    # Check that the required fields are in the POST'ed data
```

```
    required = ['sender_address', 'recipient_address', 'amount', 'signature']
    if not all(k in values for k in required):
        return 'Missing values', 400
    # Create a new Transaction
    transaction_result = blockchain.submit_transaction(values['sender_address'],
     values['recipient_address'], values['amount'], values['signature'])


    if transaction_result == False:
        response = {'message': 'Invalid Transaction!'}
        return jsonify(response), 406
    else:
        response = {'message': 'Transaction will be added to Block '+ str(transaction_result)}
        return jsonify(response), 201


@app.route('/transactions/get', methods=['GET'])
def get_transactions():
    #Get transactions from transactions pool
    transactions = blockchain.transactions


    response = {'transactions': transactions}
    return jsonify(response), 200


@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
```

```
    }
    return jsonify(response), 200


@app.route('/mine', methods=['GET'])
def mine():
    # We run the proof of work algorithm to get the next proof...
    last_block = blockchain.chain[-1]
    nonce = blockchain.proof_of_work()


    # We must receive a reward for finding the proof.
    blockchain.submit_transaction(sender_address=MINING_SENDER,
     recipient_address=blockchain.node_id, value=MINING_REWARD, signature="")


    # Forge the new Block by adding it to the chain
    previous_hash = blockchain.hash(last_block)
    block = blockchain.create_block(nonce, previous_hash)


    response = {
        'message': "New Block Forged",
        'block_number': block['block_number'],
        'transactions': block['transactions'],
        'nonce': block['nonce'],
        'previous_hash': block['previous_hash'],
    }
    return jsonify(response), 200
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
```

```
    values = request.form
    nodes = values.get('nodes').replace(" ", "").split(',')

    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': [node for node in blockchain.nodes],
    }
    return jsonify(response), 201


@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
```

```
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }
    return jsonify(response), 200


@app.route('/nodes/get', methods=['GET'])
def get_nodes():
    nodes = list(blockchain.nodes)
    response = {'nodes': nodes}
    return jsonify(response), 200



if __name__ == '__main__':
    from argparse import ArgumentParser

    parser = ArgumentParser()
    parser.add_argument('-p', '--port', default=5000, type=int, help='port to listen on')
    args = parser.parse_args()
    port = args.port

        app.run(host='127.0.0.1', port=port)
```

6.2 Server-side coding

Blockchain_client.py

```
from collections import OrderedDict

import binascii
import functools
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from dbconnect import connection

import requests
from flask import Flask, jsonify, request, render_template


class Transaction:

    def __init__(self, sender_address, sender_private_key, recipient_address, value):
        self.sender_address = sender_address
        self.sender_private_key = sender_private_key
        self.recipient_address = recipient_address
        self.value = value

    def __getattr__(self, attr):
        return self.data[attr]

    def to_dict(self):
        return OrderedDict({'sender_address': self.sender_address,
                            'recipient_address': self.recipient_address,
                            'value': self.value})

    def sign_transaction(self):
        """
        Sign transaction with private key
        """
        private_key = RSA.importKey(binascii.unhexlify(self.sender_private_key))
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf8'))
```

```
        return binascii.hexlify(signer.sign(h)).decode('ascii')



app = Flask(__name__)

@app.route('/')
def index():
        return render_template('./index.html')

@app.route('/make/transaction')
def make_transaction():
   return render_template('./make_transaction.html')
@app.route('/make/transaction1')
def make_transaction1():
   return render_template('./make_transaction1.html')
@app.route('/make/transaction2')
def make_transaction2():
   return render_template('./make_transaction2.html')
@app.route('/contract/private')
def make_transaction3():
   return render_template('./privatecontract.html')

@app.route('/view/transactions')
def view_transaction():
   return render_template('./view_transactions.html')

@app.route('/wallet/new', methods=['GET'])
def new_wallet():
        random_gen = Crypto.Random.new().read
        private_key = RSA.generate(1024, random_gen)
        public_key = private_key.publickey()
        response = {
                'private_key': binascii.hexlify(private_key.exportKey(format='DER')).decode('ascii'),
                'public_key': binascii.hexlify(public_key.exportKey(format='DER')).decode('ascii')
        }

        return jsonify(response), 200

@app.route('/generate/transaction', methods=['POST'])
def generate_transaction():
   sender_address = request.form['sender_address']
   sender_private_key = request.form['sender_private_key']
   recipient_address = request.form['recipient_address']
```

```
value = request.form['amount']
pid = request.form['pid']
c, conn = connection()
sql='select amount from  product where id="%s"'% \
        (pid)
c.execute(sql);
amount=0;
result=c.fetchall();
count=0;
for cc in result:
    count=1
    amount=cc
if(count==1):
    sql='select amount from  usdtoken where sid="%s"'% \
         (sender_address)
    c.execute(sql);
    result1=c.fetchall();
    count=0;
    amm=0
    count=0
    for cc1 in result1:
        count=1
        amm=cc1
    sql='select amount from  usdtoken where sid="%s"'% \
         (recipient_address)
    c.execute(sql);
    result1=c.fetchall();
    count=0;
    amm1=0
    count1=0
    for cc1 in result1:
        count1=1
        amm1=cc1
    if(count==1):
        res = functools.reduce(lambda sub, ele: sub * 10 + ele, amm)
        res1 = functools.reduce(lambda sub, ele: sub * 10 + ele, amount)

        res2 = functools.reduce(lambda sub, ele: sub * 10 + ele, amm1)
        result=int(res1)-int(res)
        result2=int(res2)+int(res)
        sql1='update usdtoken set amount="%s" where sid="%s"' % \
                (result,sender_address)
        c.execute(sql1)
        conn.commit()
```

```
        sql1='update usdtoken set amount="%s" where sid="%s"' % \
                (result2,recipient_address)
        c.execute(sql1)
        conn.commit()
        sql1='insert into trans(sid,rid,pid) values("%s", "%s","%s")' % \
                (sender_address,recipient_address,pid)
        c.execute(sql1)
        conn.commit()
        conn.close()
        value=res+" "+pid
    transaction = Transaction(sender_address, sender_private_key, recipient_address, value)
    response = {'transaction': transaction.to_dict(), 'signature': transaction.sign_transaction(),'pid': pid}
    return jsonify(response), 200

@app.route('/contract/smart', methods=['POST'])
def smart_contract():

        sender_address = request.form['sender_address']
        sender_private_key = request.form['sender_private_key']
        recipient_address = request.form['recipient_address']
        value = request.form['amount']
        c, conn = connection()
        sql1='insert into smartcontract(sid,key1,rid,pid,status) values("%s", "%s","%s","%s","%s")' % \
          (sender_address,sender_private_key,recipient_address,value,"Deliverd")
        c.execute(sql1)
        conn.commit()
        conn.close()
        value=value+"   Deliverd";
        transaction = Transaction(sender_address, sender_private_key, recipient_address, value)
        response = {'transaction': transaction.to_dict(), 'signature': transaction.sign_transaction()}
        return jsonify(response), 200


@app.route('/generate/transaction1', methods=['POST'])
def generate_transaction1():
    sender_address = request.form['sender_address']
    sender_private_key = request.form['sender_private_key']
    product = request.form['name']
    quantity = request.form['quan']
    amount = request.form['amount']
    value=product+","+str(quantity)+","+str(amount)
    transaction = Transaction(sender_address, sender_private_key, sender_address, value)
    c, conn = connection()
    sql='select * from  register where master="%s" AND private="%s"% \
```

```
            (sender_address,sender_private_key)
    c.execute(sql);
    result=c.fetchall();
    count=0;
    for cc in result:
        count=1
    if(count==1):
        sql1='insert into product(sid,pkey,product,quantity,amount,token) values("%s",
"%s","%s","%s","%s","%s")' % \
            (sender_address,sender_private_key,product,quantity,amount,transaction.sign_transaction())
        print(sql1)
        c.execute(sql1)
        conn.commit()
        conn.close()
    response = {'transaction': transaction.to_dict(), 'signature': transaction.sign_transaction()}
    return jsonify(response), 200

@app.route('/generate/useradding', methods=['POST'])
def useradding():

        random_gen = Crypto.Random.new().read
        private_key = RSA.generate(1024, random_gen)
        public_key = private_key.publickey()
        name = request.form['name']
        mno = request.form['mno']
        address = request.form['address']
        c, conn = connection()
        pri=binascii.hexlify(private_key.exportKey(format='DER')).decode('ascii')
        pub=binascii.hexlify(public_key.exportKey(format='DER')).decode('ascii')
        sql1='insert into register(name, mno,address,master,private) values("%s", "%s","%s","%s","%s")'
% \
          (name,mno,address,pub,pri)
        c.execute(sql1)
        conn.commit()
        conn.close()

        response = {
                'private_key': binascii.hexlify(private_key.exportKey(format='DER')).decode('ascii'),
                'public_key': binascii.hexlify(public_key.exportKey(format='DER')).decode('ascii')
        }

        return jsonify(response), 200
```

```
@app.route('/generate/transaction2', methods=['POST'])
def generate_transaction2():
    sender_address = request.form['sender_address']
    sender_private_key = request.form['sender_private_key']
    recipient_address = sender_address
    value = request.form['amount']
    transaction = Transaction(sender_address, sender_private_key, recipient_address, value)
    c, conn = connection()
    sql='select * from  register where master="%s" and private="%s"'% \
        (sender_address,sender_private_key)
    c.execute(sql);
    result=c.fetchall();
    count=0;
    for cc in result:
        count=1
    if(count==1):
        sql='select amount from  usdtoken where sid="%s" and pkey="%s"'% \
            (sender_address,sender_private_key)
        c.execute(sql);
        result1=c.fetchall();
        count=0;
        amm=0
        for cc1 in result1:
            count=1
            amm=cc1
        if(count==1):
            res = functools.reduce(lambda sub, ele: sub * 10 + ele, amm)
            amount=int(res)+int(value)
            sql1='insert into usdtoken(sid,pkey,amount,token) values("%s", "%s","%s","%s")' % \
                    (sender_address,sender_private_key,amount,transaction.sign_transaction())
            c.execute(sql1)
            conn.commit()
            conn.close()
        else:
            sql1='insert into usdtoken(sid,pkey,amount,token) values("%s", "%s","%s","%s")' % \
                    (sender_address,sender_private_key,value,transaction.sign_transaction())
            c.execute(sql1)
            conn.commit()
            conn.close()
    response = {'transaction': transaction.to_dict(), 'signature': transaction.sign_transaction()}
    return jsonify(response), 200
```

```
if __name__ == '__main__':
    from argparse import ArgumentParser

    parser = ArgumentParser()
    parser.add_argument('-p', '--port', default=8080, type=int, help='port to listen on')
    args = parser.parse_args()
    port = args.port

    app.run(host='127.0.0.1', port=port)
```

dbconnect.py

```
import pymysql
pymysql.install_as_MySQLdb()
import MySQLdb

def connection():
    conn = MySQLdb.connect(host="localhost",
                           user = "root",
                           passwd = "root",
                           db = "privacyecommerce")
    c = conn.cursor()

    return c, conn
```

7. SYSTEM TESTING

7.1 Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done the completion of an individual unit before integration. This is a structural testing, that relies on knowle of its construction and is invasive.

Unit tests perform basic tests at component level and test a specific business process, application, and/o system configuration. Unit tests ensure that each unique path of a business process performs accurately the documented specifications and contains clearly defined inputs and expected results.

7.2 Integration Testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

7.3 Test Cases & Reports / Performance Analysis

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results:

All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

8. CONCLUSION

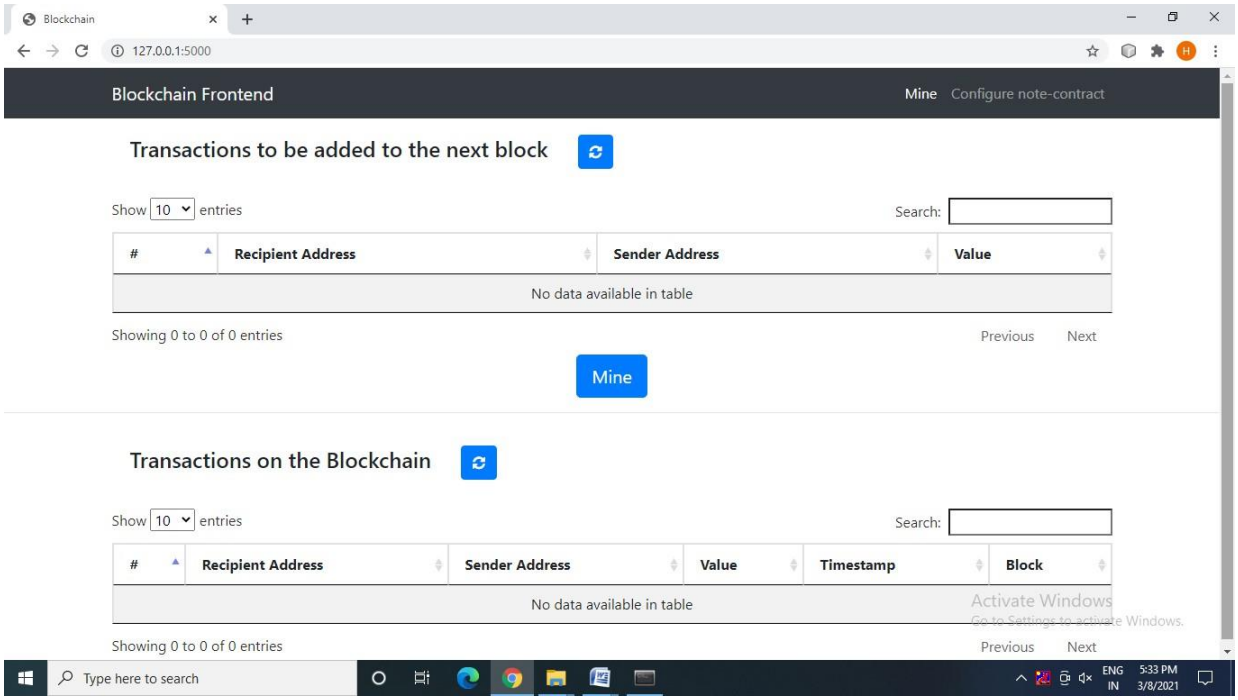
8.1 Conclusion and Future Enhancements

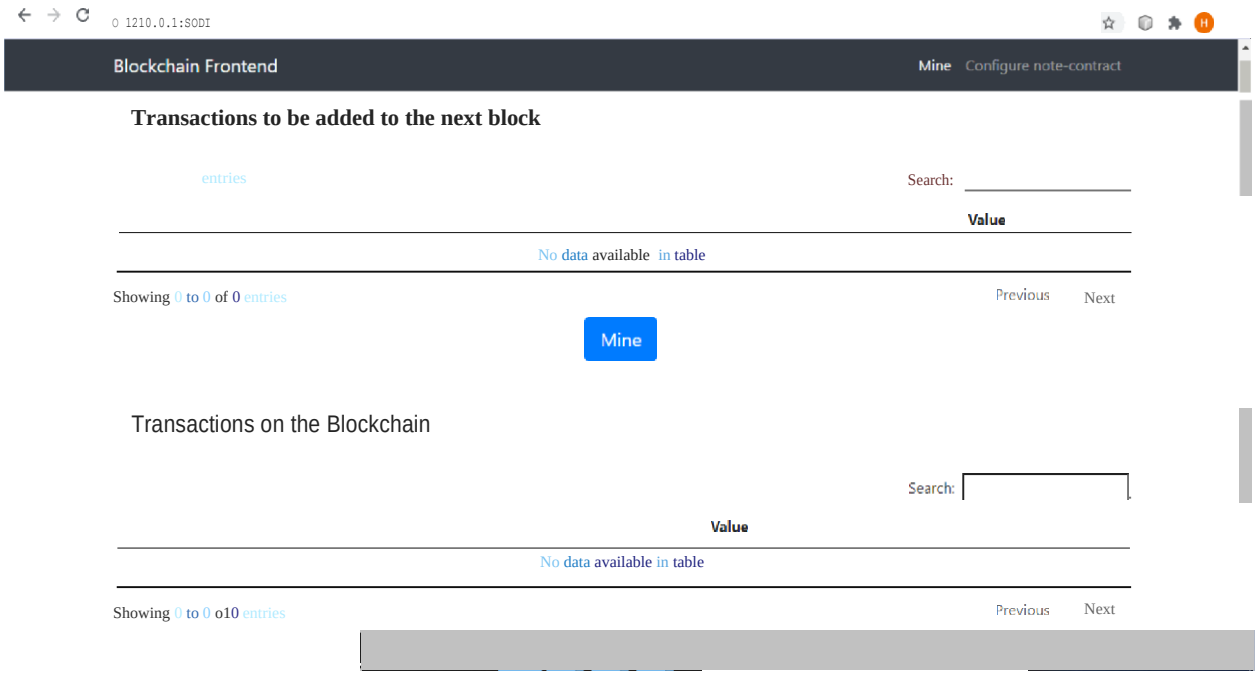
We design a privacy-preserving model for E-commerce systems based on the blockchain technology. In order to protect users' identities and guarantee proof of ownership, we employ a zero-knowledge proof algorithm called zk-SNARKs. The algorithm allows one party (the prover) prove to another party (the verifier) that a given statement is true, without conveying any information apart from the fact that the statement is indeed true by zero knowledge proof. To protect users' addresses, we index the address by a hash string and record it on the logistics ledger. To protect phone numbers, we use delivery boxes and intelligent logistics centers that work with IoT. Different from common practice, buyers do not need to provide a phone number for SMS notification. Finally, We build

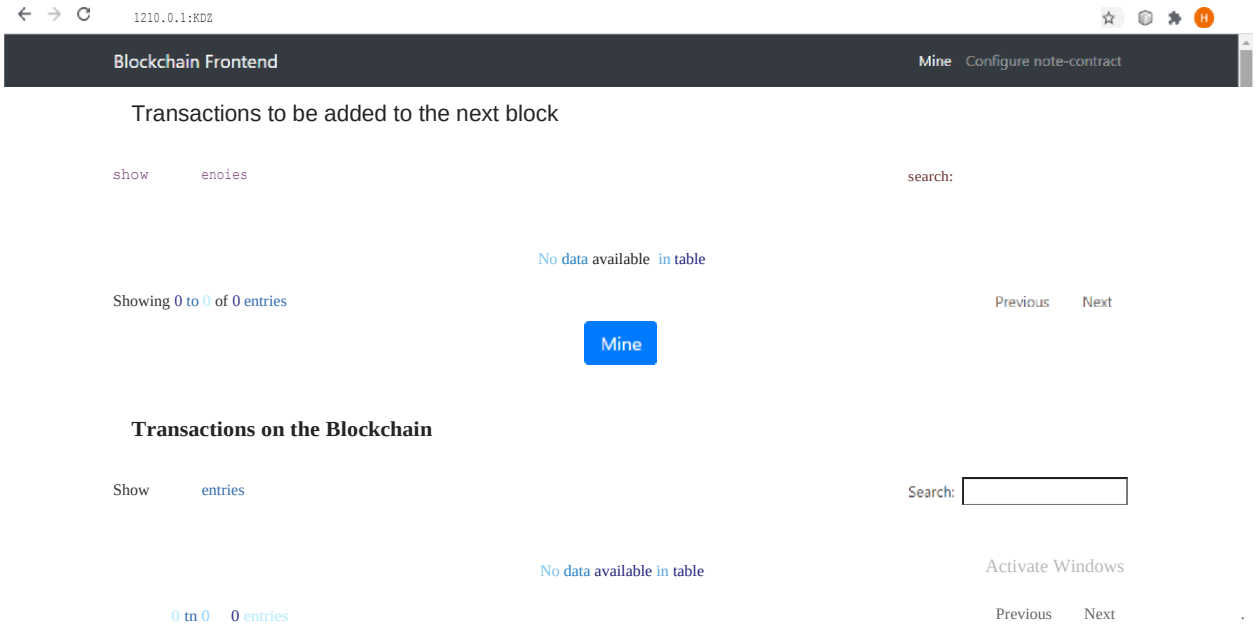
implementations of the architecture using two existing blockchain application platforms. Performance analysis of the blockchain platforms provided insights into the models feasibility and further considerations for deploying a usable implementation. In the future, we will implement our logistics chain on blockchain platforms such as IOTA, which is mainly used in the field of IoT, and perform further simulations on the hardware through the Raspberry Pi.

APPENDICES

A.1 Sample Screens




0 1210.0.1:SODI
Blockchain Frontend
Mine
Configure note-contract
Transactions to be added to the next block
entries
Search:
Value
No data available in table
Showing 0 to 0 of 0 entries
Previous
Next
Mine
Transactions on the Blockchain
Search:
Value
No data available in table
Showing 0 to 0 o10 entries
Previous
Next


1210.0.1:KDZ
Blockchain Frontend
Mine
Configure note-contract
Transactions to be added to the next block
show
enoies
search:
No data available in table
Showing 0 to 0 of 0 entries
Previous
Next
Mine
Transactions on the Blockchain
Show
entries
Search:
No data available in table
Activate Windows
0 tn 0
0 entries
Previous
Next

Transactions to be added to the next block

Shoo[H! jendies

search:

No data available in table

0 tn 0 0 entries

Previous Next

Mine

Transactions on the Blockchain

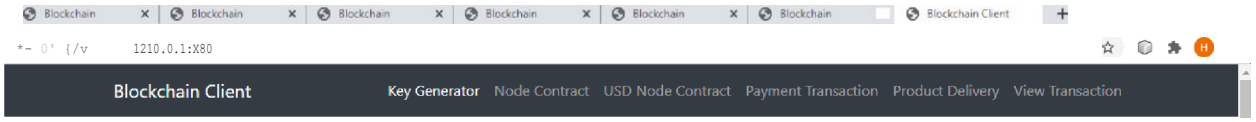
Show entries

No data available in table

Activate Windows

0 tn 0 0 entries

Previous Next .



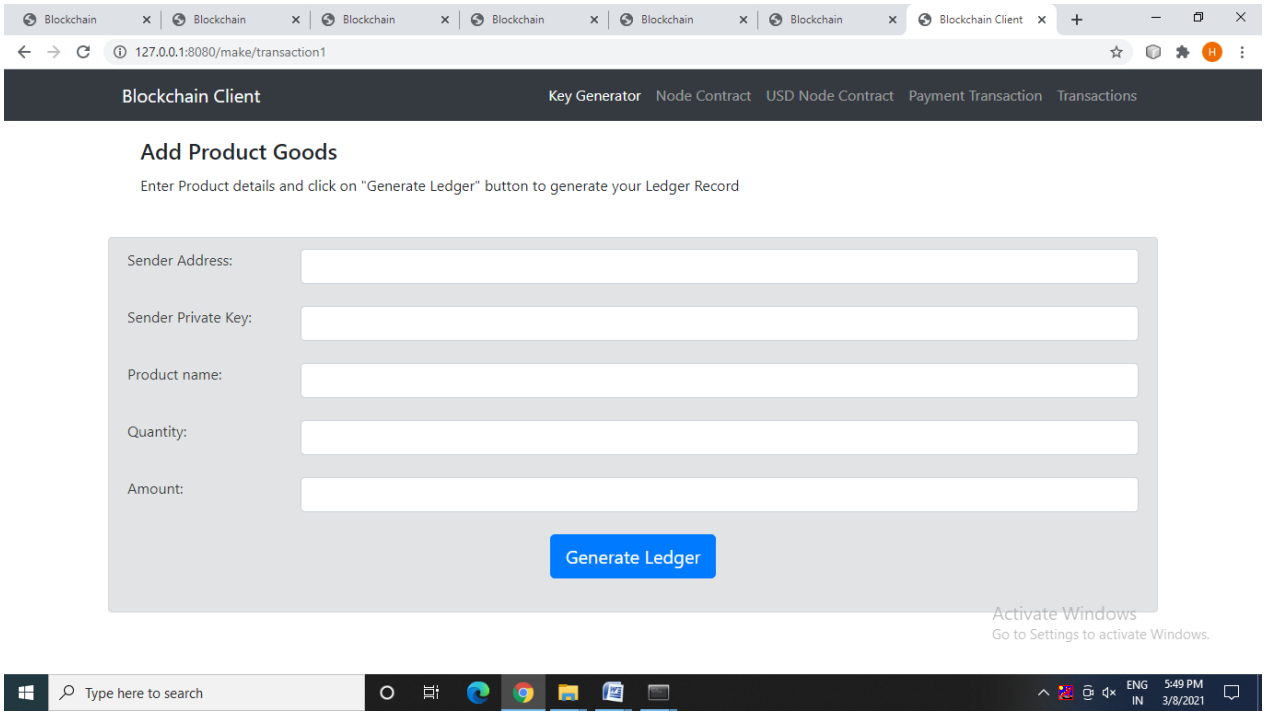
PKI Generator

Enrollment Key:

Name

Mobile No:

Address:



A.2 Publications

[1] M. Niranjanamurthy and D. D. Chahar, "The study of e-commerce security issues and solutions," International Journal of Advanced Research in Computer and Communication Engineering, vol. 2, no. 7, 2013.
[2] J. Y. Tsai, S. Egelman, L. Cranor, and A. Acquisti, "The effect of online privacy information on purchasing behavior: An experimental study," Information Systems Research, vol. 22, no. 2, pp. 254–268, 2011.
[3] B. Berendt, O. G¨unther, and S. Spiekermann, "Privacy in e-commerce: stated preferences vs. actual behavior," Communications of the ACM, vol. 48, no. 4, pp. 101–106, 2005.
[4] L. Xiong and L. Liu, "A reputation-based trust model for peer-to-peer e-commerce communities," in E-Commerce, 2003. CEC 2003. IEEE International Conference on. IEEE, 2003, pp. 275–284.
[5] D. Vandervort, "Challenges and opportunities associated with a bitcoinbased transaction rating system," in International Conference on Financial Cryptography and Data Security. Springer, 2014, pp. 33–42.
[6] https://www.cryptokitties.co/. [9] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014, pp. 459–474.
[7] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan, "Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin," in International Conference on Financial Cryptography and Data Security. Springer, 2017, pp. 321–339.
[8] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal dsa/ecdsa signatures

and an application to bitcoin wallet security," in International Conference on Applied Cryptography and Network Security. Springer, 2016, pp. 156–174.
[9] N. Hackius and M. Petersen, "Blockchain in logistics and supply chain: trick ortreat?" epubli, 2017. [15] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in International Conference on Financial Cryptography and Data Security. Springer, 2013, pp. 6–24.