Building a Blockchain Time-Locked Wallet

Zewei Wu
zwu@oxy.edu
Occidental College

1 Abstract

This research paper is a Occidental College Computer Science Comprehensive Project Proposal regarding my personal project: Blockchain Time-Locked Wallet. The paper contains eight components, including: Introduction, problem context, prior work, technical background, methods, evaluation, results and discussion, and ethical consideration, The paper will begin with an introduction of my project, then I will tackle down some of the technical aspects of the project, and sum up with presentation on some ethical considerations regarding this project.

2 Introduction

For this project, I have chosen to build a Time-Locked digital Wallet on the blockchain network. I have no prior experience with the blockchain programming language.It seems that for most of the big names digital wallets platforms like coinbase, Exodus, they do not possess a time-locked function that allows users to store a certain amount of digital assets for a fixed period of time where they cannot have access to them. So I have decided to build such a smart contract to test if that functionality is feasible and will be useful. But most important goal of this project, is for me to take this as an opportunity, to learn to code in Solidity language, and explore the the functionality of blockchain technology.

3 Problem Context

3.1 The Rise of Cryptocurrency

Cryptocurrency is a digital or virtual currency that uses cryptography for security and is decentralized, meaning it is not controlled by a central authority such as a bank or government. The rise of cryptocurrency can be traced back to the global financial crisis of 2008, which led to the creation of Bitcoin, the first decentralized cryptocurrency. Since then, many other cryptocurrencies have been created, each with their own unique features and uses. The rise of cryptocurrency has sparked a debate about its potential as a viable alternative to traditional forms of currency and has attracted the attention of both investors and regulators. Some people see cryptocurrency as a potential solution to issues such as inflation and high transaction fees, while others are skeptical of its volatility and potential for use in illegal activities.

3.2 Why Blockchain?

Blockchain engineering is a growing field with many job opportunities. A blockchain engineer is responsible for designing, implementing, and maintaining blockchain technology-based solutions for organizations. This can include developing new blockchain-based applications, integrating existing systems with blockchain technology, and ensuring the security and scalability of blockchain networks. The demand for blockchain engineers is expected to continue to grow as more and more businesses and organizations adopt blockchain technology. It is a highly technical field that requires a strong understanding of computer science and software development, as well as knowledge of cryptography and distributed systems. I am interested in pursuing a career in blockchain engineering. So this project would serve as an oppotunity for me to gain experience in these areas and stay up-to-date with the latest developments in the field.

3.3 Why time-locked wallet?

A time-locked wallet is a type of cryptocurrency wallet that prevents the funds in the wallet from being accessed or transferred until a specific time in the future. This type of wallet can be useful for a number of reasons. For example, it can be used to prevent funds from being spent impulsively, to ensure that funds are only available for use at a specific time (such as when a contract is fulfilled), or to create a savings plan by locking away funds for a specific period of time. Time-locked wallets can also be used to add an additional layer of security to a cryptocurrency account, by requiring a waiting period before funds can be accessed or transferred. This can help to protect against unauthorized access to the funds in the event that the wallet's private keys are stolen or lost. By adding this function, it will also be a challenge for me to dive deeper into the technicality aspect of Solidity language.

Blockchain security is important because it ensures the integrity and confidentiality of the information stored on a blockchain network. Blockchain technology is based on the principles of decentralization, immutability, and cryptographic security, which make it a secure and reliable way to store and transfer data. However, like any other technology, blockchain is not immune to security threats. Hackers and other malicious actors may try to exploit vulnerabilities in a blockchain network to gain unauthorized access to sensitive information or disrupt its operation. This is why it is important for blockchain networks to have robust security measures in place to prevent such attacks and protect the information stored on the blockchain. These measures can include using strong encryption algorithms, implementing strict access controls, and regularly auditing the network for vulnerabilities.

4 Technical Background

4.1 What is Blockchain

Before we dive into the topic of smart contract, we need to first have a little background on the topic of blockchain. Blockchain technology is used in cryptocurrencies. A distributed and secure database or ledger is referred to as a blockchain. dApps are the applications that conduct transactions and run the blockchain. Transactions are stored in blocks on the blockchain and subsequently validated by other users. If all of the verifiers agree on a transaction, the block is closed and encrypted, and a new block is created containing information from the preceding block. The information in each subsequent block "chains" the blocks together, giving the blockchain its name. There is no method to edit a blockchain since information in prior blocks cannot be modified without impacting subsequent blocks. The secure nature of a blockchain is provided by this notion, as well as other security mechanisms. The blockchain's novelty is that it ensures the fidelity and security of a data record while also generating trust without the requirement for a trusted third party.[**Bartos**]

4.2 What is Smart Contracts

Smart contracts are essentially programs that run when certain criteria are satisfied and are maintained on a blockchain. They're usually used to automate the execution of an agreement so that all parties can be certain of the conclusion right away, without the need for any intermediaries or time waste. They can also automate a workflow, starting the following step when certain circumstances are satisfied.

The concept of "Smart Contract" was introduced in 1994 by Nick Szabo as "a computerized transaction protocol that executes the terms of a contract".[**SzaboIdeaOf**] To reduce the need for trusted intermediaries between transacting parties and the incidence of malicious or accidental exceptions, Szabo proposed converting contractual provisions into code and embedding them into property that may self-enforce them.[**SzaboSC**]

Smart contracts are scripts stored on the blockchain in the context of blockchain. They're similar to stored procedures in relation to database management systems in terms of functionality. They have a distinct address because they are part of the chain. Smart contract is triggered when sending a transaction. It then operates independently and automatically on every node in the network in a prescribed manner. This means that every node in a smart contract enabled blockchain is running a virtual machine, and the blockchain network acts as a distributed virtual machine.

Smart contracts are self-contained actors whose actions are totally predictable. They may be trusted to advance any on-chain logic that can be represented as a function of on-chain data inputs, as long as the data they need to manage is within their grasp. To fulfill its principal purpose, a smart contract sends an address to another smart contract. The address is stored in the contract's internal database's mutable section. The contract also includes a list of members, whose addresses are used to vote on the company's actions. A rule can be placed in the contract that states that if a majority of the voters vote one way, the contract will change its behavior and call the address that obtained the majority of the votes to perform its main purpose.

Here is a simple example to help you understand the basics of the statements above. Let's say Bobby is trading digital assets Ethereum. Bobby deploys a smart contract defined as "deposit" that allows Bobby to deposit x amount of Ethereum into the contract. Then he deploys another smart contract as "withdraw" that permits him to withdraw all of his digital assets that the contract holds. These "functions" are encrypted in a way that only user "Bobby" can call upon. If Bobby sends an amount of assets to that smart contract's address, using the "deposit" and sends 3 units of ethereum to the contract, this transaction is permanently recorded on the blockchain. If someone else owes Bobby some amount of Ethereum, then the contract checks the signature to make sure the withdrawal is initiated by the contract's owner, and transfers all of its deposits back to Bobby.

Smart contracts actions are totally predictable based on their autonomous actors characteristics . They may be trusted to advance any on-chain logic that can be represented as a function of on-chain data inputs, as long as the data they need to manage is within their grasp. To fulfill its principal purpose, a smart contract sends an address to another smart contract. The address is stored in the contract's internal database's mutable section. The contract also includes a list of members, whose addresses are used to vote on the company's actions. A rule can be placed in the con-

tract that states that if a majority of the voters vote one way, the contract will change its behavior and call the address that obtained the majority of the votes to perform its main purpose.

4.3 Framework

1. Ethereum
2. Truffle Framework
3. Goerli Testnet
4. Metamask

For the project, the smart contract will be built on the Ethereum Blockchain, which is a decentralized, open-source blockchain with smart contract functionality. With the use of Truffle framework, the smart contract will be able to deploy on designated blockchain network. Goerli is a testnet for smart contracting testing, to see if the contract is able to run smoothly before spending any real cryptocurrencies. Metamask is a web app that could access smart contracts as wallets.

4.3.1 Ethereum

Ethereum is a decentralized blockchain technology that creates a peer-to-peer network for securely executing and verifying smart contract code. Participants can transact with one another without relying on a trusted central authority. Participants have full ownership and visibility of transaction data since transaction records are immutable, verifiable, and securely distributed across the network. User-created Ethereum accounts are used to send and receive transactions. As a cost of processing transactions on the network, a sender must sign transactions and spend Ether, Ethereum's native coin.[**Eth**]

Using the native Solidity scripting language and the Ethereum Virtual Machine, Ethereum provides an extraordinarily flexible platform on which to create decentralized apps. Decentralized application developers that use Ethereum to create smart contracts benefit from the robust ecosystem of developer tools and well-established best practices that have accompanied the protocol's maturation. This maturity is reflected in the quality of the user experience provided by Ethereum applications, with wallets such as MetaMask, Argent, Rainbow, and others providing straightforward interfaces for interacting with the Ethereum blockchain and smart contracts placed there. Because of Ethereum's massive user base, developers are more likely to put their applications on the network, cementing Ethereum's position as the principal platform for decentralized applications like DeFi and NFTs. The Ethereum 2.0 protocol, which is currently in development and backwards compatible, will enable a more scalable network on which to construct decentralized applications that require higher transaction throughput in the future.[**Eth**]

4.3.2 Truffle Framework

The Truffle framework is a popular development environment for Ethereum dApp development, with a large community of users. Furthermore, its goal is to make smart contract development more basic and accessible.

4.3.3 Goerli Testnet

Goerli Testnet is a public, proof-of-authority (PoA) test network for Ethereum. It was created to provide a stable and reliable testing environment for developers building on the Ethereum platform. Unlike other Ethereum test networks, which use proof-of-work (PoW) consensus algorithms, the Goerli Testnet uses PoA, which allows transactions to be processed more quickly and efficiently. This makes it an ideal testing environment for developers, who can use the Goerli Testnet to test their Ethereum-based applications without having to worry about the cost and slow processing times associated with PoW networks.

4.3.4 Metamask

MetaMask is a digital wallet that allows users to manage their cryptocurrencies and interact with decentralized applications (dApps) on the Ethereum blockchain. It is a browser extension that can be installed on popular web browsers such as Chrome, Firefox, and Brave. Once installed, MetaMask allows users to create and manage multiple Ethereum accounts, securely store and manage their cryptocurrency assets, and easily interact with dApps on the Ethereum network. MetaMask also allows users to switch between different Ethereum networks, such as the main Ethereum network, test networks, and private networks, making it a versatile and convenient tool for Ethereum users.

5 Prior Work

The concept of blockchain technology can be traced back to the early 1990s, when a group of researchers developed a system for securely timestamped digital documents.[**Bartos**] This work laid the foundation for the development of the first blockchain, which was created by the pseudonymous Satoshi Nakamoto in 2008 as the underlying technology for the digital currency Bitcoin. Since then, many other blockchain-based systems have been developed, each with their own unique features and uses. Some of the key prior work in the field of blockchain technology includes the development of cryptography-based systems for secure communication and digital asset transfer, the creation of decentralized networks for peer-to-peer

exchange, and the creation of smart contract technology for automated and self-executing agreements. These and other innovations have laid the groundwork for the widespread adoption of blockchain technology in a variety of industries.

Recent years, the volume of the DeFi market has been growing rapidly. The value of funds that are stored in DeFi smart contracts has reached 10 billion USD in 2021. The study speculates that the growth of these smart contracts assets reflects on DeFi becoming more relevant in the broader context, which can have acute interests to financial institutions and policy makers.[**Bartos**]

6 Methods

6.1 Overview

The basic steps of creating a time-locked wallet are as follows:

1. Generate a new public/private key pair using a secure cryptographic algorithm, I will be using the truffle framework for initialization.
2. Create a smart contract on the Ethereum blockchain, define the terms and functionality of the smart contract, such as the amount of funds to be locked, the duration of the lock, and any conditions for unlocking the funds.
3. Use the private key to sign and send a transaction to the Ethereum network that deploys the smart contract and transfers the funds to be time-locked to the contract's address.
4. Once the transaction is processed and the smart contract is deployed, the funds will be locked and cannot be accessed or transferred until the specified time has passed and the conditions for unlocking the funds are met.
5. To unlock the funds, send a transaction to the smart contract using the private key, which will trigger the contract to release the funds.

6.2 Initialization

First, I had to install the following dependencies and necessary software and development environment: **Node.js, Git, OpenZeppelin Contracts, Truffle Framework**

Then I initialized a new Truffle project that will create a project directory contains the following files:

- The **contracts** directory is where we will write all our Smart Contracts code.
- The **migrations** directory is where we will write all our deployment scripts, following a specific convention.
- The **test** directory is where we will write all our Smart Contract tests.
- The **truffle-config.js** file is where we will set up network configuration, builds and artifacts directory, and more.

To create a timelock contract, I had to run `- truffle create contract Timelock` command, which will create a **Timelock.sol** solidity file. The contract will consist of multiple functions that we will describe shortly, but all those functions will revolve around these main two main functions:

- **queue()** : This function will be used to broadcast transactions that are going to execute sometimes in the future (after a certain amount of time).
- **execute()** : Once the waiting time of a specific transaction has passed, it can now be called by the execute().

This strategy gives users enough flexibility to make modifications/changes to their transactions before it gets executed (Updating the amount to transfer, change the waiting time, cancel the transaction, etc).

6.3 Function Modifier and Ownable Contracts

While writing the smart contracts, I need to ensure that only addresses with the right authorizations can execute certain functions on our contract. To do so, I will make use of function **modifiers**. Inside the Timelock contract, I created two modifiers that will restrict access to certain functions.

- **onlyOwner()** : this modifier will ensure that only the owner for the smart contract can access a specific resource inside the smart contract.
- **isValidTimestamp(uint256_timestamp)** : this modifier will ensure that the timestamp passed as argument meets the requirements.

6.4 Solidity Mapping

Mapping in Solidity acts like a hashtable or dictionary in any other language. These are used to store the data in the form of key-value pairs. Mappings are mostly used to associate the unique Ethereum address with the associated value type. In this case, I declared 3 different mappings:

// Maps an address to a list of Deposits
mapping(address =>Deposit[]) public deposits;

// Maps a depositId to a Deposit
mapping(bytes32 =>Deposit) public depositIdToDeposit;

//Maps a txId(Queued Tx) to a Deposit (tx id =>queued)
mapping(bytes32 =>Deposit) public queued;

- **Deposits** : this mapping will store all Deposits made by a specific account

- **depositIdToDeposit** : this mapping will store and retrieve a specific Deposit by its **depositId**.
- **Queued** : this mapping will store and retrieve a specific Deposit by its **txId**.

6.5 Solidity Events

An event is an inheritable member of the contract, which stores the arguments passed in the transaction logs when emitted. Generally, events are used to inform the calling application about the current state of the contract. It is always recommended to emit an event every single time the state of the blockchain is mutated.

For my case, I have total of 6 events:

- **DepositedFundsEvent** : which emits an event whenever a deposit occurs.
- **UpdatedDepositEvent** : Emits an event whenever a deposit is updated.
- **QueuedEvent** : Emits an event whenever a transaction is successfully queued for withdrawal.
- **ExecutedTxEvent** : Emits an event whenever a queued transaction is successfully executed.
- **CanceledTxEvent** : Emits an event whenever a queued transaction is successfully canceled.
- **ClaimedDepositEvent** : Emits an event whenever a deposit has been claimed by the receiver.

6.6 Testing Smart Contracts

Testing smart contracts is one of the most important measures for improving smart contract security. In this section, I wrote a test suite in JavaScript for the smart contracts to make sure they are deployed correctly, and I used a TDD (Test Driven Development) approach throughout this project.

- I loaded and interacted with compiled contracts through **artifacts.require()** function.
- Truffle uses the **Mocha** framework, but with the benefits of Truffle's clean room, which means that contracts are deployed before tests are executed.
- Because of the asynchronous nature of the Blockchain, I leveraged the **async/await** syntax of JavaScript.

6.7 Solidity Functions

A function is basically a group of code that can be reused anywhere in the program, which generally saves the excessive use of memory and decreases the runtime of the program. In this case, I coupled functions that will directly or indirectly depend on the Queue() and Execute() functions of the timelock smart contract;

There are many functions in the contract as I will not cover all of them in this section. You are welcome to reference back to the contract files and read the comments I have for each function.

6.8 Deploy the Smart Contract To Goerli Testnet With Infura and Metamask

Using the Goerli network, along with Infura to avoid downloading a local blockchain or an Ethereum client. Infura will allow me to connect to an Ethereum node that it manages. Infura facilitates deployments to the mainnet, Sepolia, and Goerli. In order to test my deployed Smart contracts on these Networks, I needed some FaucetETH. For Goerli, I had to request some test Ether.

With the Goerli configuration complete, I funded my account with some test ether.

```
$ truffle migrate --network goerli --reset
Compiling your contracts...
===========================
Starting migrations... ======================
> Network name: 'goerli'
> Network id: 5
> Block gas limit: 30000000 (0x1c9c380)
1_initial_migration.js ======================
Replacing 'Migrations' ---------------------- >
transaction hash:
0x03698f56403be2d85595293f7092d18c8e54ab00eee33
7dd33b36ff0c819c2ca
> Blocks: 0
> contract address: > block number:
> block timestamp: > account:
> balance:
> gas used:
> gas price:
> value sent:
> total cost:
Seconds: 8 0x934AAbed845B65DCE4d5B3330Dc1719b14
2E44d5 8008403
1669254180 0x800705369a9244e399250574B7f8Fa41F8
1CbcCc 0.093714564994246458
250154 (0x3d12a)
2.500000023 gwei
0 ETH
0.000625385005753542 ETH
   Pausing for 2 confirmations...
------------------------------
> confirmation number: 1 (block: 8008404)
> confirmation number: 2 (block: 8008405)
> Saving migration to chain.
> Saving artifacts
    -------------------------------------
> Total cost: 0.000625385005753542 ETH
2_deploy_contracts.js
=====================
   Deploying 'Timelock'
--------------------
   > transaction hash:
0x9095f7914fca222e2678de096e282d1dd34b0fe3d86a7
7a14b6dd2aa64953ffa
> Blocks: 1
> contract address:
```

```
> block number:
> block timestamp:
> account:
> balance:
> gas used:
> gas price:
> value sent:
> total cost:
Seconds: 16
0x6FFF51144e39084A0e02D4834Fb13EE1B8eab310
8008408
1669254264
0x800705369a9244e399250574B7f8Fa41F81CbcCc
0.083017437349041002
4232938 (0x4096ea)
2.500000034 gwei
0 ETH
0.010582345143919892 ETH
   Pausing for 2 confirmations...
   -------------------------------
   > confirmation number: 1 (block: 8008409)
   > confirmation number: 2 (block: 8008410)
   Deploying 'TimelockFactory'
   ---------------------------
   > transaction hash:
0xa28eba6af921020693f0a0b3c1bdfb48b91f15811b601
b45beb5936c7c752ee7
> Blocks: 1
> contract address:
> block number:
> block timestamp:
> account:
> balance:
> gas used:
> gas price:
> value sent:
> total cost:
Seconds: 8
0x3EedA472f9b6441F2C7C12445975EdCbB00Ef589
8008411
1669254300
0x800705369a9244e399250574B7f8Fa41F81CbcCc
0.269746874615479102
5308225 (0x50ff41)
2.500000044 gwei
0 ETH
0.0132705627335619 ETH
Pausing for 2 confirmations...
-------------------------------
> confirmation number: 1 (block: 8008412)
> confirmation number: 2 (block: 8008413)
> Saving migration to chain.
> Saving artifacts
-------------------------------------
> Total cost:
Summary
=======
> Total deployments:
> Final cost:
0.023852907877481792 ETH
3
  0.024478292883235334 ETH
```

The contracts have successfully migrated to the Goerli public testnet. Two transaction hashes were generated as a result of deploying each contract. Each contract has an address and the account that deployed the contract is identified as

0x800705369a9244e399250574B7f8Fa41F81CbcCc.

To verify that the transaction got published to the **Rinkeby testnet**, I copied the transaction hashes paste onto **https://goerli.etherscan.io/** to see detailed information about the transaction itself.With the transaction hash that was generated when the **Timelock.sol** contract was deployed:

`0x9095f7914fca222e2678de096e282d1dd34b0fe3d86a77a14b6dd2aa64953ffa`

7 Evaluation

First of all, the project should be considered as successful, because the fact that the wallet is able to store digital assets, and can be stored for a period of time, and be able to release the funds after the locked time. After the time-locked place on wallet, users is not be able to not have access to their assets and will retreat their locked assets after the prompted time. After all the fundamentals concerns are eliminated, The security aspect of the wallet will be the most important element of evaluating the project. There are several key considerations for smart contract security, including:

- **Code quality** : The smart contract code should be written in a clear, concise, and well-structured manner. Which helps to prevent errors and vulnerabilities that could be exploited by attackers.
- **Security audits** : The smart contract code should be thoroughly reviewed and tested to identify and address any potential vulnerabilities.
- **Access control** : The smart contract should have strict access controls in place to prevent unauthorized access and modifications.
- **Cryptography** : The smart contract should use strong cryptography algorithms to ensure the confidentiality and integrity of the data stored on the blockchain.

My code is relatively clean and organized, it has strict access control in place to unauthorized access and modification because my choice of using non-public accessible functions. I have also used ownable pattern, which is an address that is set as the "owner" of the contract during the contract-creation process. Protected functions are assigned an OnlyOwner modifier, which ensures the contract authenticates the identity of the calling address before executing the function. Calls to protected functions from other addresses aside from the contract owner always revert, preventing unwanted access.

In terms of cryptography, Solidity supports a number of standard cryptographic algorithms and functions that can be

used to implement various security features in smart contracts. For my project, there was not opportunities for me to implement these algorithms given the level of complexity of my project relates to the field of other blockchain technologies. Using the correct function to prevent non-owner access was the key focus for my project and I think the codes are competent for this purpose.

Although I was not able to run a security test on this smart contract, but I had experts from Southern California Blockchain conference and online forums to go over my codes, and the feedback on the functionality are generally positive, and some of them emphasized that securing the private keys that could locate the address of the wallet is equally important.

However, given the prerequisites of running this wallet, the user interface of the time-locked wallet is not quite beginner friendly. It requires some level of coding experience and browser knowledge to assemble the entire interface. In that case, that would be a slight dedection on the overall grading of this project.

But most importantly, my knowldge of the solidity language and the overall structure of blockchain, is definitely more sophisticated to a degree that allows me to pursue and study the higher level, more complex area of this modern technology.

8 Results and Discussion

In this project, I learned how to set up my project to work like a blockchain project using the truffle init command. This generated a boilerplate for me to use. Next, I wrote two smart contracts and their corresponding tests. I also delved into advanced concepts such as Solidity state variables and functions. I learned how to use a factory contract to deploy new versions of a contract, and explored different ways to deploy it. I tested the smart contract on a local blockchain like Truffle, and finally deployed it to a public testnet called Goerli with the help of Infura and Metamask.

One word to describe my process of building this project will be arduous. Starting from scratch, from knowing nothing about the Solidity language, to successfully deploying a smart contract to the public net, was rewarding.

When learning how to code in the Solidity, prior JavaScript experience will be an advantage to master the language. But the whole process of understanding blockchain itself, should, and is worth to be credited class at universities. But because blockchain technology is a relatively new and rapidly evolving field, so it is difficult to predict exactly what the future will hold for it. However, there are several trends and developments that are likely to shape the future of blockchain.

One of the most significant trends in the blockchain space is the increasing adoption of the technology by businesses and organizations. Many companies are exploring the use of blockchain for a variety of applications, such as supply chain management, identity verification, and asset tracking. As more and more businesses begin to use blockchain, it is likely that the technology will become more mainstream and widely accepted.

Another trend in the blockchain space is the development of new and improved protocols and consensus algorithms. Currently, the most widely used blockchain protocol is the proof-of-work (PoW) algorithm, which is used by the Bitcoin and Ethereum networks. However, there are many other consensus algorithms that are being developed and tested, such as proof-of-stake (PoS) and delegated proof-of-stake (DPoS). These new algorithms are designed to be more efficient and scalable than PoW, and could potentially lead to the development of faster and more decentralized blockchain networks in the future.

In addition to these trends, there are also many exciting developments in the field of decentralized finance (DeFi), which is the use of blockchain technology to create decentralized financial applications and services. DeFi has the potential to revolutionize the financial industry by enabling people to access financial services without the need for intermediaries like banks. The growth of DeFi is likely to continue, and it could lead to the creation of new and innovative financial products and services that are more accessible and transparent than those offered by traditional financial institutions.

Overall, the future of blockchain is likely to be marked by increasing adoption, the development of new protocols and algorithms, and the growth of decentralized finance. These trends will likely lead to the creation of new and exciting opportunities in the blockchain space, and could potentially change the way we think about money and finance.

9 Ethical Considerations

9.1 DeFi

The current state of decentralized finance related research encompasses a wide range of ethical challenges and situations, as well as a diversified user base. With the continued development of Bitcoin technology and its widespread adoption, it is feasible that this technology may become a global financial transaction mechanism. However, there are still a lot of unsolved questions. These decentralized peer-to-peer transactions raise numerous privacy and ethical concerns. Furthermore, the future of some cryptocurrencies has remained dubious.[**FabianDeFi**]

Blockchain's immutability Concerning the Blockchain's immutability, an important ethical point must be posed. From an operational standpoint, blockchain is, of course,

a fantastic breakthrough. To have a limitless number of immutable transaction records, as well as a large amount of data to support each Ledger entry. Amazing and incomprehensible at the moment. The idea behind the "immutable model" is that having a better collection of "real-time data" will result in better forecasting of bitcoin trends. The removal of ambiguity in the Micro could lead to errors in the Macro. Is technology capable of truly bridging the valuation, accounting, and ledger gaps? Alternatively, the grand concept of Blockchain may be destined to be used solely for the distribution of information.

Necessity of governance In DeFi, there is a "decentralization illusion," because the necessity for governance necessitates some level of centralisation, and structural characteristics of the system result in power concentration. DeFi's flaws could jeopardize financial stability if it becomes widely used. Due to excessive leverage, liquidity mismatches, built-in interconnection, and a lack of shock absorbers like banks, these can be severe. Existing governance processes in DeFi would serve as natural reference points for authorities when dealing with issues such as financial stability, investor protection, and illegal activities. Various sorts of intermediation support the cryptocurrency markets. DeFi, in theory, can be used to supplement traditional financial activity. However, it currently has few real-world applications and is primarily used for speculation and arbitrage among several crypto assets. Given its self-contained character, DeFi-driven disruptions in the broader financial system and actual economy appear to be restricted for the time being. Concentration can enable collusion and hinder the viability of blockchains. It raises the possibility that a few major validators will gather enough authority to manipulate the blockchain for financial advantage.

While DeFi is still in its infancy, it provides services that are similar to those given by traditional finance and has some of the same flaws. The fundamental mechanisms that give birth to these vulnerabilities — leverage, liquidity mismatches, and their interaction via profit-seeking and risk-management techniques – are all well-known in the established financial system. However, several aspects of DeFi may make them particularly disruptive. We'll start with the role of leverage and run-risk in stablecoins due to liquidity mismatches in this section, then go on to spillover channels to traditional intermediaries.

Security Aspect The DeFi ecosystem is continuously evolving, despite its tremendous growth. It is now focused mostly for crypto asset speculation, investment, and arbitrage, rather than real-economy use cases. DeFi is vulnerable to criminal activities and market manipulation due to the insufficient application of anti-money laundering and transaction anonymity. Overall, DeFi's core premise – lowering the rents paid to centralized middlemen – does not appear to have been accomplished.[**Fletcher**] History has shown that the early development of novel technologies is typically accompanied by bubbles and a loss of market integrity, despite the fact that it produces inventions that could be useful to a wider audience in the future. DeFi might still play a key role in the financial system with advancements to blockchain scalability, large-scale tokenization of traditional assets, and, most critically, appropriate regulation to maintain protections and boost confidence.

9.2 Environmental Consideration

Expansion of crypto The explosive growth of cryptocurrencies has been astounding. The global cryptocurrency market was worth 793 million dollars in 2019. According to market research conducted by de Vries, it is now predicted to reach over 5.2 billion dollars by 2026. The global use of cryptocurrencies increased by more than 880 percent in just one year, from July 2020 to June 2021. However, environmentalists are concerned about the growing popularity of cryptocurrencies, because the digital "mining" of it produces a significant carbon footprint due to the massive amount of energy required.

Carbon Emission The amount of energy used by blockchain computation is a significant factor to consider. According to a February 2021 CNBC article, the carbon footprint of Bitcoin, the world's largest cryptocurrency, is equivalent to that of New Zealand, based on data from the Bitcoin Energy Consumption Index from Digiconomist, an online tool created by data scientist Alex de Vries. Both emit nearly 37 megatons of carbon dioxide into the atmosphere every year.[**DownToEarth**] This energy-intensive procedure is enabled through "mining," a process in which computer problems are solved in order to authenticate transactions between users, which are subsequently added to the blockchain.[**deVries**] According to Digiconomist, the carbon footprint of a single Ethereum transaction was 102.38 kg of CO2, which is "equivalent to the carbon footprint of 226,910 VISA transactions or 17,063 hours of watching YouTube." Meanwhile, a single Ethereum transaction consumes nearly the same amount of electricity as an ordinary US household consumes in 8.09 days, according to the website.[**Digiconomist2022**]

Combined with the fact that major corporations such as ATT, Home Depot, Microsoft, Starbucks, and Whole Foods have begun to accept bitcoin payments, might pave the way for widespread adoption. However, if the bulls are correct and the price of a single Bitcoin finally reaches 500,000 usd, it will emit more CO2 into the sky than countries like Brazil and Mexico.[**Tully**]

Limited supply Another important feature of most cryptocurrencies is that there is a finite supply. As more cryptocurrencies are mined, the complicated math problems required for transactions get more difficult to solve, requiring more energy.[**Bariviera**] The system is set up in such a way that each digital token created has its own unique cryptographic reference to the blockchain, which ensures its security. The problem of energy use over time is aggravated by mining incentives. When a miner solves the complex hashing process required to create bitcoin, they are rewarded with a little amount of the cryptocurrency.