

ONE WALLET - MULTISIG AND MULTICHAIN WALLET

Enroll. No. (s) - 19103007, 19103072, 19103074

Name of Student (s) - Piyush Mittal, Amulya Kumar, Paras Aghija

Name of Supervisor - Dr. Kapil Madan



MAY 2023

**Submitted in partial fulfillment of the Degree of
Bachelor of Technology
in
Computer Science Engineering**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING AND INFORMATION
TECHNOLOGY**

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

DECLARATION

We, Paras Aghija (19103074), Amulya Kumar (19103072) and Piyush Mittal (19103007) , hereby declare that the Major Project Report titled **One Wallet - Multisig and Multichain Wallet**, submitted towards the fulfillment of the requirements for the undergraduate B.Tech (CSE) Degree, at Jaypee Institute of Information Technology, is my original work.

We affirm that the project was completed under the guidance of Dr. Kapil Madan, and This report's contents have not been submitted in whole or in part for the conferral of any other degree or certificate.

All the sources of information used in this project have been duly acknowledged and referenced in the bibliography section of this report. The data and research findings presented in this report are authentic and have been verified by us through rigorous analysis and evaluation.

Date: 01/05/2023

Place: Jaypee Institute of Information Technology, Noida

Group Members

Name	Enrollment No	Signature
Piyush Mittal	19103007	<i>Piyush Mittal</i>
Amulya Kumar	19103072	<i>Amulya Kumar</i>
Paras Aghija	19103074	<i>Paras Aghija</i>

CERTIFICATE

This is to certify that Paras Aghija (19103074), Amulya Kumar (19103072), Piyush Mittal (19103007), students of BTech (CSE) at Jaypee Institute of Information Technology, have successfully completed their Major Project titled **One Wallet - Multisig and Multichain Wallet** under the guidance of Dr. Kapil Madan.

After being evaluated, the department acknowledged the project as fulfilling the criteria for granting the BTech (CSE) degree.

The project report submitted by the student has been found to be satisfactory and meets the standards set by the department for the completion of the degree.

Date: May 1, 2023

Place: Jaypee Institute of Information Technology

Signature of Supervisor -

Name of Supervisor - Dr. Kapil Madan

Date - 01/05/2023

ACKNOWLEDGEMENT

The team members of Group 83 would like to express their sincere appreciation to everyone who helped our Major Project be completed successfully.

First and foremost, we would like to thank Dr. Kapil Madan, our project guide, for their invaluable guidance and support throughout the project. Their insightful feedback, constructive criticism, and unwavering encouragement have been instrumental in shaping the project and bringing it to fruition.

We would also like to thank the faculty members of the Computer Science Department for their support and guidance throughout the course of our degree program. Their expertise and knowledge have been invaluable in shaping our ideas and approaches to the project.

We extend our heartfelt gratitude to our families and friends, for their constant support, encouragement, and motivation throughout the project. Their unwavering support has been a source of strength and inspiration for us.

Lastly, we would like to thank all the participants who volunteered to be a part of our study. Without their cooperation and participation, the project would not have been possible. We are grateful to all those who have contributed to this project, directly or indirectly. Thank you all for your support and encouragement.

Group Members

Name	Enrollment No	Signature
Piyush Mittal	19103007	<i>Piyush Mittal</i>
Amulya Kumar	19103072	<i>Amulya Kumar</i>
Paras Aghija	19103074	<i>Paras Aghija</i>

SUMMARY

A multisignature (multisig) wallet is a type of digital wallet that requires multiple parties to authorize a transaction before it can be executed. This helps prevent fraudulent or unauthorized transactions and ensures that the funds can only be accessed by the authorized parties. Multisig wallets can be implemented using smart contracts on various blockchain platforms, such as Ethereum, Bitcoin, or other compatible networks.

Multisig wallets provide enhanced security and management capabilities, making them a valuable tool for securing digital assets and enabling complex financial transactions. This project is an example of a multisig wallet implementation on the Ethereum Virtual Machine (EVM), using the Scaffold-ETH development framework. The project provides a multisig wallet smart contract, client libraries in multiple programming languages, and documentation to help developers interact with the multisig wallet and deploy it on the Ethereum blockchain.

The term "multichain" in the context of multisig wallet implementations refers to the ability to deploy the multisig wallet on different blockchain networks, not limited to just one specific network. This can allow users to access the same multisig wallet across different blockchain platforms, increasing its versatility and potential usability. However, it is important to note that multisig wallet implementations may not always be compatible with every blockchain network and may require specific adjustments or adaptations to work effectively.

Signature of Supervisor

Group Members




Name	Enrollment No	Signature
Piyush Mittal	19103007	
Amulya Kumar	19103072	
Paras Aghija	19103074	

TABLE OF CONTENTS

	Page No.
<i>Declaration</i>	ii
<i>Certificate</i>	iii
<i>Acknowledgement</i>	iv
<i>Summary</i>	v
<i>List of Figures</i>	viii
<i>List of Tables</i>	ix
Chapter - 1 Introduction	1-3
1.1 General Introduction	
1.2 Problem Statement	
1.3 Significance/Novelty of the problem	
1.4 Field Study	
1.5 Brief Description of the Solution Approach	
1.6 Comparison of existing approaches to the problem faced	
Chapter - 2 Literature Survey	4-10
2.1 Summary of papers studied	
2.2 Integrated summary of the literature studied //new	
Chapter - 3 Requirement Analysis and Solution Approach	11-16
3.1 Overall description of the project	
3.2 Requirement Analysis	
3.3 Solution Approach //new	

Chapter - 4	Modeling and Implementation Details	17-35
	4.1 Design Diagrams	
	4.1.1 Use case diagrams	
	4.1.2 Class diagrams/Control flow diagrams	
	4.1.3 Sequence Diagram/Activity diagrams //new	
	4.2 Implementation details and issues	
	4.3 Risk Analysis and Mitigation	
Chapter - 5	Testing	36-41
	5.1 Testing tool	
	5.2 Component decomposition and type of testing required	
	5.3. List all test cases in prescribed format	
	5.4 Error and Exception Handling	
	5.5 Limitations of the solution	
Chapter - 6	Findings, Conclusions and Future Work	42-43
	6.1 Findings	
	6.2 Conclusions	
	6.3 Future Work	
	References	44-45

LIST OF FIGURES

Figures Name	Page No
Figure 1. Modeling and Implementation of MultiSig Wallet	17
Figure 2. Transaction of MultiSignature Wallet	18
Figure 3. Use Case Diagram for Multi Signature Wallet	19
Figure 4: Sequence Diagram of Multi Signature Wallet	20
Figure 5: Flow Chart of the Multi Signature Wallet	21

LIST OF TABLES

S.No	Name of Tables	Page No
1.	Research Paper entitled “Smart contract development: Challenges and opportunities.”	4
2.	Research Paper entitled “Blockchains Through Ontologies”	5
3.	Research Paper entitled “Ethereum: state of knowledge and research perspectives”	6
4.	Research Paper entitled “Security Analysis of Multi-signature Bitcoin Wallets”	7
5.	Research Paper entitled “A Review of Smart Contract Security Vulnerabilities and Their Mitigation Techniques”	8
6.	Research Paper entitled “Smart Contracts Security: A Systematic Mapping Study”	9
7.	Research Paper entitled “Smart Contract-Based Access Control for Digital Assets”	10

Chapter 1. Introduction

1.1 General Introduction

The use of cryptocurrencies has grown significantly in popularity and adoption during the past several years. As a result, a number of blockchain-based platforms and programmes, including smart contracts, have been created. In smart contracts, the terms of the agreement between the buyer and seller are directly written into lines of code. These contracts self-execute. Without using middlemen, these contracts may automate a number of operations, including payments. The security of digital wallets is one of the major issues facing the bitcoin ecosystem. Traditional wallets rely on private keys, which can cause financial loss if they are misplaced or stolen. However, the creation of multisig wallets offers a solution to this issue. Multiple signatures are needed to approve transactions in multisig wallets, also known as multi-signature wallets. They are much safer than conventional wallets as a result.

1.2 Problem Statement

The safety of cryptocurrency digital wallets has been a major source of consumer anxiety. Private key-based wallets may be susceptible to theft, loss, or hacking, which might result in the loss of money. To address this issue, multisig wallets have been created, however they do have certain drawbacks. Multisig wallets can be more difficult to deploy than regular wallets, requiring a high level of technical knowledge, which may deter some users from using them. Furthermore, there are flaws in both centralised and decentralised multisig wallets that could cost you money. These issues call for the development of a user-friendly and safe cryptocurrency asset management system. The solution should offer a streamlined user experience, be resistant to threats, and protect the security of users' assets. In order to provide a safe and convenient method of handling bitcoin assets, this project tries to identify the main issues with current multisig wallets and provide a solution that resolves them.

1.3 Significance/Novelty of the Problem

The use and expansion of cryptocurrencies depend heavily on the security of digital wallets. The demand for secure wallet solutions is rising as the use of cryptocurrency becomes more widespread. Although multisig wallets have been created to address this issue, a more user-friendly and secure solution is still required. The importance of this project rests in the creation of a multisig wallet powered by smart contracts that solves the problems with current multisig wallets. The suggested solution will contribute

significantly to the bitcoin ecosystem by offering improved security features, a streamlined user experience, and resilience to assaults. The wallet will be safe, open, and decentralised because to the solution's smart contract-based architecture, giving consumers a high degree of confidence in how their bitcoin holdings are managed. The possible effects of the suggested approach go beyond the bitcoin ecosystem, though. The creation of a safe and convenient multisig wallet might open the way for blockchain technology to be adopted in other sectors like banking and logistics, where safe digital asset management is crucial. Overall, the importance of this issue lies in its potential to offer a more user-friendly and secure solution for managing cryptocurrency assets as well as in its potential to encourage the adoption of blockchain technology in other industries.

1.4 Brief Description of the Solution Approach

A smart contract-based multisig wallet is the suggested remedy, and it makes use of blockchain technology to offer improved security, openness, and decentralised governance. A self-executing digital contract known as a "smart contract" can automate the process of managing transactions between parties. The multisig wallet is more secure than conventional wallets since it needs several signatures from authorised parties in order to approve a transaction. Due to the fact that the smart contract is executed on a decentralised network of computers rather than a centralised server, it guarantees that the wallet is open, secure, and resistant to attacks. The problem-solving strategy entails a detailed examination of the difficulties faced by current multisig wallets, followed by the creation and evaluation of a prototype multisig wallet based on smart contracts. The prototype will be created to solve the problems noted and guarantee that the wallet is simple to use, safe from threats, and attack-resistant. To make sure the wallet is dependable and adheres to the highest standards of security and usability, the testing phase will comprise a number of demanding security and usability tests. In order to further hone and improve the solution, the prototype will then be placed on a blockchain network and its performance and user feedback will be assessed. Overall, the solution method entails a thorough examination of the problems with the multisig wallets that are now in use, followed by the creation of a unique multisig wallet based on smart contracts that solves these problems and adds improved security and usability features.

1.5 Comparison of existing approaches to the problem faced

To solve the security issues posed by cryptocurrency digital wallets, a number of existing strategies have been created. The most popular wallets are those based on private keys, but they can be lost, stolen, or hacked, which might result in the loss of money. To address this issue, multisig wallets were created, although they have limitations. Users of centralised multisig wallets must put their faith in a central

authority, which goes against the decentralised and open nature of cryptocurrencies. Decentralised multisig wallets, on the other hand, provide greater decentralisation and transparency but can be difficult to create and need advanced technical knowledge, which may prevent some users from using them. Other methods, like hardware wallets, offer better security features but can be costly and difficult for some users to use. Although mobile wallets are simple to use, they can be susceptible to malware or hacking attacks, which could result in the loss of money.

Chapter 2. Literature Survey

2.1 Summary of the papers studied

Table 1: Research Paper entitled “Smart contract development: Challenges and opportunities.”

Title	Smart contract development: Challenges and opportunities ^[1]
Author	Weiqin ZOU David LO Pavneet Singh KOCHHAR Xuan-Bach D. LE
Year	2021
Publication Detail	Research Collection School Of Computing and Information Systems
Summary	<p>The introduction of blockchain technology has generated a lot of interest in smart contracts, which automate legal arrangements. However, the phrase "smart contract" is now frequently used to describe low-level blockchain scripts. This research focuses on the difficulties that Ethereum platform developers must overcome while building these smart contracts. The researchers conducted surveys and interviews, and the results showed that the development of smart contracts is still in its early phases. Smart contract code security is presently not generally acknowledged, and there are few online learning tools and community resources available. The researchers made specific recommendations for future study and practise to promote the creation of smart contracts in light of their results. Making strides in these ways will speed up the creation of smart contracts.</p>
DOI	10.1109/tse.2019.2942301

Table 2: Research Paper entitled “Blockchains Through Ontologies”

Title	Blockchains Through Ontologies ^[2]
Author	Giampaolo Bella, Domenico Cantone, Cristiano Longo Marianna Nicolosi Asmundo
Year	2022
Publication Detail	Studies in Computational Intelligence
Summary	The Ethereum blockchain and the smart contracts that have been implemented on it are represented in this article using the OASIS ontology, with a focus on those that follow the ERC721 standard for NFT management. In order to search the blockchain for smart contracts and related NFTs based on needed properties, the researchers used OASIS to describe Ethereum semantically. The OASIS ontological model makes it possible to distinguish between smart contracts and the NFTs that go with them.
DOI	10.48550/arXiv.2109.02899

Table 3: Research Paper entitled “Ethereum: state of knowledge and research perspectives”

Title	Ethereum: state of knowledge and research perspectives ^[3]
Author	Sergei Tikhomirov
Year	2018
Publication Detail	International Symposium on Foundations and Practice of Security.
Summary	<p>With an emphasis on Ethereum specifically, this paper provides an overview of the current state of knowledge in the area of smart contracts. The authors give a technical overview of Ethereum, talk about current problems, and go at potential remedies. They discuss other blockchains that allow smart contracts as well. Smart contracts, which are Turing complete programmes run on a decentralised network that manipulate digital units of value, are often employed on the Ethereum platform. A peer-to-peer network of nodes that run programmes upon request maintains the overall state. Similar to Bitcoin, the state is kept in a blockchain that is protected by a proof-of-work consensus process. The main benefit of Ethereum is its full-featured programming language, which can be used to create sophisticated business logic. In industries like crowdfunding, financial services, identity management, and gaming, decentralised apps without a trusted third party are particularly appealing. Cryptography, consensus methods, programming languages, governance, finance, and law are just a few of the many topics covered by the complex academic issue of smart contracts.</p>
DOI	10.1007/978-3-319-75650-9_14

Table 4: Research Paper entitled “An efficient multi-signature wallet in blockchain using bloom filter”

Title	An efficient multi-signature wallet in blockchain using bloom filter ^[4]
Author	Jongbeen Han, Mansub Song, Hyeonsang Eam, Yongseok Son
Year	2021
Publication Detail	36th Annual ACM Symposium on Applied Computing
Summary	<p>A digital signature is used by the blockchain wallet, a tool for managing digital assets, to sign transactions. However, using a single-signature scheme has security flaws, and using a multi-signature scheme has performance problems. An innovative and effective multi-signature wallet that improves performance, storage efficiency, and privacy without changing the blockchain protocol has been offered as a solution to these problems. For high validation performance, this new wallet uses the threshold elliptic curve digital signature technique (T-ECDSA) to handle multi-signature as a single-signature. The wallet utilises a bloom filter for transactions to identify participants without disclosing their identities and minimise transaction size in order to guarantee maximum storage efficiency and privacy. According to test findings, the suggested multi-signature wallet performs better at verification and has smaller transaction sizes than other wallets.</p>
DOI	10.1145/3412841.3441910

Table 5: Research Paper entitled “Smart Contract Security: A Software Lifecycle Perspective”

Title	Smart Contract Security: A Software Lifecycle Perspective ^[5]
Author	Y. Huang, Y. Bian, R. Li, J. L. Zhao, P. Shi
Year	2019
Publication Detail	IEEE Access Volume 7
Summary	<p>Assuring the security of smart contracts that are carried out in a blockchain system is the subject of the study field known as "smart contract security." These automatically running executable scripts work on the blockchain to uphold a contract between parties to a transaction. Unfortunately, a number of security problems have been discovered, resulting in large monetary losses. Given that the execution environment is based on decentralised blockchain computing, this presents new challenges for researchers. There have been several half-measures suggested, but a thorough investigation of smart contract security is required. In this context, researchers have reviewed the available research on smart contract security from the standpoint of the software lifecycle. They have listed typical security flaws and highlighted the essential blockchain elements that might compromise security. We've looked at recent developments in security for smart contracts throughout four phases of development, including security design, implementation, testing, and monitoring. To strengthen their security and lower risks, growing possibilities and difficulties in smart contract security have been identified.</p>
DOI	10.1109/ACCESS.2019.2946988

Table 6: Research Paper entitled “Wallet Contracts on Ethereum”

Title	Wallet Contracts on Ethereum ^[6]
Author	M. Salamah, N. Modi, L. Xu, and M. M. Hassan
Year	2020
Publication Detail	2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)
Summary	<p>While cryptographic tokens are all-purpose tools for managing assets and rights in the blockchain ecosystem, cryptocurrencies operate like money. To communicate with smart contracts and the blockchain network, software wallets are utilised. Some wallets are designed as smart contracts that enable extra features like daily restrictions, multiple signatures, and approvals to boost transparency and security. Ethereum is a well-liked option for wallet contracts since it is a well-known platform for tokens and smart contracts. The article goes through many techniques for identifying wallet contracts by looking at their bytecodes and interaction styles. Additionally, it describes the six different types of wallets and their distinctive features.</p>
DOI	10.1109/ICBC48266.2020.9169467

Table 7: Research Paper entitled “The Advance of Cryptocurrency Wallet with Digital Signature”

Title	The Advance of Cryptocurrency Wallet with Digital Signature ^[7]
Author	Pu Ji
Year	2023
Publication Detail	Cryptology ePrint Archive, Paper
Summary	<p>This study analyses the bitcoin wallet by looking at its elements, capabilities, classifications, and security aspects. The introduction of the private key, public key, and address—the three main parts of a cryptocurrency wallet—focuses on how they are generated using the secp256k1 Elliptic Curve Digital Signature Algorithm and the Keccak hash function. Multiple crypto wallets are described along with classifications of the various types of crypto wallets based on their use, display format, and internet connectivity. In Ethereum, the secp256k1 elliptic curve is utilised for both the generation of public keys and transaction signing. This article explores the secp256k1 and Keystore utilised in the signing process as well as the safety of the cryptocurrency wallet, underlining possible dangers brought on by the growth of processing power and the transfer of Ethereum's validation mechanism.</p>
DOI	10.54097/hset.v39i.6714

Chapter 3. Requirement Analysis and Solution Approach

3.1 Overall description of Project

A multisig smart contract wallet called One Wallet enables many owners to securely and openly manage and transfer assets like Ethereum (ETH). One Wallet users may easily add or remove owners, deploy the wallet on any chain that is EVM-compatible, and modify transactions by manually inserting calldata. The project was developed using the most modern web technologies, such as Scaffold-eth 2, Next.js^[13], WAGMI, Ether.js, and Rainbowkit. The One Wallet is exceptionally scalable, rapid, and trustworthy because to its technological stack.

The ultimate objective of the One Wallet project is to simplify the management of digital assets while providing users with a straightforward, secure, and customizable multisig wallet solution. It gives consumers complete control over their digital assets thanks to the greater protection of numerous owners.

3.1.1 Product Perspective

From a product standpoint, One Wallet is an original and cutting-edge method of controlling digital assets. In addition to offering a great degree of customisation through human calldata entry, it enables a safe and transparent means for many owners to maintain and transfer wealth, similar to Ethereum.

The ability for numerous owners to govern and administer the wallet is one of One Wallet's primary advantages. The fact that no owner may make decisions or perform transfers without the consent of the other owners adds an extra degree of security and responsibility. One Wallet also allows users to manually enter calldata to carry out specific transactions that are normally not feasible with other wallets, making it more flexible.

3.1.2 Product Functions

The One Wallet project offers a variety of features to assist users in securely and openly managing and transferring digital assets. Some of the key functions of the product include:

- **Multisig Functionality:** Multiple owners can control and manage the wallet using One Wallet. No owner may decide or make a transfer without the consent of the other owners, adding an extra degree of security and responsibility.

- **Add/Remove Owners:** The wallet's users may quickly add or remove owners as needed. This makes it simple to govern who has access to the wallet and guarantees that only people who are supposed to have it may.
- **Transfer Value:** Users may securely and openly move money across wallets using One Wallet, including Ethereum. Any of the authorised owners may carry out this job, providing that transfers are only conducted with the consent of the other owners.
- **Custom Transactions:** Users of One Wallet may manually enter calldata to carry out customised transactions that are normally impractical to carry out using other wallets. Users may customise and have a lot of freedom with this function.
- **Chain Compatibility:** One Wallet is a flexible solution for a variety of use cases and scenarios since it can be implemented on any chain that is EVM-compatible.

Overall, One Wallet's features give customers the ability to safely and transparently handle and transfer digital assets while also having a lot of customization and flexibility options. It is a distinctive and potent option for both people and corporations because to the multisig functionality and flexibility of transactions.

3.1.3 User Characteristics

The One Wallet project is made to satisfy the requirements of a wide variety of users who are interested in securely and individually managing their digital assets. The primary user characteristics for the One Wallet project include:

- **Individual Users:** One Wallet offers a safe and transparent way to manage and transfer your assets, thus individual users who own digital assets like Ethereum may find it handy. One Wallet provides a high level of security and accountability thanks to its multisig capabilities. This feature can be very helpful for people who are worried about the protection of their digital assets.
- **Business Users:** One Wallet may be helpful for companies that manage digital assets, such as bitcoin exchanges or investment organisations. It's the perfect choice for companies who need a high level of security and flexibility in managing their digital assets because of the multisig capabilities and ability to customise transactions.

- **Developers:** One Wallet could also be helpful for developers who want to create decentralised apps (dApps) or smart contracts. The wallet is a flexible option for developers who want a dependable and adaptable way to handle digital assets since it can be deployed on any EVM-compatible chain and customised transactions by manual calldata input.

Overall, the One Wallet project is made to satisfy the requirements of a wide spectrum of users who are interested in securely and individually managing their digital assets. One Wallet is a strong and cutting-edge solution for managing digital assets with trust and convenience, regardless of whether you are an individual user, corporate user, or developer.

3.1.4 Constraints

The One Wallet project, like any software project, is subject to several limitations that may affect how it functions or how it is developed. Some of the key constraints of the One Wallet project include:

- **Security:** The necessity for high levels of security to safeguard user cash and prevent unauthorised access is one of the project's main limitations. To make sure they are working effectively and safeguarding user cash, the wallet's multisig capabilities and other security features must be thoroughly tested and audited.
- **Compatibility:** The requirement for interoperability with a broad range of EVM-compatible chains is another restriction placed on the One Wallet initiative. In order to guarantee that the wallet can operate effectively on every chain it is deployed on, thorough testing and development are required.
- **User Experience:** The development of a user-friendly interface that is simple for people to use and comprehend is one of the project's main difficulties. The development team must concentrate on making an easy-to-use interface without sacrificing security or functionality.
- **Smart Contract Limitations:** The limitations of the underlying smart contract technology also place a cap on the One Wallet project's capability. To make sure that the smart contract can carry out the specified functions and that any restrictions or limits are appropriately addressed, rigorous planning and development are necessary.

Overall, there are a lot of restrictions for the One Wallet project in terms of security, interoperability, user experience, and smart contract capabilities. But these limitations can be removed with careful

planning, development, and testing to produce a potent and cutting-edge method of managing digital assets.

3.1.5 Assumptions and dependencies

Assumptions and dependencies are important factors that can impact the development and implementation of the One Wallet project. Some of the key assumptions and dependencies for the One Wallet project include:

- **Blockchain Adoption:** The ongoing acceptance and development of blockchain technology is one of the foundational tenets of the One Wallet initiative. The continued usage and use of blockchain technology is essential to the project's success, but these developments are dependent on a variety of economic and technological variables that are outside the control of the development team.
- **Third-Party Integration:** The One Wallet project's capacity to interact with external tools and services, such as Ethereum wallets or blockchain explorers, is also essential to its success. To guarantee that the wallet is compatible with a wide range of tools and services, rigorous design and development are required.
- **User Adoption:** The acceptance and usage of the wallet by both individual and commercial users is another important presumption of the One Wallet concept. The project's success depends on its capacity to draw in and hold onto a user base, which is influenced by a variety of elements such as user experience, security, and market circumstances.
- **Regulatory Compliance:** The One Wallet project's success also depends on adhering to legal regulations and norms, which may change according on the location. To guarantee that the wallet complies with all applicable regulatory requirements and standards, thorough planning and development are necessary.

Overall, the success of the One Wallet project depends on a range of assumptions and dependencies related to blockchain adoption, third-party integration, user adoption, and regulatory compliance. By carefully considering and addressing these factors, the development team can maximize the potential for success and create a valuable solution for managing digital assets.

3.2 Requirement Analysis

3.2.1 Functional requirements

- **Multisig Feature:** Multiple authorised signatures must be required by the wallet before a transaction can be approved, increasing security and lowering the possibility of theft or money loss.
- **User-friendly Interface:** Even non-technical users should be able to use and comprehend the wallet's user-friendly interface.
- **Transaction Management:** The wallet must make it simple for users to handle their transactions, such as sending and receiving bitcoin and keeping track of past transactions.
- **Security Measures:** Advanced security methods like encryption and two-factor authentication must be used by the wallet to prevent unauthorised access and guarantee the protection of user cash.
- **Transparency:** Users must be able to see the wallet's transactions and activities, and the smart contract must make sure that the wallet's operations are visible and auditable.
- **Compatibility:** The wallet has to work with a variety of cryptocurrencies to allow users to manage their various portfolios in one place.
- **Network Compatibility:** For consumers to manage their assets across numerous blockchain ecosystems, the wallet must be interoperable with a variety of blockchain networks.
- **Testing and Auditing:** To guarantee that the wallet satisfies the greatest requirements for security, usability, and functionality, it must go through a thorough testing and auditing process.

3.2.2 Non Functional requirements

- **Performance:** Even when the network is busy, the wallet must be able to handle transactions swiftly and effectively.
- **Scalability:** To be able to scale to meet the needs of a growing user base, the wallet must be built to handle a high volume of transactions.
- **Reliability:** To guarantee that users may access their money at all times, the wallet must be dependable with a low chance of downtime or failure.
- **Security:** To prevent unauthorised access or the theft of user cash, the wallet has to offer a high degree of security with reliable encryption and authentication measures.
- **Usability:** To guarantee that even non-technical users can handle their bitcoin holdings with ease, the wallet must be user-friendly with a clear and straightforward UI.

- **Compatibility:** The wallet must work with a variety of hardware and software, including desktop computers, mobile phones, and web browsers, to enable users to access their money at any time and from any location.
- **Regulatory Compliance:** Users must feel confident using the wallet since it complies with all applicable rules and legislation regulating the administration of digital assets.
- **Maintenance and Support:** To guarantee that the wallet is safe, useful, and current, a trustworthy team of developers must provide frequent upgrades and maintenance

Chapter 4. Modeling and Implementation Details

Cold wallets and hot wallets are used in conjunction by centralised exchanges to safeguard customer funds. While hot wallets store a smaller amount of cryptocurrency needed for daily operations and are connected to the internet for real-time trades, cold wallets store the majority of the exchange's cryptocurrency holdings offline in an extremely secure environment. Users should still take security safeguards to safeguard their own money even though this helps to reduce the danger of theft or security breaches.

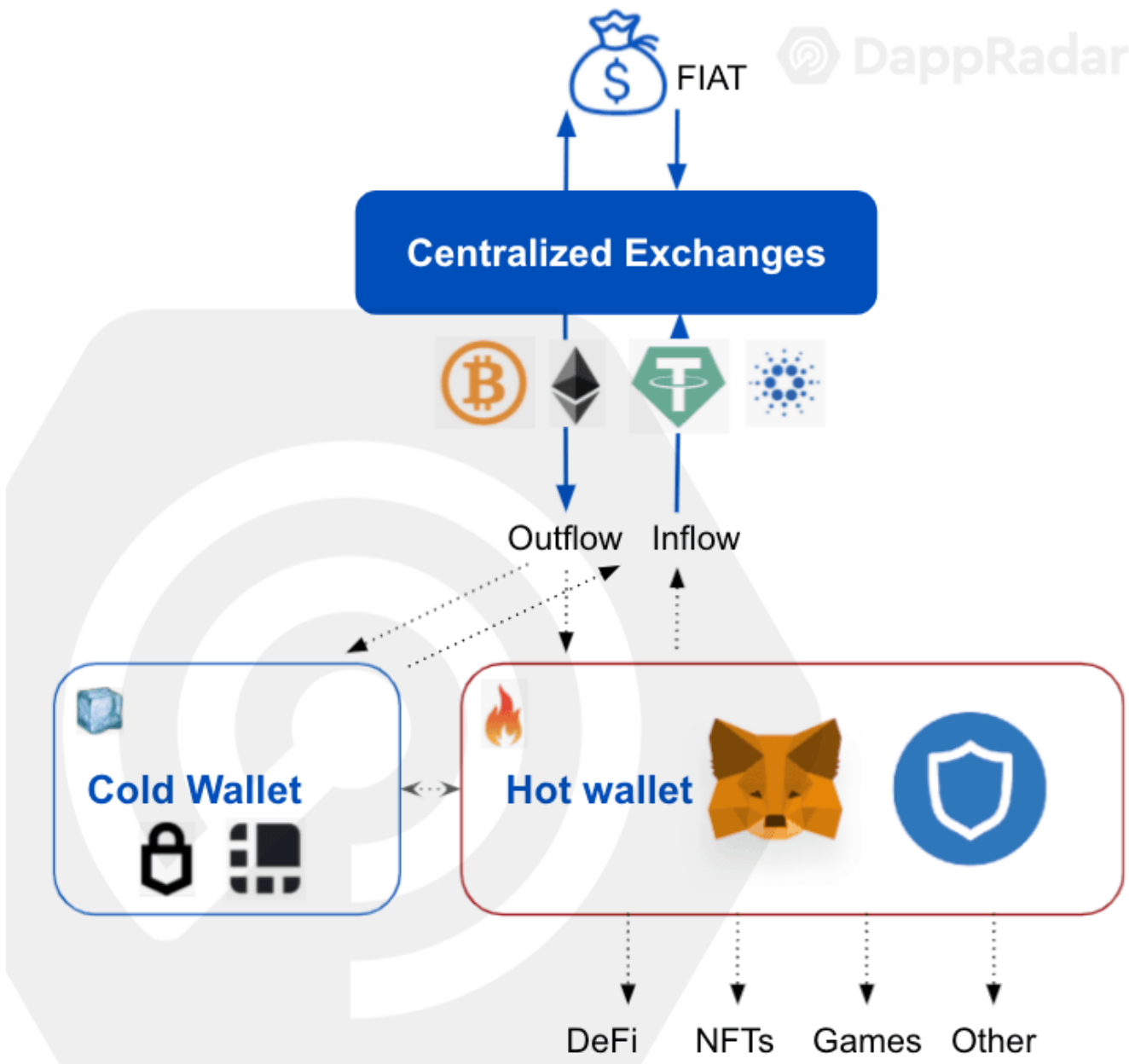


Figure 1: Modeling and Implementation of MultiSig Wallet

4.1 Design Diagram

A multisig wallet is a type of cryptocurrency wallet that requires more than one signature to authorize a transaction. This added layer of security helps to protect against theft and fraud, as multiple parties must agree to a transaction before it can be executed. To use a multisig wallet, multiple private keys are required to sign off on a transaction, with each key being held by a different party. For example, a company might use a 2-of-3 multisig wallet, where two out of three people must sign off on a transaction before it can be executed.

When a transaction is initiated in a multisig wallet, it is broadcast to the network and verified by the nodes on the network. Once the required number of signatures have been collected, the transaction is then executed and added to the blockchain. This process can take longer than a standard transaction, but the added security is often worth the extra time. Multisig wallets are often used by businesses, exchanges, and other high-value entities that require an extra layer of security for their cryptocurrency holdings.

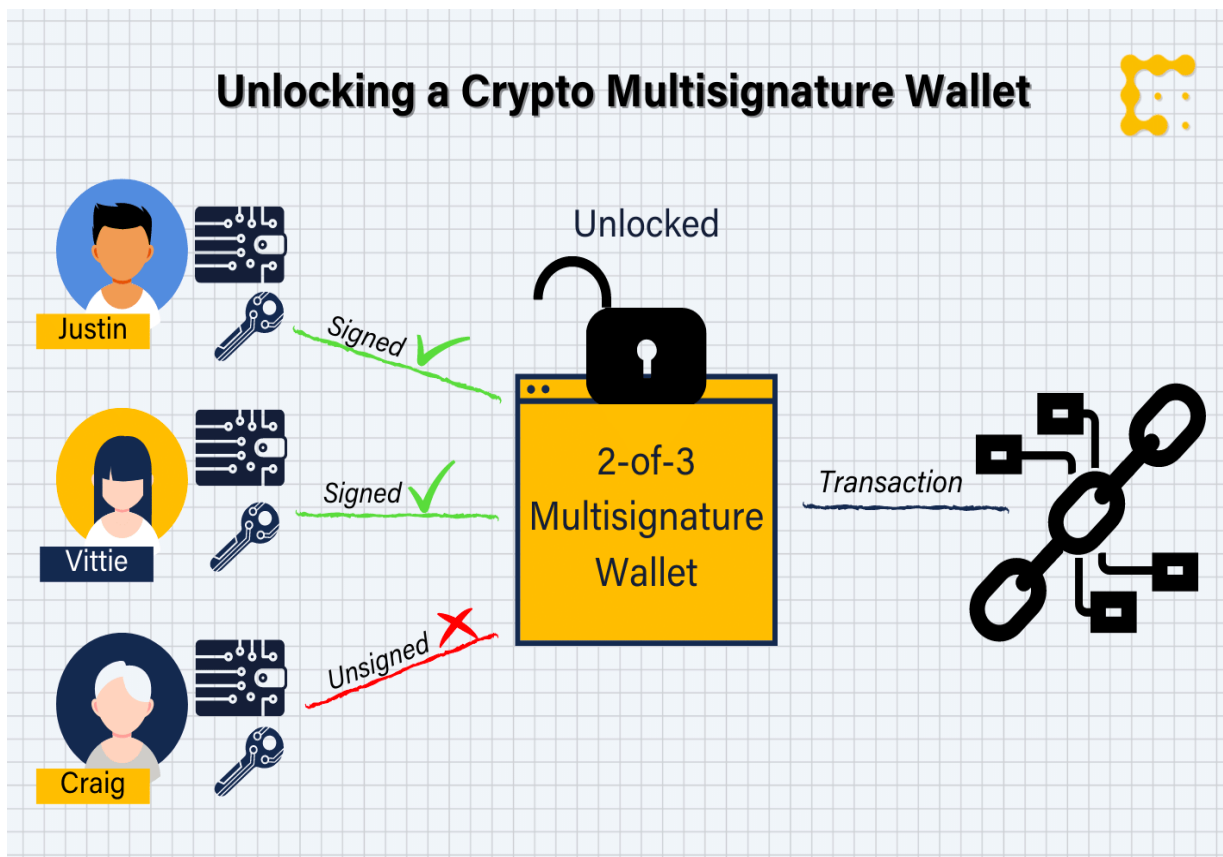


Figure 2 : Transaction of MultiSignature Wallet

4.1.1 Use Case Diagram

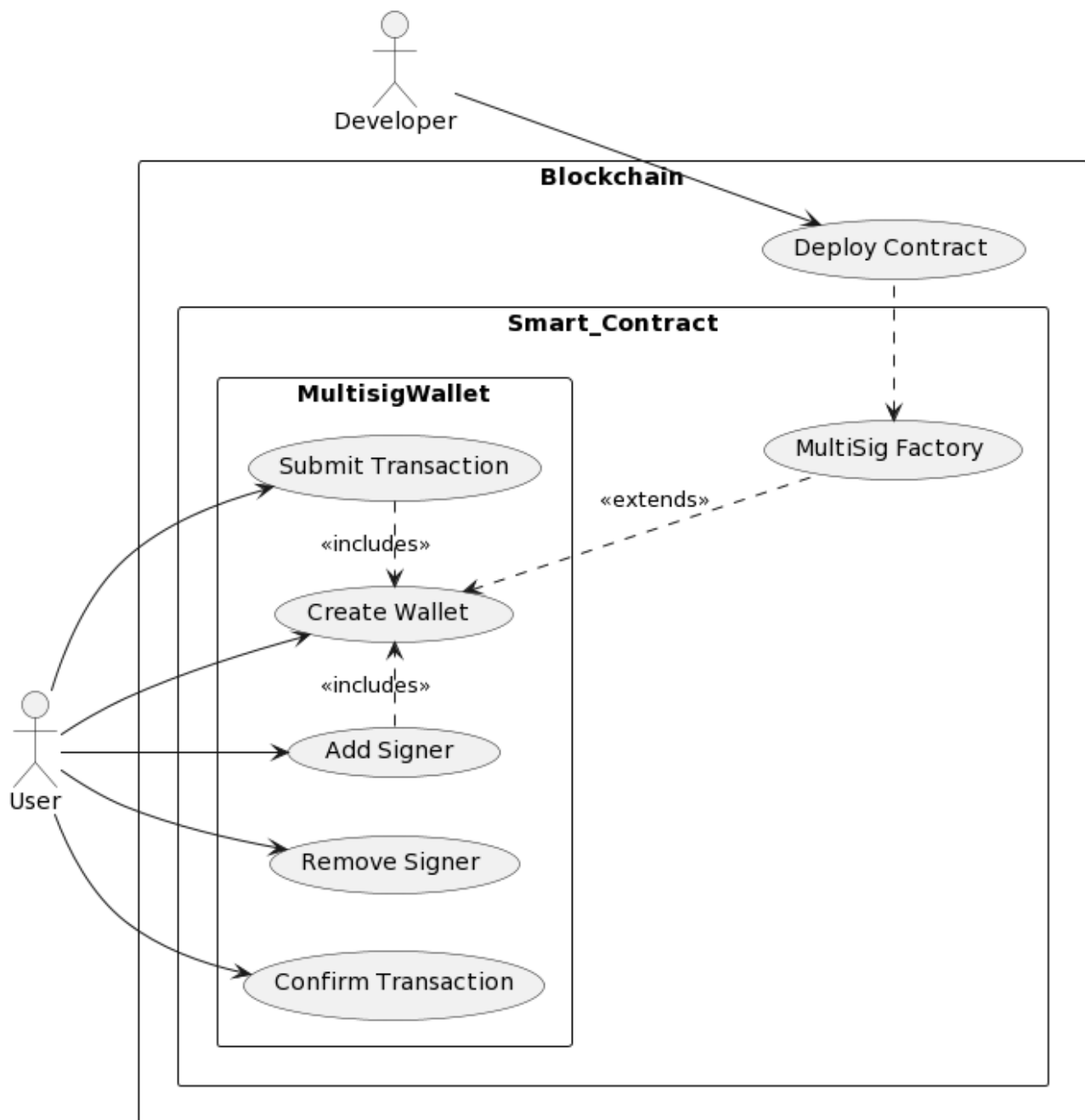


Figure 3. Use Case Diagram for Multi Signature Wallet

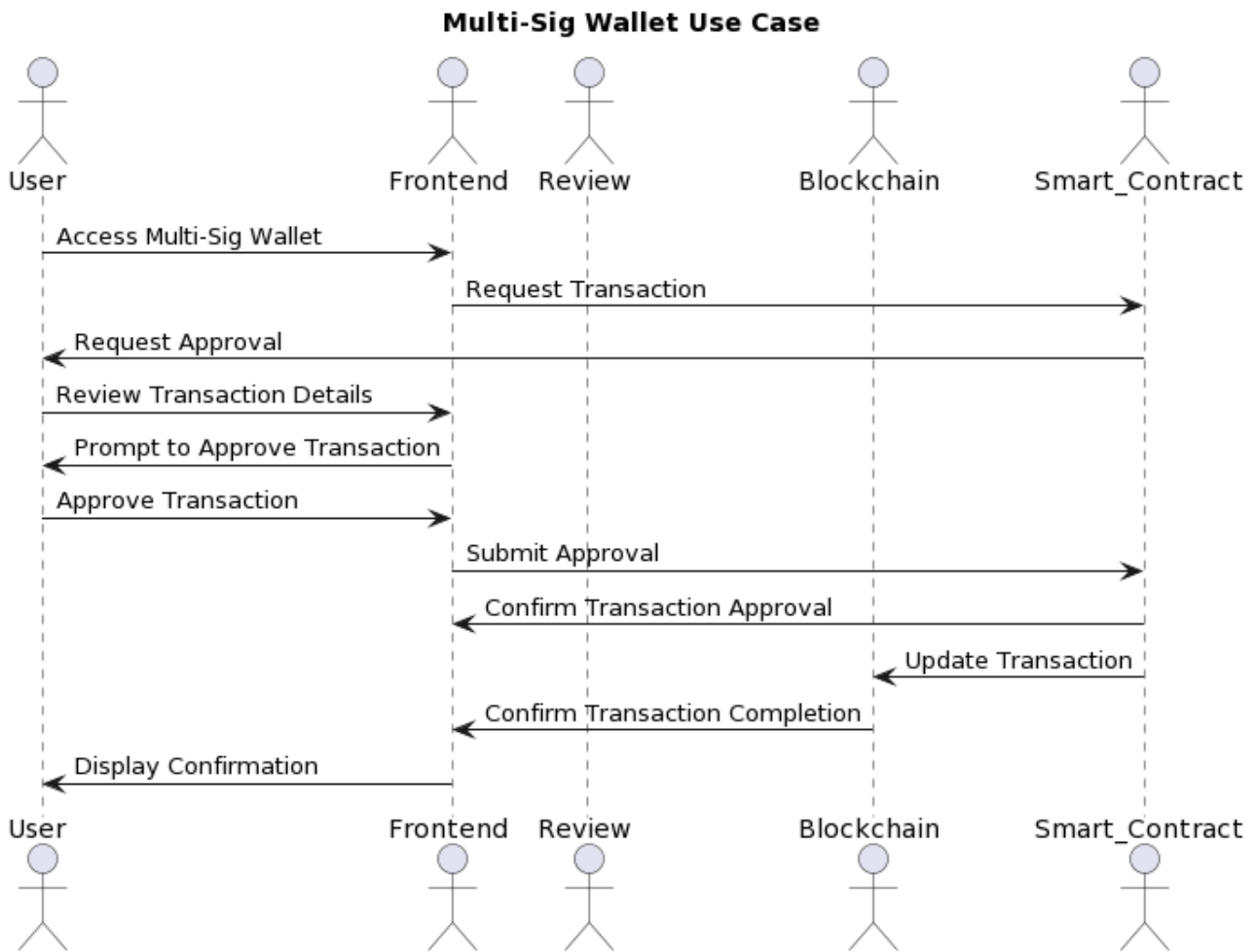


Figure 4. Sequence Diagram of Multi Signature Wallet

4.1.4 Flow Chart

Multisignature Wallet Transaction

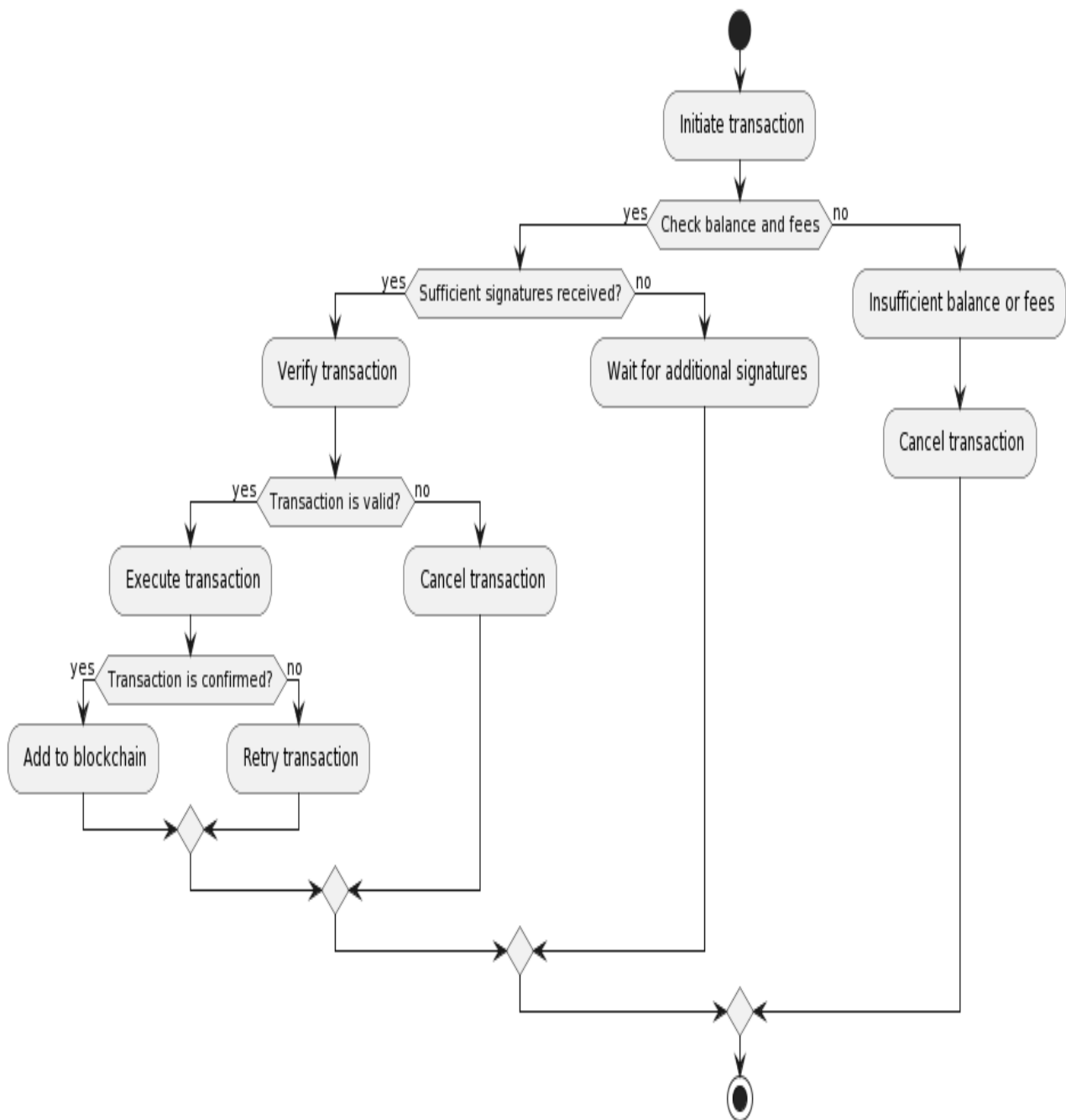


Figure 5. Flow Chart of the Multi Sig Wallet

4.2 Implementation Details

4.2.1 Ethereum Blockchain

The Ethereum blockchain is a distributed, decentralised ledger that enables users to log and verify transactions independently of a centralised authority. A network of nodes, or machines that execute the Ethereum software and keep a copy of the blockchain, powers the Ethereum blockchain. These nodes update the blockchain with fresh blocks of transactions by agreeing on the sequence and validity of transactions using a consensus mechanism, such as proof of work or proof of stake.

A broad variety of applications, including as smart contracts and decentralised applications (dapps), are supported by the Ethereum blockchain. When specific criteria are satisfied, smart contracts are computer programmes that run on the Ethereum blockchain that automatically carry out the terms of a contract. Dapps, or decentralised applications, are programmes that are constructed on top of the Ethereum blockchain and employ smart contracts to give users a decentralised and trustworthy way to access services and programmes. A platform for producing and managing digital assets, such as cryptocurrencies and non-fungible tokens (NFTs), is the Ethereum blockchain. Overall, the Ethereum blockchain offers a strong and adaptable foundation for creating and implementing additional blockchain-based applications, including decentralised ones.

Rainbow kit

In order to make it simpler to create decentralised apps (dApps) on top of Ethereum and other blockchain platforms, Rainbow Kit is a collection of UI elements and frameworks. The Rainbow Kit is a collection of pre-built React components created by the Connex team that can be quickly included into a range of dApps, including wallets, exchanges, and other financial applications.

The simplicity and adaptability of Rainbow Kit are two of its main advantages. The components' usage of React allows for easy customization and integration into a variety of current applications. Additionally, the components are made to be highly modular so that developers can pick and choose which ones to use and modify them as necessary.

Ether.js

For communicating with the Ethereum blockchain, a well-liked and widely-used JavaScript library is called ether.js. Ether.js, created and maintained by the ethers.io team, gives programmers a straightforward and user-friendly API for interacting with the Ethereum blockchain. This includes sending and receiving transactions, querying smart contracts, and controlling user accounts.

The simplicity and adaptability of Ether.js are two of its main advantages. Ether.js is a contemporary JavaScript framework that can be readily integrated into a variety of projects, including online and mobile apps, due to its modular structure and flexible architecture.

Support for several Ethereum networks, including as the mainnet, testnets, and private networks, is another crucial aspect of Ether.js. As a result, it serves as a flexible tool for developers working on a variety of projects and use cases.

Overall, many developers who work within the Ethereum ecosystem now consider Ether.js to be a necessary tool. Ether.js has made it easier for developers to create decentralised apps and take part in the fascinating world of blockchain technology by offering an easy-to-use API for communicating with the Ethereum blockchain.

4.2.2 Solidity Programming Language

Solidity^[12] is a high-level programming language that is used to develop smart contracts on the Ethereum network. In order to make it easy for developers who are already familiar with these languages to learn and use, it is meant to be comparable to other commonly used programming languages, such as JavaScript. Solidity has been widely adopted by the Ethereum community, and there are many tools, libraries, and frameworks that support it, making it easier to write and utilise smart contracts. Solidity is also compatible with the Ethereum Virtual Machine (EVM), the software used to execute smart contracts on the Ethereum network. This makes it possible for Ethereum accounts and Solidity contracts that are running on the EVM to communicate with one another and with other smart contracts.

WAGMI

Wagmi is a collection of smart contract utilities and libraries that are used in the One Wallet project to provide additional functionality and security. Here are some of the key features of Wagmi that are relevant to the One Wallet project:

Multisig Wallet Library: A multisig wallet library from Wagmi makes it possible to create a multisig wallet contract on the Ethereum network^[10]. To implement the multisig capability, which enables many wallet owners to control and administer the wallet, the One Wallet project uses this library.

Timelock Library: The development of a time-locked smart contract on the Ethereum blockchain is also made possible using Wagmi's timelock library. The One Wallet project uses this library to offer a timelock feature that necessitates a waiting period before some transactions can be carried out. This lessens the possibility of fraud and prevents unauthorised access to the wallet.

Access Control: Access control lists for smart contracts may be created using the access control library that Wagmi offers. Based on the user's role and permissions, this library is used in the One Wallet project to limit access to specific wallet features and functionalities.

Security Features: Wagmi has a number of security components that assist shield smart contracts from threats and weaknesses. These functions consist of gas optimisation, input validation, and error handling. The One Wallet project requires a high degree of security to secure user assets, hence these security measures are a crucial component.

Overall, Wagmi is a crucial part of the One Wallet project since it offers a variety of tools and libraries that make it possible to build a reliable multisignature wallet on the Ethereum network. The development team was able to create a strong and dependable wallet that satisfies user demands and safeguards user assets by utilising Wagmi's capabilities.

Multisig

A multisig wallet is a particular kind of digital wallet designed to store and manage coins that need many valid signatures to approve a transaction. A typical digital wallet is vulnerable to theft or loss if the private key is stolen since only one private key is needed to authorise transactions. On the other hand, a multisig wallet is more secure and less prone to theft or loss of cash since it needs several valid

signatures to approve a transaction. A transaction must get the approval of two or more participants in order to be completed in a multisig wallet. A multisig wallet, for instance, can demand that two out of three participants authorise a transaction before it can be carried out. This makes it far more challenging to steal money since even if one of the parties' private keys is stolen, an attacker would still require access to the other party's private key to authorise a transaction. Depending on the platform or blockchain being utilised, multisig wallets may be deployed in many ways. While some wallets employ a decentralised strategy in which the multisig wallet is managed by a smart contract on the blockchain, others employ a centralised strategy in which a dependable third party manages the multisig wallet on behalf of the users.

Multi-chain^[8]

Access to Multiple Blockchains: With a multi-chain wallet, users may access many blockchains and the cryptocurrencies they are connected to through a single wallet interface. Users benefit from improved convenience and flexibility as a result of being able to manage all of their digital assets in a single wallet rather than having to utilise several wallets. **Asset diversification:** By supporting numerous blockchains, a multi-chain wallet enables users to spread out their cryptocurrency assets over other networks, lowering their exposure to the dangers or vulnerabilities of any one blockchain. **Reduced Transaction Fees:** By enabling users to select the blockchain network with the lowest fees for a given transaction, a multi-chain wallet can help lower transaction prices. This can be especially useful during periods of high network congestion, when fees can be prohibitively high on some networks. **Improved Security:** By enabling users to distribute their bitcoin^[11] holdings over several blockchains, a multi-chain wallet can increase security by lowering the likelihood of a single point of failure or vulnerability. Advanced security features like multi-signature authentication and hardware wallet integration are also available in some multi-chain wallets. **Interoperability:** By enabling users to swap and move assets between multiple chains inside the same wallet, a multi-chain wallet can aid in the promotion of interoperability between various blockchain networks. This can foster increased collaboration across various projects and apps and assist lessen fragmentation in the blockchain ecosystem. **Future-Proofing:** By supporting new blockchain networks and cryptocurrencies, a multi-chain wallet can aid users in future-proofing their holdings of digital assets. Users may benefit from having more freedom and assurance in their long-term investing strategy as a result.

4.2.5 Smart Contract Implementation^[9]

The Ethereum blockchain's smart contract technology is used by the decentralised application (dapp) I've just described to make it easier to acquire and trade non-fungible tokens (NFTs). The ERC1155 standard, which enables the development of many token kinds inside a single contract, is used to construct the smart contract. This permits the fractionalization of NFTs, which gives users the option to sell their NFTs at a loss in order to access liquidity more quickly. The smart contract gives the seller a number of ERC1155 tokens in return, which may then be sold in the marketplace for a profit.

Because it enables the efficient and secure management of various token types within a single contract, the ERC1155 standard is used. This is advantageous for the dapp since it permits the fractionalization of NFTs, which enables users to trade their NFTs for a number of tokens at a discount when they sell their NFTs.

The openZeplin library, which offers a variety of practical functions and capabilities for working with ERC1155 tokens, is used to construct the ERC1155 smart contract. The openZeplin library has more complex functions for maintaining and transferring tokens in addition to simple operations for creating and burning tokens. The openZeplin library enables the dapp to manage the multiple token kinds used in the contract quickly and effectively.

The Polygon network, which the contract is placed on, enables quicker and less expensive transactions on the Ethereum blockchain. By enabling users to quickly and securely buy and sell NFTs on the Ethereum blockchain, along with the advantages of fractionalization and shorter transaction times, this improves the effectiveness and convenience of the dapp.

Overall, the dapp is able to safely and effectively handle the buying and selling of NFTs on the Ethereum blockchain thanks to the usage of the openZeplin library, the ERC1155 standard, and the Polygon network. The fractionalization of NFTs increases the market's flexibility and liquidity, while the usage of the Polygon network makes transactions quicker and less expensive.

4.2.6 Functionalities

```
function emitOwners(  
    address _contractAddress ↑,  
    address[] calldata _owners ↑,  
    uint256 _signaturesRequired ↑  
) external onlyRegistered {  
    emit Owners(_contractAddress ↑, _owners ↑, _signaturesRequired ↑);  
}
```

ftrace | funcSig

```
function numberOfMultiSigs() public view returns (uint256) {  
    return multiSigs.length;  
}
```

ftrace | funcSig

```
function getMultiSig(uint256 _index ↑)  
    public  
    view  
    returns (  
        address (uint256) Argument getMultiSig.signaturesRequired  
        uint256 signaturesRequired ↑,  
        uint256 balance ↑  
    )  
{  
    MultiSigWallet multiSig = multiSigs[_index ↑];  
    return (  
        address(multiSig),  
        multiSig.signaturesRequired(),  
        address(multiSig).balance  
    );  
}
```

- emitOwners: This function emits an Owners event with the contract address, list of owners, and the required number of signatures.
- numberOfMultiSigs: This function returns the number of deployed multisig wallets.
- getMultiSig: This function returns the address, required signatures, and balance of a specific multisig wallet based on its index in the multiSigs array. It first retrieves the multisig wallet using the index, then returns its address, required signatures, and balance as a tuple.

```

function create2(
    address[] calldata _owners↑,
    uint256 _signaturesRequired↑,
    string calldata _name↑
) public payable {
    uint256 id = numberOfMultiSigs();

    bytes32 _salt = keccak256(
        abi.encodePacked(abi.encode(_name↑, address(msg.sender)))
    );

    /**-----
     * create2 implementation
     * -----*/
    address multiSig_address = payable(
        Create2.deploy(
            msg.value,
            _salt,
            abi.encodePacked(
                type(MultiSigWallet).creationCode,
                abi.encode(_name↑, address(this))
            )
        )
    );

    MultiSigWallet multiSig = MultiSigWallet(payable(multiSig_address));

    /**-----
     * init remaining values
     * -----*/
    multiSig.init(_owners↑, _signaturesRequired↑);

    multiSigs.push(multiSig);
    existsMultiSig[address(multiSig_address)] = true;

    emit Create2Event(
        id,
        _name↑,
        address(multiSig),
        msg.sender,
        _owners↑,
        _signaturesRequired↑
    );
    emit Owners(address(multiSig), _owners↑, _signaturesRequired↑);
}

```

This function `create2` is a public function that creates a new instance of a `MultiSigWallet`. The `MultiSigWallet` contract is a contract for managing a multi-signature wallet, where multiple owners can control the wallet and multiple signatures are required to execute a transaction.

The three input arguments for the function are `_owners`, `_signaturesRequired`, and `_name`. The owners of the new wallet instance are represented by an array of addresses in the `_owners` argument. The number of signatures needed to complete a transaction is indicated by the integer value of the `_signaturesRequired` property. The name of the new wallet instance is represented by the string in the `_name` argument.

By invoking the `numberOfMultiSigs` function, the function first finds the ID of the new multi-signature wallet instance. The `_name` field and the transaction sender's address are then hashed to produce a salt value. The `Create2` function uses the salt value to deploy the new `MultiSigWallet` contract in a deterministic manner.

By deploying the contract's construction code, which contains the constructor arguments `_name` and the address of the active contract, the `Create2` function creates the new `MultiSigWallet` instance. The function returns the new instance's address.

Once the new instance is created, the function initializes the remaining values of the `MultiSigWallet` instance by calling the `init` function of the `MultiSigWallet` contract, passing in the `_owners` and `_signaturesRequired` parameters.

The function then updates the `existsMultiSig` mapping for the new instance to true and adds the new `MultiSigWallet` instance to the `multiSigs` array.

Finally, the function emits two events: `Create2Event` and `Owners`. The `Create2Event` event contains information about the new instance, including its ID, name, address, and owner addresses. The `Owners` event emits the same owner addresses and signature requirements as the `Create2Event`.

```
function computedAddress(string calldata _name↑)
    public
    view
    returns (address)
{
    bytes32 bytecodeHash = keccak256(
        abi.encodePacked(
            type(MultiSigWallet).creationCode,
            abi.encode(_name↑, address(this))
        )
    );

    bytes32 _salt = keccak256(
        abi.encodePacked(abi.encode(_name↑, address(msg.sender)))
    );
    address computed_address = Create2.computeAddress(_salt, bytecodeHash);

    return computed_address;
}
```

This function determines the address that would be produced if the create2 function were used to deploy a new instance of the MultiSigWallet contract with the specified `_name` argument.

Prior to calculating the address using the `computeAddress` function of the Create2 contract, it computes the hash of the MultiSigWallet contract formation code and the constructor arguments (`_name` and the contract address of the calling contract).

The calculated address is then reported back to the client.

```
function addSigner(address newSigner↑, uint256 newSignaturesRequired↑)
    public
    onlySelf
    onlyValidSignaturesRequired
{
    if (newSigner↑ == address(0) || isOwner[newSigner↑]) {
        revert INVALID_SIGNER();
    }

    isOwner[newSigner↑] = true;
    owners.push(newSigner↑);
    signaturesRequired = newSignaturesRequired↑;

    emit Owner(newSigner↑, true);
    multiSigFactory.emitOwners(
        address(this),
        owners,
        newSignaturesRequired↑
    );
}
```

The code first verifies that the new signer is a valid address and is not the wallet's existing owner. The function reverts with an error message if one of these requirements is not satisfied.

The function adds the new signer as an owner of the wallet if they are a legitimate signer by changing the `isOwner` mapping to true and adding their address to the `owners` array. Additionally, the `signaturesRequired` variable is updated with the new value.

The function ends by emitting an `Owner` event containing the address of the new signer and true to show that they have been added as an owner.

```

function removeSigner(address oldSigner↑, uint256 newSignaturesRequired↑)
    public
    onlySelf
    onlyValidSignaturesRequired
{
    if (!isOwner[oldSigner↑]) {
        revert NOT_OWNER();
    }

    _removeOwner(oldSigner↑);
    signaturesRequired = newSignaturesRequired↑;

    emit Owner(oldSigner↑, false);
    multiSigFactory.emitOwners(
        address(this),
        owners,
        newSignaturesRequired↑
    );
}

```

An owner can be removed from the multisig wallet contract using this method. It requires the two parameters newSignaturesRequired and oldSigner. The owner's address that is to be dropped from the multisig wallet contract is specified by the oldSigner option. The new number of signatures required for subsequent transactions following the owner's removal is specified by the newSignaturesRequired option.

The function first verifies that the owner of the multisig wallet contract at the oldSigner address is legitimate. It returns with an error message if the owner is invalid. The _removeOwner() method is used to remove the owner from the owners array and the isOwner mapping if the oldSigner address is valid. After the removal of the owner, the signaturesRequired variable is updated with the newSignaturesRequired parameter. An Owner event is emitted with the oldSigner address and a false boolean value to indicate that the owner has been removed. The emitOwners() function of the multiSigFactory is called with the address of the multisig wallet contract, the owners array, and the newSignaturesRequired parameter to emit an Owners event.


```

function _removeOwner(address _oldSigner) private {
    isOwner[_oldSigner] = false;
    uint256 ownersLength = owners.length;
    address lastElement = owners[ownersLength - 1];
    // check if the last element of the array is the owner to be removed
    if (lastElement == _oldSigner) {
        owners.pop();
        return;
    } else {
        // if not then iterate through the array and swap the owner to be removed with the last element in the array
        for (uint256 i = ownersLength - 2; i >= 0; ) {
            if (owners[i] == _oldSigner) {
                address temp = owners[i];
                owners[i] = lastElement;
                lastElement = temp;
                owners.pop();
                return;
            }
            unchecked {
                --i;
            }
        }
    }
}

```

To delete a specific owner from the owners array in the contract, use this secret internal method. It initially sets the isOwner mapping for the provided owner to false before determining if the owner to be deleted is represented by the final member of the owners array. If it is, the pop() method is used to delete it, and the function then returns. If it isn't, the method iterates over the array of owners, beginning with the second-to-last member, and determines if each entry is the owner who needs to be deleted. When it locates the first instance, it uses pop() to get rid of the last element in the array and swap the owner with it before returning.

```

function executeTransaction(
    address payable to↑,
    uint256 value↑,
    bytes calldata data↑,
    bytes[] calldata signatures↑
) public onlyOwner returns (bytes memory) {
    uint256 _nonce = nonce;
    bytes32 _hash = getTransactionHash(_nonce, to↑, value↑, data↑);

    nonce = _nonce+1;

    uint256 validSignatures;
    address duplicateGuard;
    // get a local reference of the length to save gas
    uint256 signatureLength = signatures↑.length;
    for (uint256 i = 0; i < signatureLength; ) {
        address recovered = recover(_hash, signatures↑[i]);
        if (recovered <= duplicateGuard) {
            revert DUPLICATE_OR_UNORDERED_SIGNATURES();
        }
        duplicateGuard = recovered;

        if (isOwner[recovered]) {
            validSignatures++;
        }
        unchecked {
            ++i;
        }
    }

    if (validSignatures < signaturesRequired) {
        revert INSUFFICIENT_VALID_SIGNATURES();
    }

    (bool success, bytes memory result) = to↑.call{value↑: value↑}(data↑);
    if (!success) {
        revert TX_FAILED();
    }

    emit ExecuteTransaction(
        msg.sender,
        to↑,
        value↑,
        data↑,
        _nonce,
        _hash,
        result
    );
    return result;
}

```

}

A public function called `executeTransaction` uses the supplied arguments to carry out a transaction. The caller must be the owner of the multi-signature wallet. It accepts the transaction parameters, which include the data payload, the destination address, and the amount of value to send. Additionally, it accepts a variety of valid signatures from the multi-sig wallet's owners.

Prior to computing the transaction's hash and incrementing the nonce, it obtains the current nonce value. Following that, it iterates through the array of signatures to validate each one and count how many are valid. It makes sure there aren't any duplicate or randomly arranged signatures.

Next, it checks if the number of valid signatures is greater than or equal to the required number of signatures for the multi-sig wallet.

If the number of valid signatures is sufficient, it attempts to execute the transaction by calling the address with the value and data provided. If the transaction fails, it reverts the changes and returns an error.

Finally, it emits an `ExecuteTransaction` event with the relevant information and returns the transaction result.

4.2.7 Scaffold Eth2

Scaffold-eth 2 is a developer tool suite and framework for building decentralized applications (DApps) on the Ethereum 2.0 network. It is a successor to the original Scaffold-eth framework, which was developed for Ethereum 1.0. Scaffold-eth 2 provides a set of pre-built smart contracts, front-end components^[14], and developer tools to help streamline the DApp development process. It is designed to be modular, flexible, and customizable, allowing developers to easily integrate their own smart contracts and custom front-end designs. Some of the key features of Scaffold-eth 2 include: Integration with Ethereum 2.0: Scaffold-eth 2 is built specifically for the Ethereum 2.0 network, which uses a proof-of-stake consensus mechanism instead of the traditional proof-of-work used in Ethereum 1.0. Pre-built Smart Contracts: Scaffold-eth 2 includes a set of pre-built smart contracts for common DApp functionalities such as user authentication, ERC-20 tokens, and NFTs. Customizable Front-End: Scaffold-eth 2 provides a set of customizable front-end components, including a user interface for interacting with smart contracts and a development server for testing and deployment. Integration with Web3 Libraries: Scaffold-eth 2 integrates with popular Web3 libraries such as `ethers.js` and `web3.js`, making it easy to interact with smart contracts and the Ethereum network. Developer Tools: Scaffold-eth 2 includes a set of developer tools such as a local blockchain simulator, a testnet deployment script, and a contract deployment script, to help streamline the development and deployment process.

Overall, Scaffold-eth 2 provides a comprehensive tool suite for developers to build and deploy DApps^[15] on the Ethereum 2.0 network, with a focus on modularity, flexibility, and customization.

This information can be used later by other functions in the contract, such as the buyTokens function, which allows users to buy fractionalized tokens from the contract. Overall, these two functions provide a convenient and user-friendly way for users to list their NFTs on the contract's marketplace, and to mint and sell fractionalized tokens.

4.2.8 Metamask

Users may connect with dapps and manage their Ethereum accounts using the browser plugin MetaMask. Popular web browsers like Google Chrome and Mozilla Firefox support MetaMask, which is simple to set up and use. After being installed, MetaMask offers users a simple interface that enables them to sign transactions, establish and manage Ethereum accounts, and examine account balances and transaction history. Ledger and Trezor hardware wallets, which offer additional protection for customers who wish to keep their Ethereum holdings on a hardware device, are supported by MetaMask as well.

Because it gives users a convenient way to handle their Ethereum accounts and engage with dapps, MetaMask is essential for any dapp that runs on Ethereum. You integrated MetaMask into your dapp to let users control their Ethereum accounts and sign payments made to your smart contract. This made it possible for consumers to engage with your dapp quickly and simply without needing to manage or install any additional software or Ethereum nodes. Because it signs transactions using user-controlled private keys, which are never shared with the dapp or any other third party, MetaMask also gave users a secure way to do so.

Chapter 5. Testing

5.1 Testing Tool

Hardhat

A complete toolkit called Hardhat is intended to speed up the creation and testing of smart contracts for the Ethereum network. It has a number of features and plugins that may be used to optimise code, lower transaction costs, and find and repair problems early in the development cycle. Smart contracts may be tested automatically thanks to the built-in testing framework, and developers can check the state of their contracts at various stages of execution thanks to the debugging tools. A gas estimate tool is also available from Hardhat to aid in maximising the effectiveness of smart contracts and lowering transaction costs. The forking functionality further enables programmers to build a local instance of the Ethereum blockchain for testing and development. Because Hardhat's plugin system is so flexible, programmers can integrate it with other well-liked development frameworks and services like Truffle, Ganache, and ethers.js. Due to its extensive capabilities and simplicity of use, Hardhat has grown to be a well-liked option among Ethereum developers. Hardhat is an all-around strong and versatile tool suite.

5.2 Component decomposition and type of testing required

5.2.1. Component Decomposition:

In order to implement a multisig wallet, there are several components that need to be developed, tested, and integrated together. These components include:

- **Smart Contract:** This is the core component of the multisig wallet implementation and is responsible for managing the authorization and execution of transactions. The smart contract should be thoroughly tested for correctness and security vulnerabilities.
- **User Interface:** This component provides a graphical or command-line interface for users to interact with the multisig wallet. The user interface should be tested for usability and functionality.

- **Client Libraries:** These libraries provide an interface for developers to interact with the multisig wallet using various programming languages. The client libraries should be tested for correctness and compatibility with the smart contract.
- **Integration Code:** This code integrates the various components of the multisig wallet implementation together. It should be tested for compatibility and correctness.

5.2.2. Type of Testing Required:

In order to ensure that the multisig wallet implementation is correct and secure, several types of testing are required. These include:

- **Unit Testing:** This testing involves testing each individual component in isolation to ensure that it functions correctly.
- **Integration Testing:** This testing involves testing the integration of the various components to ensure that they work together correctly.
- **Functional Testing:** This testing involves testing the overall functionality of the multisig wallet implementation to ensure that it meets the requirements.
- **Security Testing:** This testing involves testing the multisig wallet implementation for security vulnerabilities, such as attacks on the smart contract or user interface.
- **Performance Testing:** This testing involves testing the performance of the multisig wallet implementation under various loads to ensure that it can handle the expected transaction volume.

Overall, thorough testing of the multisig wallet implementation is crucial to ensure its correctness and security. By decomposing the various components and identifying the types of testing required, you can ensure that the multisig wallet implementation is thoroughly tested and ready for deployment.

5.3 List All Test Cases

The "MultiSigWallet" smart contract is used for managing a multi-signature wallet where multiple users can sign transactions before they are executed on the blockchain.

The test suite is divided into two main parts: "Deployment" and "Testing MultiSigWallet functionality".

Deployment

This test suite contains two tests that are focused on the deployment of the "MultiSigWallet" smart contract and the "TestERC20Token" token contract.

IsOwner should return true for the deployer owner address

This test checks if the `isOwner` function of the `MultiSigWallet` contract returns `true` for the deployer owner address. If this test passes, it confirms that the deployment of the `MultiSigWallet` contract was successful and that the deployer is an owner of the multi-sig wallet.

MultiSig Wallet should own all the TestERC20Token token

This test checks if the `MultiSigWallet` contract owns all the tokens of the `TestERC20Token` token contract. If this test passes, it confirms that the deployment of the `TestERC20Token` contract was successful and that the tokens were transferred to the `MultiSigWallet` contract.

Testing MultiSigWallet functionality

This test suite contains multiple tests that are focused on testing the functionality of the `MultiSigWallet` contract.

Adding a new signer, then removing it

This test checks if a new signer can be added to the multi-sig wallet and then removed from it successfully. It creates a new signer address and adds it to the multi-sig wallet with a required signature count of 2. After adding the new signer, it confirms that the new signer address is now an owner of the wallet. Then, it removes the new signer and confirms that the new signer address is no longer an owner of the wallet.

Trying to remove an owner who is the last signer

This test checks if the last owner of the multi-sig wallet can be removed successfully. It gets the owner address and signature count of the last owner of the wallet and tries to remove it. If this test passes, it confirms that the `removeSigner` function of the `MultiSigWallet` contract works correctly even when removing the last signer from the array.

Trying to remove an owner who is not the last signer

This test checks if an owner of the multi-sig wallet who is not the last signer can be removed successfully. It creates a new signer address and adds it to the multi-sig wallet with a required signature count of 2. Then, it tries to remove the first owner of the wallet. If this test passes, it confirms that the `removeSigner` function of the `MultiSigWallet` contract works correctly even when removing an owner who is not the last signer in the array.

Trying to remove a non-existent owner

This test checks if an owner who does not exist in the multi-sig wallet can be removed successfully. It creates a new signer address and tries to remove it from the wallet. If this test passes, it confirms that the `removeSigner` function of the `MultiSigWallet` contract works correctly even when trying to remove an owner who does not exist in the array.

Trying to remove an owner with an invalid signature

This test checks if an owner of the multi-sig wallet can be removed successfully with an invalid signature. It creates a new signer address and adds it to the multi-sig wallet with a required signature count of 2


```

Dell@DESKTOP-40CUEBT MINGW64 ~/Desktop/major/multisig-se2 (master)
$ yarn hardhat:test

MultiSigWallet Test
Deployment
  ✓ isOwner should return true for the deployer owner address
  ✓ Multi Sig Wallet should own all the TestERC20Token token
Testing MultiSigWallet functionality
  ✓ Adding a new signer, then removing it
  ✓ Adding a new signer, then removing the old one
  ✓ Adding a new signer - execute with owner
  ✓ Transaction reverted: Remove the only signer
  ✓ Transaction reverted: Invalid MultiSigWallet, more signatures required than signers
  ✓ Transaction reverted: Update Signatures Required to 2 - trying to lock all the funds in the wallet, because there is only 1 signer
  ✓ Transferring 0.1 eth to addr1
  ✓ Allowing addr1 to spend 10 TestERC20Tokens. Then addr1 transfers the TestERC20Tokens to addr2

-----|-----|-----|-----|
| Solc version: 0.8.17 | Optimizer enabled: false | Runs: 200 | Block limit: 30000000 gas |
|-----|-----|-----|-----|
| Methods |
|-----|-----|-----|-----|
| Contract | Method | Min | Max | Avg | # calls | gas (avg) |
|-----|-----|-----|-----|-----|
| MultiSigFactory | create2 | 2606707 | 2635153 | 2609293 | 11 | - |
|-----|-----|-----|-----|-----|
| MultiSigWallet | executeTransaction | 70602 | 134858 | 98295 | 9 | - |
|-----|-----|-----|-----|-----|
| TestERC20Token | transferFrom | - | - | 55532 | 1 | - |
|-----|-----|-----|-----|-----|
| Deployments |
|-----|-----|-----|-----|
| MultiSigFactory | - | - | - | 3872746 | 12.9 % | - |
|-----|-----|-----|-----|-----|
| TestERC20Token | - | 1173934 | 1173946 | 1173943 | 3.9 % | - |
|-----|-----|-----|-----|-----|

10 passing (5s)

```

Fig 15. Test Results

5.4 Limitations of the solution

Here are some potential limitations of the One Wallet multisig smart contract project:

- **Limited to EVM-compatible chains:** The current version of One Wallet is limited to EVM-compatible chains. This means that it may not be compatible with other blockchain platforms that use different virtual machines or consensus mechanisms. This could limit the potential user base of the wallet.
- **Manual entry of calldata:** While the wallet does allow for custom transactions by manually entering calldata, this could potentially lead to errors or security vulnerabilities if the user is not familiar with the format and structure of calldata.
- **Dependency on external libraries:** The One Wallet project relies on several external libraries such as scaffold-eth 2, nextjs, wagmi, etherjs, and rainbowkit. Any updates or changes to these libraries could potentially impact the functionality and reliability of the wallet.

- Security risks: Multisig wallets are generally considered more secure than single-owner wallets, but they are not immune to security risks. Any vulnerabilities in the smart contract code or external libraries used in the project could potentially lead to loss of funds or other security breaches.

Overall, while the One Wallet project has several useful features and is built on a robust technology stack, it is important to keep these limitations in mind when evaluating its potential use cases and suitability for different scenarios.

Chapter 6. Findings, Conclusions and Future Work

6.1 Findings

Several significant conclusions can be drawn from a smart contract multisig wallet like the One Wallet project's characteristics and features. Increased security is one of the main benefits of multisig wallets since transactions must be approved by multiple signatures from various owners. The possibility of security lapses or money theft may be reduced with the use of this feature. Additionally, it might be advantageous to think about integrating the wallet with other well-known blockchain ecosystems in order to maximise the wallet's potential user base and functionality. To ensure the stability and dependability of the wallet, it is essential to manage dependencies on external libraries. Debugging and testing are also essential for finding and fixing any possible security flaws or problems in the smart contract code. The user experience must be prioritised since features that are simple to use and intuitive for non-technical users are crucial to promoting wallet uptake and usage. The project team may contribute to maximising the potential success and impact of the wallet by taking these findings into consideration during the development and design of the One Wallet project.

6.2 Conclusion

In conclusion, the One Wallet project offers a major advancement in the creation of flexible and safe multisig smart contract wallets. The project offers a useful solution for customers looking for more security and functionality in their digital wallets by offering features like the capacity to add or remove owners, transfer currency, and deploy on evm compliant networks. The wallet's stability and dependability are increased by the usage of tools like scaffold-eth 2, Next.js, Wagmi, EtherJS, and RainbowKit, and its ability to manually input customised transaction calldata also gives some flexibility and customisation. There are, however, restrictions and possible areas for development, just like with every software project. In order to guarantee the success and acceptance of the wallet, it is crucial to manage dependencies, conduct testing and debugging, and give user experience priority. The project team may continue to make significant contributions to the creation of blockchain-based financial products and services by solving these issues and expanding on the advantages of the One Wallet initiative.

6.3 Future Work

Incorporating NFTs into the wallet will give users direct access to all of their digital assets on a single platform. Furthermore, it is also possible to add the capacity for NFT transfers.

The One Wallet project's future development will place a high priority on user experience and accessibility as a key area of focus. It will become more crucial to create and build tools that are simple to use and available to a wide variety of consumers, regardless of their level of technical ability, as the use of blockchain technology continues to rise.

References

- [1] Zou, W., Lo, D., Kochhar, P.S., Le, X.B.D., Xia, X., Feng, Y., Chen, Z. and Xu, B., 2019. Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering*, 47(10), pp.2084-2106
- [2] Bella, G., Cantone, D., Longo, C., Nicolosi Asmundo, M. and Santamaria, D.F., 2022. Blockchains Through Ontologies: The Case Study of the Ethereum ERC721 Standard in. In *International Symposium on Intelligent and Distributed Computing* (pp. 249-259).
- [3] Tikhomirov, S., 2017, October. Ethereum: state of knowledge and research perspectives. In *International Symposium on Foundations and Practice of Security* (pp. 206-221).
- [4] Han, J., Song, M., Eom, H., & Son, Y. (2021, March). An efficient multi-signature wallet in blockchain using bloom filter. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing* (pp. 273-281).
- [5] Huang, Y., Bian, Y., Li, R., Zhao, J.L. and Shi, P., 2019. Smart contract security: A software lifecycle perspective. *IEEE Access*, 7, pp.150184-150202.
- [6] Di Angelo, M. and Slazer, G., 2020, May. Wallet contracts on Ethereum. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 1-2). IEEE.
- [7] Ji, P. (2023) “The Advance of Cryptocurrency Wallet with Digital Signature”, *Highlights in Science, Engineering and Technology*, 39, pp. 1098–1103.
- [8] Li, Z. and Zhang, Z., 2020. Research and implementation of multi-chain digital wallet based on hash timelock. In *Blockchain and Trustworthy Systems: First International Conference, BlockSys 2019, Guangzhou, China, December 7–8, 2019, Proceedings 1* (pp. 175-182). Springer Singapore.
- [9] Ismailisufi, A., Popović, T., Gligorić, N., Radonjic, S. and Šandi, S., 2020, February. A private blockchain implementation using multichain open source platform. In *2020 24th International Conference on Information Technology (IT)* (pp. 1-4). IEEE.

- [10] Di Angelo, M. and Salzer, G., 2020, September. Characteristics of wallet contracts on Ethereum. In 2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS) (pp. 232-239). IEEE.
- [11] Nakamoto, S., 2008. Bitcoin: A peer-to-peer electronic cash system. Decentralized business review, p.21260.
- [12] Dannen, C., 2017. Introducing Ethereum and solidity (Vol. 1, pp. 159-160). Berkeley: Apress
- [13] Thakkar, M. and Thakkar, M., 2020. Next. js. Building React Apps with Server-Side Rendering: Use React, Redux, and Next to Build Full Server-Side Rendering Applications, pp.93-137.
- [14] Lazuardy, M.F.S. and Anggraini, D., 2022. Modern Front End Web Architectures with React. Js and Next. Js. Research Journal of Advanced Engineering and Science, 7(1), pp.132-141.
- [15] Besançon, L., Da Silva, C.F., Ghodous, P. and Gelas, J.P., 2022. A Blockchain Ontology for DApps Development. IEEE Access, 10, pp.49905-49933.

19103007 ONE WALLET - MULTISIG AND MULTICHAIN WALLET

ORIGINALITY REPORT

5%

SIMILARITY INDEX

4%

INTERNET SOURCES

2%

PUBLICATIONS

3%

STUDENT PAPERS

PRIMARY SOURCES

1

github.com

Internet Source

2%

2

www.coursehero.com

Internet Source

<1%

3

Submitted to Sheffield Hallam University

Student Paper

<1%

4

Jongbeen Han, Mansub Song, Hyeonsang Eom, Yongseok Son. "An efficient multi-signature wallet in blockchain using bloom filter", Proceedings of the 36th Annual ACM Symposium on Applied Computing, 2021

Publication

<1%

5

network.bepress.com

Internet Source

<1%

6

apps.dtic.mil

Internet Source

<1%

7

Submitted to Clark University

Student Paper

<1%

8

scholars.cityu.edu.hk

Internet Source

<1 %

9

Submitted to Franklin University

Student Paper

<1 %

10

Submitted to University of Portsmouth

Student Paper

<1 %

11

Submitted to MAHSA University

Student Paper

<1 %

12

erepository.uonbi.ac.ke:8080

Internet Source

<1 %

13

link.springer.com

Internet Source

<1 %

14

"Foundations and Practice of Security",
Springer Science and Business Media LLC,
2018

Publication

<1 %

Exclude quotes On

Exclude bibliography On

Exclude assignment template On

Exclude matches < 14 words