UNIVERSITY OF
GLOUCESTERSHIRE

Improving the Security and Performance of Ethereum Blockchain Transactions: A Decentralised Autonomous Organisations Model

Sepideh Mollajafari

University of Gloucestershire

Supervisor

Professor Kamal BECHKOUM

This dissertation is submitted for the degree of
Doctor of Philosophy

May 2024

Declaration

I hereby declare that this thesis is entirely my original work, except where otherwise acknowledged. All sources used for this research have been properly cited and acknowledged. This work has not been submitted for any other degree, or qualification, at the University of Gloucestershire, or any other institution.

Sepideh Mollajafari

May 2024

doi:10.46289/ZFHX1698

Acknowledgements

First, I would like to express my sincere gratitude and Profound respect to my supervisor, Professor Kamal Bechkoum for his unwavering support, guidance and encouragement throughout my PhD journey. I am truly grateful for Professor Bechkoum's mentorship, willingness to share his expertise and insightful feedback that have significantly enriched the quality and depth of this research and shaped my academic achievement.

Words cannot express the depth of my gratitude to my beloved parents, husband, sisters and niece. Their endless encouragement and belief in my abilities have inspired me to persevere through the completion of my doctorate.

Many thanks to my colleagues from the CyberTech at the University of Gloucestershire who provided me with invaluable insight and support throughout my research endeavour.

Last, my sincerest thanks go to the Security Analyst and Solidity developers who evaluated the proposed Genuine DAO and provided valuable feedback.

Publications

- ☑ *Mollajafari, S. and Bechkoum, K. (2023) 'Blockchain Technology and Related Security Risks: Towards a Seven-Layer Perspective and Taxonomy',* Sustainability*, 15(18), p. 13401. doi: 10.3390/su151813401.*

- ☑ *Mollajafari, S. (2022) 'Warning on Blockchain security risk!',* British Computer Society (BCS)*. Available at:* *https://www.bcs.org/articles-opinion-and-research/warning-on-Blockchain-security-risk/.*

- ☑ *Mollajafari, S. and Bechkoum, K. (2022) 'An Overview of Blockchain Technology and Security Risks: a layering perspective', in. The British Blockchain Association - 4th Blockchain International Scientific Conference. Available at: file:///C:/Users/s2115469/Downloads/34726-conference-proceedings-of-4th-Blockchain-international-scientific-conference-isc2022-2.pdf.*

Abstract

Blockchain technology has recently received a great deal of attention from industry and academia due to its apparent benefits. From the initial foundation based on cryptocurrency to the development of smart contracts, Blockchain technology continues to promise significant business benefits for various industry sectors. Notwithstanding its known benefits, and despite having some protective measures and security features, this technology still faces significant security challenges within its different abstract layers. This work focuses on the critical cybersecurity threats and vulnerabilities inherent to the different layers of the Blockchain architecture, with a view to mitigate against the associated risks.

From the perspective of architectural layering, each layer of the Blockchain has its own corresponding security issues. In this work, a seven-layer architecture is used, whereby the various components of each layer are set out, highlighting the related security risks and corresponding countermeasures. A taxonomy is then developed, that establishes the inter-relationships between the vulnerabilities and attacks in a smart contract. A specific emphasis is placed on the issues caused by centralisation within smart contracts, whereby a "one-owner" controls access, thus threatening the very decentralised nature that Blockchain is based upon. Smart contracts with centralised ownership pose major security issues and act as a single point of failure, allowing single individuals, or teams, to have complete control over the Blockchain network. To mitigate against the risks associated with centralised control, decentralised autonomous organisations (DAOs) promote a decentralised decision-making process whereby the power of decision-making is distributed and therefore preventing smart contract ownership monopoly.

The main contribution of this thesis is the development of a novel automated decentralised application, "Genuine DAO", that promises to reduce security risks and improve the performance of Blockchain networks. "Genuine DAO" achieves the reduction in security risks by enforcing automated rules that are encoded in smart contracts thus reinforcing the community-based governance and minimising the threats inherent to centralisation, which can be caused by smart contracts' owners/developers. Additionally, "Genuine DAO"

strengthens the security of the network by guarding against the threats caused by *Frontrunning* attacks.

Three further contributions emanate from this work. The first one is an improvement of the overall performance of the Blockchain network, through gas optimisation, cost reduction, and network throughput. This is achieved by using a Polygon layer 2 scaling solution built on the Ethereum network. The second one is the development of a general taxonomy that compiles the different vulnerabilities, the types of attacks, and the related countermeasures within each of the seven layers of the Blockchain. The third one stems from a deep dive into one layer of the Blockchain namely, the Contract Layer. A model application is developed depicting, in detail, the security risks within the Contract Layer, while enlisting the best practices and tools to adopt in order to mitigate against these risks. The understanding gained from delving into the details of security risks within the Contract Layer reinforced the need for developing countermeasures to alleviate the security risks and vulnerabilities inherent to one-owner control in smart contracts, which ultimately led to the main contribution of this work: Genuine DAO.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

1.1 Background

The notion of Blockchain technology was introduced by Nakamoto who published an article about cryptocurrency in 2008, and in 2009 bitcoin became the first decentralised cryptocurrency. This technology has, over the last few years, received a great deal of attention from industry and academia due to its favourable characteristics such as decentralisation of control, reliability and consistency of data and transactions, immutability and anonymity (Xiao *et al.,* 2020; Ul Hassan, Rehmani and Chen, 2020; Chen *et al.,* 2020; Lin and Liao, 2017).

Since the first generation of Blockchain, based on bitcoin, developers started to believe that a Blockchain could do more than simple currency transactions. The second generation of Blockchain introduced Ethereum, an open and decentralised platform, which enables users to develop smart contracts by using a programming language called Solidity (Chen *et al.,* 2020). With the introduction of smart contracts, Blockchain technology gained significant attention, enabling mutually distrusted users to complete data exchange, or transactions, without the need of intermediaries (Xiao *et al.,* 2020).

The key features of Blockchain technology are described in detail in Chapter 2.

Notwithstanding its benefits for numerous business domains, and despite having some protective measures and security features, the technology still faces significant security challenges and vulnerabilities specifically within smart contracts, where a notable number of issues remain unexplored. This statement is based on the detailed analysis of the existing work, which is presented in chapter 3. While blockchain promotes decentralisation, smart contracts can introduce centralisation risks. The centralisation within smart contracts contradicts the fundamental principles of blockchain, creating a critical vulnerability.

A Decentralised Autonomous Organisation (DAO) model can significantly reduce centralisation risks for smart contracts by distributing decision-making power, automating execution, aligning incentives, decentralising development, and implementing fair conflict resolution mechanisms. These features collectively enhance the security, transparency, and trustworthiness of smart contracts within blockchain ecosystems.

Despite DAO, smart contract developers, or contract owners, often have significant control over the contract's functions, potentially creating single points of failure. If the blockchain is not sufficiently decentralised, these centralisation points can be exploited, undermining the security of the entire system.

It is within this area that this research falls, as outlined in the following section.

1.2 Problem Statement and Research Aims

While decentralisation is a key concept in Blockchain, paradoxically, centralisation is one of the main vulnerabilities that raises security concerns in Blockchain transactions. These centralisation risks emanate from different layers of the Blockchain. The main area of interest of this research is the security of Blockchain transactions, with a particular focus on Ethereum smart contracts within the Contract Layer. This special attention stems from the fact that, although smart contracts are meant to be decentralised, developers can exploit the network to inject centralisation into the smart contracts. This is the case because when digital assets are in the control of developers/owners, and Blockchain is not sufficiently decentralised, the risk moves to the smart contract itself. This makes smart contracts one of the major areas of security concerns in Blockchain transactions (Chen *et al.,* 2020, Xiao *et al.,* 2020, Sai *et al.*, 2021; CertiK, 2022). A detailed examination of smart contracts' security risks within the Contract Layer can be found in Chapter 3 of this thesis. This examination reinforced the need for developing countermeasures to mitigate the centralisation security risks, which are caused by smart contracts. There is, therefore, a strong argument for further research to develop and implement robust security protocols that guard against smart contracts' vulnerabilities and ensure the true decentralisation and security of Blockchain transactions.

With the above-mentioned centralisation concerns in mind, this research aims to strengthen smart contracts resilience against potential attacks, such as front-running attack, and to minimise centralisation risks using an automated approach based on the concept of decentralised autonomous organisations (DAOs).

1.3 Research Questions

In working towards this aim, this research intends to answer the following research questions:

RQ1: What are the current security concerns in Ethereum Blockchain transactions?

RQ2: What types of vulnerabilities are inherent to smart contracts in Blockchain?

RQ3: How can a DAO-based framework enhance the decentralisation and security of smart contracts?

1.4 Research Objectives

In order to achieve the research aims, and answer the research questions, the following objectives have been identified:

RO1: To investigate current security concerns in Ethereum Blockchain transactions.

RO2: To systematically examine the various vulnerabilities in smart contracts.

RO3: To create a detailed taxonomy of security issues in Blockchain smart contracts.

RO4: To design a DAO-based framework that reinforces decentralisation and enhances smart contract security.

RO5: To develop and implement a robust methodology to test and evaluate the proposed framework.

1.5 Key Research Contributions

The main contribution of this thesis is the development of a novel automated decentralised application, “Genuine DAO”, that promises to reduce security risks and improve the performance of blockchain networks. “Genuine DAO” achieves the reduction in security risks by minimising the threats inherent to centralisation, which can be caused by smart contracts’ owners/developers, and by guarding against Front-running attacks.

Although the above is the main contribution, other significant contributions to knowledge in the Blockchain field are summarised as follows:

Before delving into the Blockchain Contract Layer, which is the main focus of this research, there was a need to develop a deeper and more comprehensive understanding of the vulnerabilities that exist within each of the Blockchain layers. For this, a seven-layer architecture is adopted leading to the development of a general taxonomy (see Figure 10) that compiles the different vulnerabilities, the types of attacks, and the related countermeasures within each of the seven layers of the Blockchain.

A model application is developed depicting, in detail, the security risks within the Contract Layer, while enlisting the best practices and tools to adopt in order to mitigate against these risks, as shown in Figure 11. The understanding gained from delving into the details of security risks within the Contract Layer reinforced the need for developing countermeasures to alleviate the security risks and vulnerabilities inherent to one-owner control in smart contracts, which ultimately led to the main contribution of this work: Genuine DAO.

This research also identified the different types of vulnerabilities and attacks across each layer of the Ethereum Blockchain and described the inter-relationships between these vulnerabilities, attacks, and their related consequences. Additionally, a systematic investigation was carried out into the existing tools and mechanisms proposed by researchers and developers to detect and prevent these vulnerabilities and attacks. The findings, summarised in Tables 2-8, detail the location of vulnerabilities or attacks within the Blockchain, the nature of these vulnerabilities/attacks, key related works by various authors, and the detection tools or preventive techniques available.

Another contribution of this work is the enhancement of the overall performance of the Blockchain network. This improvement is realised through gas optimisation, cost reduction, and increased network throughput. The enhancement is achieved by utilising a Polygon layer 2 scaling solution built on the Ethereum network.

1.6 Thesis Structure

In addition to this Introduction Chapter, this thesis is organised as follows.

Chapter 2 covers the background information and some of the Blockchain principles and key features used in this work. Such features include the architecture of Blockchain technology, the abstract layers forming the Blockchain network and key components such as cryptography, smart contracts and the Ethereum platform.

Chapter 3 describes the findings of a systematic literature review, which encompasses the different vulnerabilities and attacks associated with each layer of a seven-layer Blockchain. The review also highlights the potential consequences of these attacks and suggests countermeasures yielding a taxonomy which outlines, within each layer, the inter-relationships between the vulnerabilities, the attacks, and the corresponding potential consequences.

Chapter 4 discusses the research methodology used to reduce centralisation risks and enhance smart contracts' security. An experiment-based approach is adopted, which is supported by a mixture of qualitative and quantitative analysis. This chapter also presents the research tools and techniques, including software experiments, which were applied in this study.

Chapter 5 describes the technical architecture of the developed decentralised application, "Genuine DAO", whereby smart contracts are written in such a way to prevent one-owner control, and to minimise the risks of Front-running attacks. This chapter is dedicated to describing the design and implementation of the "Genuine DAO" framework covering development and testing aspects. In this chapter, a description is given of the detailed elements of test cases used, including how these test cases were designed and executed to assess the functionality of the framework, its performance and its security. The results of software experiments and qualitative analysis conducted during this research can be found in this chapter.

Chapter 6 discusses the key finding and provides an evaluation of the results, measured against the research aims and objectives.

Concluding notes are to be found in **Chapter 7**, including recommendations for future work and research limitations.

1.7 Code Repository

The code of the Genuine DAO application, back-end written with Solidity and front-end written in JavaScript, is available on the GitHub repository https://github.com/Sepideh-M.

Chapter 2: The Blockchain Technology and its Key Features

2.1 Introduction

This chapter covers the architecture of Blockchain technology and its key features, including a description of the different layers of the Blockchain and some of the key mechanisms and concepts used in blockchains, such as decentralisation, cryptography, the Ethereum platform and smart contracts. This description is needed to set the scene for the literature review chapter (Chapter 3), which focuses on the security issues and vulnerabilities associated with the technology.

2.2 Blockchain Technology

Blockchain is a technology which is receiving a growing attention from many researchers, scientists, and application developers. Data is stored in a transparent, shared distributed ledger, which is verified and maintained by the nodes in a decentralised network. The data in Blockchain are immutable as it is guarded by cryptography to ensure security, integrity and privacy (Ul Hassan *et al.*, 2020). This innovative tool promises a secure digital world, and offers more reliable and convenient services. As a result, many organisations, from the private and public sectors, are eagerly looking at this advanced technology to enable digital business and implement innovative applications in diverse areas such as finance, IoT, cryptocurrency, digital identity, real estate, social media, distributed cloud storage and healthcare. All of this illustrates that this advanced technology has the potential to overtake several industrial business models and change the way transactions are conducted in everyday life (Ul Hassan *et al.*, 2020). Nonetheless, and despite the fact that this advanced technology has great benefits for businesses, it is facing a large number of security challenges which is hindering its successful adoption (Wen *et al.,* 2021).

2.3 Blockchain Key Features

Decentralised distributed ledger, cryptographic algorithms, consensus algorithms are fundamental elements of Blockchain. Combining these elements, provide a reliable database for transparent and secure transactions (Raikwar, Gligoroski and Kralevska, 2019). The sections below give a brief description of these key features.

2.3.1 Decentralised Distributed Ledger

Blockchain is a decentralised and distributed digital ledger, or a shared database that stores data (blocks) and maintains the ledger in a decentralised way (Yap, Chin and Klemeš, 2023). In Blockchain Peer to Peer (P2P) network, nodes communicate directly and all of them have access to the shared ledger and keep the same copy of all valid transactions (Raikwar, Gligoroski and Kralevska, 2019). This decentralised nature of the technology and its inherent features of distribution and absence of a single point of failure makes Blockchain a technology that promises a significant enhancement of data security and transparency (Ahmadisheykhsarmast, *et al.*, 2023). The security aspect is further strengthened by other features such as cryptography.

2.3.2 Cryptography

Cryptography is one of the key features of Blockchain and is used to achieve four main security goals namely, data confidentiality, integrity, authentication and privacy. Data confidentiality and authenticity are achieved by using digital signature. Data integrity is achieved by employing hash functions (Lone and Naaz, 2020). Data privacy is achieved by using zero knowledge proof (Chi, Lu and Guan, 2023). These functions are described below.

2.3.2.1 Digital Signature

Most cryptographic systems typically rely on three key dimensions. 1) operations to convert plaintext into ciphertext. These operations determine the encryption and decryption processes. 2) the number of keys involved in cryptographic operations. 3) the algorithm employed for processing plaintext. There are many different types of encryption algorithms. Symmetric and asymmetric encryptions are the most common types of encryption. In a symmetric encryption, known as a secret key, the same key is used for encrypting and decrypting data. On the other hand, asymmetric encryption, referred to as public key, is a safer method whereby a pair of keys is used, public key and private key (Lone and Naaz, 2020). Ethereum uses Elliptic Curve Digital Signature Algorithm (ECDSA) for transaction signing and verification. Asymmetric encryption in Ethereum Blockchain is used to ensure authenticity, integrity and non-repudiation of transactions (Lone and Naaz, 2020).

Decentralised authentication and authorisation mechanisms are critical to the security of Ethereum Blockchain. Authentication is a process of validating user identities, whereas authorisation determines what operations and functions users are permitted to perform once they are successfully authenticated (Zhong *et al.,* 2021; Ghaffari *et al.,* 2020). In Ethereum Blockchain, cryptography (digital signature) is used for data origin authentication. The user creates a message called transaction and signs it digitally by its private key before broadcasting the transaction to the network. The new transaction has a signature and a public key of the user. The signing process is carried out in a client side through the user's wallet and proves that they are the rightful owner of the asset (Bashir, 2020). When a smart contract receives the transaction, Ethereum Blockchain checks that transaction signature to verify if the signature matches the public key of transaction. If it is not matched, the transaction fails. If it is matched, the address of the user, which is derived from its public key, is put aside using the "msg.sender" variable of the smart contract. In fact, a smart contract does not need any authentication itself, the Ethereum Blockchain takes care of that. The sender of the transaction is identified by the address stored inside the "msg.sender". More importantly, the private key of the user is not stored on the Blockchain and is stored in the user's wallet. If hackers managed to get into the user's wallet, only the user's private key is compromised and other users' keys remain secure. The authentication mechanism is completely different from that of a web application or a centralised system where a hack of the centralised database can compromise all users' accounts (EatTheBlocks, 2019).

The authorisation mechanism is used to establish access to resources and handles user's privileges in the Blockchain network (Zhong *et al.,* 2021). Writing the correct authorisation control is significantly important when developing a smart contract. As part of the authorisation mechanism, the visibility function is critical when users call the function. There are four levels of accessibility for each function in a smart contract: *public*, *external*, *internal*, and *private,* with the default visibility of a function being *public*. Therefore, the visibility function plays a key role in the security of a smart contract, ensuring that the level of access is limited (Adi, 2022).

2.3.2.2 Hashing Functions

Kuznetsov, *et al.*, (2024) explained a hash function as a mathematical algorithm that takes an arbitrary length input *x* and produces a fixed-length output, typically referred to as the hash value *h*, such as *H(x)=h.*

Cryptographic hash functions are used to create Merkle trees, generate Ethereum address and message digest in digital signatures thus providing efficiency, security, and integrity of large data (Bashir, 2020). Each transaction has a unique identifier called a transaction hash. Each block of the Blockchain network has its own hash as well as the hash of the previous block. The linking of blocks through the hashing functions provides immutability. The hash function algorithm used in Bitcoin is SHA-2 and its variant SHA256, whereas Ethereum uses Keccak-256, part of SHA-3, to create a chain, generate and verify digital signatures and create Ethereum accounts (Raikwar, Gligoroski and Kralevska, 2019; Apriani and Sari, 2021).

Cryptography hash functions are critical in a) ensuring data integrity through the use of digital signature (Saini *et al.,* 2022) and 2) achieving collision resistance by making it hard to find two different inputs *x* and *x′* that produce the same output, *H(x) = H(x′).*

Antonopoulos and Wood (2018) explained that the Ethereum address (account*)* is derived from a public key and uses the hashing function keccak256. The public key is derived from the private key by using a hashing algorithm called Elliptic Curve Digital Signature (ECDSA), as mentioned above. The private key is generated randomly. Ethereum software uses the underlying operating system's random number generators to generate 256 bits of entropy. Figure 2 shows the process of the Ethereum address generation (Lone and Naaz, 2020; Antonopoulos and Wood, 2018).

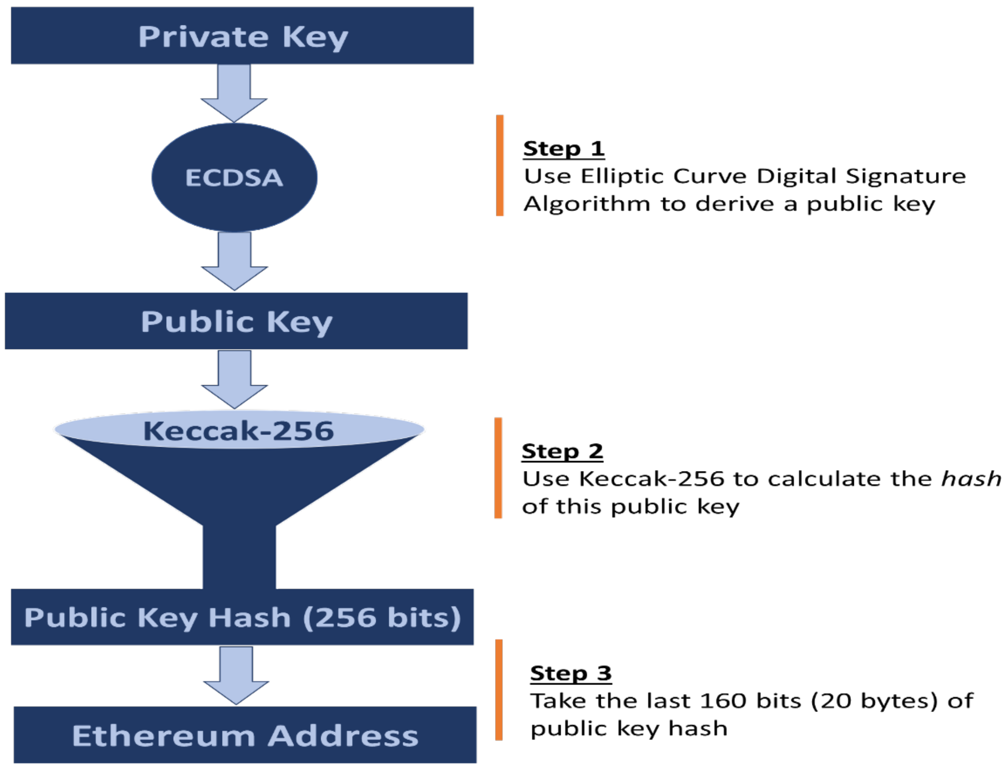

Private Key
ECDSA
Public Key
Keccak-256
Public Key Hash (256 bits)
Ethereum Address
Step 1
Use Elliptic Curve Digital Signature Algorithm to derive a public key
Step 2
Use Keccak-256 to calculate the *hash* of this public key
Step 3
Take the last 160 bits (20 bytes) of public key hash

Figure 1 - The Process of the Ethereum Address Generation (Adapted from Lone and Naaz, 2020; Antonopoulos and Wood, 2018)

2.3.2.3 Zero-Knowledge Proofs (ZKPs)

Bashir (2020) stated that ZKP is a cryptographic protocol for proving that a prover possesses a secret without revealing it to the verifier. Completeness, soundness and zero knowledge are three properties that are required in the context of ZKPs.

Completeness Property ensures that a valid statement can be proven to be true. Therefore, the prover can convince a verifier of the truth of a statement (Bashir, 2020)**.**

The Soundness Property ensures that if an assertion is false, no malicious prover would be able to convince the verifier that it is true (Bashir, 2020).

The Zero-knowledge Property prevents the verifier from learning any additional knowledge about the prover’s secret and ensures that absolutely no information will be revealed about the assertion except whether it is true or false (Chi, Lu and Guan, 2023; Bashir, 2020).

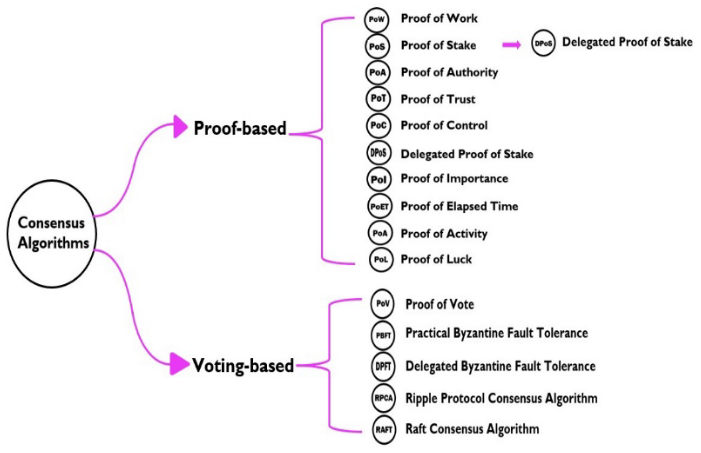
Zero-knowledge proofs are used to enhance the user’s privacy and provide secure authentication, anonymous transactions within a Blockchain system (Chi, Lu and Guan, 2023; Bashir, 2020).

2.3.3 Consensus Algorithms

Bashir (2020) highlights the fact that Blockchain is a distributed system that relies on consensus algorithms to guarantee the security and liveness of the network. Consensus algorithms are protocols that force the consensus rules to ensure all nodes in a decentralised network reach an agreement on the Blockchain data state (Antonopoulos and Wood, 2018).

Consensus algorithms are categorised into two types, proof-based and voting-based and are utilised to ensure all nodes reach an agreement on the data validity of newly generated blocks in the decentralised network (Wen *et al.,* 2021; Xiao *et al.,* 2020; Nguyenand Kim, 2018; Alsunaidi and Alhaidari, 2019; Suresh *et al.,* 2020). In Bitcoin, miners use a significant computational power and compete to solve a cryptographic puzzle (Find a nonce value) to verify transactions and add a new block. When cryptocurrency become more popular, more computers join the network, driving up the network's computing power. The greater the number of people who join the network, the harder it becomes to mine. Eventually, users start relying on Graphics Processing Units (GPUs) and then once those become insufficient, miners have to start investing in an Application-Specific Integrated Circuit (ASIC) hardware. Later miners find that joining a mining pool, combing individuals computing power, provides a more stable hashrate for them and tremendously increases the probability of validating blocks and earning rewards [Beikverdi and JooSeok 2015; Minima, 2022).

The Proof of Work (PoW) algorithm was firstly adopted for Ethereum to reach the consensus. This algorithm requires a significant amount of computational powers and other resources, such as huge electricity consumptions, to verify transactions and add a new block to the ledger. It also takes a long time (roughly 10 minutes) to achieve data consistency.
Ethereum announced in September 2022 that it moved to Proof of Stake (PoS). With the PoS mechanism, validators are responsible for block creation. In order to participate as a validator, a user must hold 32 ETH and stake them (Ethereum, 2023). Therefore, Ethereum with PoS relies on validators, not miners, to add new blocks to the chain (Elliott, 2022). Figure 3 depicts the different types of consensus algorithms.



Consensus Algorithms
Proof-based
PoW Proof of Work
PoS Proof of Stake
DPoS Delegated Proof of Stake
PoA Proof of Authority
PoT Proof of Trust
PoC Proof of Control
DPoS Delegated Proof of Stake
PoI Proof of Importance
PoET Proof of Elapsed Time
PoA Proof of Activity
PoL Proof of Luck
Voting-based
PoV Proof of Vote
PBFT Practical Byzantine Fault Tolerance
DPFT Delegated Byzantine Fault Tolerance
RPCA Ripple Protocol Consensus Algorithm
RAFT Raft Consensus Algorithm

Figure 2 - Classification of Consensus Algorithms, (Adapted from Alsunaidi and Alhaidari, 2019; Zhu et al., 2020; Chaudhry and Yousaf, 2018)

2.4 The Ethereum Blockchain Platform

There are different Blockchain platforms such as Ethereum, Hyperledger, Corda, Quorum, IBM Blockchain and many more that offer the necessary tools, protocols, and functionalities to build and deploy Blockchain-based applications. According to (Gartner, 2023), the Ethereum platform is the most used Blockchain platform. For this reason, this research uses Ethereum as a platform.

Ethereum allows developers to create and deploy smart contracts on Blockchain network and develop decentralised applications (Bashir, 2020). The core component of Ethereum is the Ethereum Virtual Machine (EVM). The EVM provides a runtime environment to handle smart contract development and execution (Antonopoulos and Wood, 2018; Marchesi *et al.,* 2020). The EVM operates as a virtual machine, similar to how a CPU executes machine code in traditional computing. It is used by developers for creating and deploying smart contracts to develop decentralised applications using, mainly, Solidity as the programming language. Before being executed, within an Ethereum environment, the Solidity source code needs to be compiled into the bytecode that EVM understands (Bashir, 2020).

There are several locations where the data is stored on Ethereum Blockchain. These include storage (persistent memory to store smart contract state variables), memory (temporary data space to store data during the execution), stack (local computations data store in the EVM to store data during contract execution) and call data (Read-only data for external function calls). The EVM is a stack-based execution machine that stores data in the memory on a stack (Marchesi *et al.,* 2020).

Bashir, (2020) explained that Ethereum operates on the peer to peer network where nodes contribute to the consensus mechanism, validate and verify transactions and contribute in order to maintain the Blockchain. The native currency in the Ethereum network is called Ether (ETH). Ether is used as "gas" to power transactions and execute smart contracts on the Ethereum network. Every operation performed on the network, such as sending Ether or interacting with a smart contract, requires a certain amount of gas, which is paid by whoever sends the transaction (Marchesi *et al.,* 2020).

To calculate the transaction fee, gas used should be multiplied by gas price. The more gas indicates a higher transaction fee. Gas price is measured in gwei. Each gwei is equal to one-billionth of an Eth (1gwei= 0.000000001 Eth or 10^{-9}). Transaction costs can be estimated using the following formula: (Bashir, 2020; Nico, 2024).

$$Transaction\ cost\ (Eth) = gasUsed\ *\ gasPrice\frac{gwei}{1000000000} * EtherPrice\ (£)$$

$$Transaction\ cost = gasUsed\ * \frac{gasBaseFee\ + gasPriorityFee}{1000000000} * EtherPrice$$

According to Bashir, (2020) the *gasPrice* is set by the transaction originator as an incentive to the validators for them to include a transaction in a block during block creation. The *gasPrice* includes the *base fee* that is a value set by networks (Ethereum Mainnet and Polygon in this research) and the *priority fee,* which is a value set by the user as an incentive to the validator to include a transaction in a block (Nico, 2024). The *base fee* may increase due to the limitation on the maximum throughput in the network per block when numerous users attempt to interact simultaneously (Baldauf, Sonnleitner and Kurz, 2023).

The *gasUsed* represents the total amount of gas that is used by the transaction during the execution (Bashir, 2020). There is a gas limit in the Ethereum system, which refers to the maximum amount of gas that the transaction originator is keen to consume on a transaction. The minimum amount of gas that is set for an operation that affects the state of the EVM such as transferring Ether between two accounts, is 21000 gas (Marchesi *et al.,* 2020). The gas is charged depending on the resources required for a particular operation and is determined by the computational complexity. Operations such as deploying smart contracts, interacting with different smart contracts, message calls, complex functions, storage operations (reading from and writing to storage in smart contracts) can consume a huge amount of gas (Bashir, 2020; Marchesi *et al.,* 2020). *EtherPrice* is provided by Ethereum exchange and will change based on the supply and demand. To calculate the current worth, the EtherPrice can be easily converted to British Pound.

Marchesi et al. (2020) and Li. (2021) argued that gas fees have been a significant challenge and have had obvious implication on the Ethereum network. There are many optimisation tools and techniques to minimise the cost of gas. The transition from Ethereum 1.0 to Ethereum 2.0 not only enhances the security, scalability and speed, it also reduces the transaction cost. However, the Ethereum gas fee remains quite high even after the transition to Ethereum 2.0 due to different factors, such as unfamiliarity of developers with smart contract and EVM (Kong *et al.,* 2022).

Marchesi et al. (2020) suggested 24 patterns with the aim of saving gas in designing and developing smart contracts. The following are the five categories with solutions that have been presented:

External transactions: a) use Proxy delegate patterns which are a set of smart contracts working together to streamline the upgrading process of smart contracts. B) For external systems requiring past event data, grant direct access to the Blockchain's Event Log, avoiding smart contracts when unnecessary.

Storage: a) limit storage by using memory for temporary data and limiting storage updates upon completing all computations. b) Packing variables and Booleans.

Saving space: a) utilise unsigned integers of 128 bits when packing variables in one slot, otherwise it is better to use uint256 variables. B) use mappings instead of arrays to manage lists of data, c) minimise on-chain data in Storage variables.

Operations: a) Limit External Calls**.** b) use internal function calls rather than public functions Whenever possible. c) make a balance with the function number with their complexity (no many small functions and not too big functions). d) limit modifiers. e) Avoid redundant operations and double checks**.**

Miscellaneous: a) freeing storage by deleting unnecessary variables when they are no longer necessary and b) employing the Solidity Optimiser.

Li. (2021) collected data and combined knowledge from existing research resources to design a list of general gas-saving best practices for enhancing developer's knowledge. Kong et al. (2022) analysed 160,000 smart contracts on the Ethereum network and found that 52.75% of contracts contained at least one gas inefficiency. They presented an approach to detect and optimise six inefficient patterns at the source code level, focusing on development issues arising from developers. They proposed that conducting gas optimisation prior to smart contract deployment could result in considerable cost savings.

The Ethereum ecosystem consists of the following components:

Cryptographic Keys and Ethereum addresses: these components represent ownership and transfer ether. As we explained earlier in section 2.3.2.2, Private Key is a randomly chosen 256-bit number that serves as the main identifier and secret piece of information for an Ethereum account. The public key is derived from the private key and it shared and used to verify digital signatures. An address is derived from the public key and used for sending and receiving Ether and interacting with smart contracts (Bashir, 2020).

Ethereum accounts: include externally owned accounts (EOAs) and contract accounts (CAs). By having accounts in Ethereum, users can interact with Blockchain and participate in decentralised applications. Each externally owned account has a unique Ethereum address derived from a cryptographic key pair. The account holder maintains control over the private key associated with the EOA, which is used to sign transactions and prove ownership. EOAs re responsible for initiating transactions, interact with smart contracts. Smart accounts are responsible for executing Smart Contracts, interacting with externally owned accounts, maintaining the state of a smart contract by storing and updating the state as transactions (Bashir, 2020).

Ethereum Clients: Ethereum clients are software such as Geth that runs on nodes to connect to the Ethereum network. it provides several functions such as validate transactions, and maintain a copy of the Blockchain (Bashir, 2020).

2.5 Smart Contracts

According to Vivar et al. (2020) a smart contract is a computer program written using a programming language, such as Solidity, that runs on a decentralised basis and the overall state of the system is stored in a Blockchain. It can be written in various high-level programming languages. Solidity is object-oriented programming language which is influenced by C++, Python, and JavaScript languages. The Solidity language is most widely used for writing smart contracts in Ethereum platforms and is compiled into a bytecode, which is then executed by the EVM.

Currently, there are several platforms that can support smart contracts such as Ethereum, Hyperledger Fabric, Corda, Stellar, Rootstock, Polkadot, and Solana (Zheng *et al.,* 2020). This work is focused on Ethereum, one of the most popular smart contract platforms.

As smart contracts are automated and deployed on the decentralised ledger, they can eliminate the need for a central entity, decrease the maintenance cost, enhance access control mechanisms, and minimise the inherent threats to centralised systems (Ghaffari *et al.,* 2021). The user who deploys smart contracts on the Ethereum Blockchain has no permission to change the smart contract. If developers want to correct a bug, the system forces them to deploy a new smart contract with a new unique address. However, there is the ownership of a smart contract which the system automatically assigns to the contract creator at the time of deployment. The address of the owner will be stored on the Blockchain during the initialisation (Hooper Solorio and Kanna, 2019; Larson, 2022). This ownership poses a threat to the concept of decentralisation on which the Blockchain technology is based.

2.6 The Architecture of Blockchain Technology

Most researchers describe the architecture as a six-layer model. Examples of six-layer models can be found in the work of Wen et al. (2021); Yang et al. (2020); Deng, Huang and Wang. (2022). Others, like Homoliak et al. (2021) and Chen et al. (2021) condense the architecture into a four-layer model (Homoliak *et al.,* 2021; Chen *et al.,* 2020). Huang et al. (2019) on the other hand, use a seven-layer architecture, adding a *physical layer* to the six-layer model (Huang *et al.,* 2019). Having reviewed the literature, this research adopts a seven-layer architecture as its conceptual framework. The rationale behind this choice is that a seven-layer architecture provides a better granularity to account for all possible security risks. In order to develop a more detailed understanding of the sources of vulnerabilities within each of the seven layers of the Blockchain, an understanding of the role and components of each layer is needed. For this, a brief description of each layer is provided below, highlighting some of the key vulnerabilities that will be discussed in more detail in Chapter 3.

2.6.1 Application Layer

This layer comprises various forms of application scenes such as programmable currency, programmable finance, and programmable society. The introduction of smart contracts, instead of humans, to execute contracts provides a great opportunity to implement Blockchain solutions for use across different applications and industries (Wen *et al.,* 2021; Ahmed and Kumar, 2019). Within this layer, threats are broad and can include internal and external attackers, malicious exchanges/service providers, malware, design and configuration (Homoliak *et al.,* 2021). Users are using decentralised, centralised exchanges and different platforms to exchange digital assets. Thus, these exchanges play an important role in the development/adoption of Blockchain. Despite of numerous benefits offered by decentralised exchanges such as better security, no middle man in the ownership and transfer of funds, control over the assets and less transaction fees, users are using centralised exchanges which come with “centralisation risks” (Annessi and Fast, 2021; Nathan Sexer ,2018). One of the risks is that users’ assets can be controlled by the exchange operator (or malicious operator), which provides full control over the funds on their servers (Annessi and Fast, 2021; Nathan

Sexer ,2018). Another risk is that a centralised exchange acts as a centralised network owner, which causes a single point of failure (Sai *et al.*, 2021). Centralised exchanges are also vulnerable to hacks through insider attacks or hardware failures (Nathan Sexer ,2018). Currently, large centralised exchanges lead centralised staking activities. Therefore, large companies will have the majority share of the network and make it more centralised (Jha, 2022). To eliminate the single point of failure, which emanates from centralisation (Homoliak *et al.,* 2021), it is important to use decentralised exchanges. However, they may contain some vulnerabilities that come from smart contracts or other features of Blockchain.

2.6.2 Contract Layer

The contract layer contains components such as script codes, smart contracts and algorithms. In order to run a smart contract, the codes should compile to the low-level bytecode that executes in the Ethereum virtual machine (EVM). Once compiled, the smart contract deploys on the Ethereum Blockchain and is identified by a unique contract address generated upon a successful creation transaction (Antonopoulos and Wood, 2018; Destefanis *et al.,* 2018). Algorithms define the mechanism for all participating nodes to interact with each other and set relative execution and data resource. When the pre-defined rules are met, the relative operation will be performed in the network (Wen *et al.,* 2021; DevCon, 2018). Through the literature review, 13 vulnerabilities/attacks have been found within this layer. This research is focused on “owner control” that poses a serious centralisation risk by enabling developers and external attackers to exploit the Blockchain through contracts’ ownerships.

2.6.3 Incentive Layer

The incentive layer is responsible for providing some rewards and incentivise as many mining and validating nodes as possible to become part of the network. This layer includes the issuing and allocating mechanisms for issuance and the distribution of rewards (Huang *et al.,* 2019; Wen *et al.,* 2021; Han *et al.,* 2023).

To ensure security and decentralisation, the Blockchain system needs a large number of honest nodes (greater than 50%) to verify and validate each transaction. Incentive mechanisms are required to motivate nodes to participate in maintaining the safety of the system (Han *et al.,* 2023). Therefore, the incentive mechanism plays a vital role in the

Blockchain system ensuring that the majority of the network is honest (Sai *et al.*, 2021). On the other hand, some researchers stated that incentive mechanisms would cause a centralisation risk. To increase their chances of mining, individual miners use a mining pool to increase the chance of getting any reward from block creation. This process leads toward a centralised point that mining power and control over incentive distribution would be in the hands of a few individuals in the Blockchain network (Han *et al.,* 2023). Furthermore, if honest nodes withdraw from being active miner, it will impact on the value of hashing power of the network. As a result, the distribution of rewards can be skewed towards a few participants (especially the small number of participants that are part of a mining pool) leading to centralisation of hashing power of mining pool, reward centralisation and control over the network (Sai *et al.*, 2021; Leonardos, Leonardos and Piliouras, 2019). This may lead to a decrease in participation due to unfair incentive distribution, to a reduced security due to centralisation in mining pools, and to an increase in the threat of selfish mining and 51% attack (Sai *et al.*, 2021, Han *et al.,* 2023).

There are some mining pools available such as PPLNS (Pay Per Last N Shares), PPS (Pay Per Share), SMPPS (Shared Maximum Pay Per Share) and PROP (Proportional) that can be used in distributing rewards based on mining pools protocol (Wen *et al.,* 2021; Beikverdi and JooSeok 2015).

2.6.4 Consensus Layer

The consensus layer contains various algorithms that are utilised to ensure all nodes reach an agreement on the data validity of newly generated blocks in the decentralised network (Wen *et al.,* 2021; Xiao *et al.,* 2020). Initially, Ethereum used the PoW algorithm, then it moved to PoS in 2022 (Ethereum, 2023). For mining process and block creation, a few individuals have been joined to mining pools to combine their computational powers and control a large portion of (hold 51%) hashrate on a Blockchain network. This process is going to greatly damage the security and decentralisation of the network, as highlighted by (Han *et al.,* 2023; Beikverdi and JooSeok 2015; Minima, 2022). On the other hand, PoS promises to provide a more energy-efficient, scalable mechanism that reduces the centralisation risk which leads to greater decentralisation (Ethereum, 2023). However, there is still the risk of centralisation because the validation of blocks is controlled by validators who hold the majority of the token (Xiao *et al.,* 2020; Mollajafari, 2022).

The fact is that, lots of ETH holders (validators) have been staking their coins through large centralised exchanges. It means that centralised entities become dominant holders and have a majority share of network. Therefore, they have a much higher probability to add new blocks to the chain and control the process of block creation (Elliott, 2022; Jha, 2022).

2.6.5 Network Layer

The network layer comprises transmission protocols, a propagation mechanism and a verification mechanism. These protocols and mechanisms are deployed using a Peer to Peer (P2P) network for data transmission and verification across the distributed nodes (Huang *al.,* 2021). It is worth emphasising the fact that there is no centralised node or hierarchical structure in a P2P network (Yang *et al.,* 2020; Huang *al.,* 2021).

Transmission protocols allow Blockchain nodes to communicate directly with each other and to synchronise data among them. Each node has the opportunity to broadcast blocks, or transactions, in a shared ledger. Transmission protocols help nodes to be aware of all the data and broadcast only valid data to the network (Huang *al.,* 2021; Essaid *et al.,* 2018; Xu, 2018; Antonopoulos and Wood, 2018).

As part of communication between nodes on a peer to peer network, a node discovery protocol is required. This protocol works based on DNS seed address that distributes the address of other active nodes on the network (Sai *et al.*, 2021). DNS itself is a week protocol and relies on centralised network. It suffers from security and privacy issues due to a weak verification mechanism. Blockchain-based DNS assists in minimising some of the security concerns. Ethereum Name Service (ENS) contains critical information which is stored on smart contracts to manage domain name ownership (Liu *et al.,* 2019). Therefore, it may be controlled, or manipulated by malicious developers or owners, and make it centralised.

2.6.6 Data Layer

This layer acts as the Blockchain data structure. A block is a collection of valid transactions in a shared ledger, made of a block header and a block body (Liang, 2020; Yang *et al.,* 2020; Wen *et al.,* 2021). The first block in Blockchain network called Genesis block and it differs from normal blocks primarily due to a unique block hash and the data it contains (Bashir, 2020). Ethereum block includes the following components such as block number, timestamp, nonce, difficulty, gas limit, gas used, parent hash, transactions (Min, 2023).

According to (Bashir, 2020; Min, 2023) the block header contains the metadata representing the most detailed components within an Ethereum block. The followings are the elements of a block header:

- **Block number**: the total number of previous blocks (Genesis block is block zero).
- **Timestamp:** the epoch Unix time, or time and date when the block was generated.
- **Difficulty**: a value that represents difficulty level of the current block.
- **Nonce**: A64-bit hash (random value) used to create a valid block.
- **State root:** contains of the Keccak256-bit hash of the root node of the state trie after the execution of transactions in the block.
- **Receipts root**: contains of the Keccak256-bit hash of the root node of the transaction receipt trie.
- **Transaction root**: contains of the Keccak256-bit hash of the root node of the transaction trie (the root hash of the Merkle tree) which represents the list of transactions in the block.
- **Gas used:** the maximum amount of gas consumed in executing transactions per block.
- **Gas limit:** a set value of gas to consume per block.
- **Beneficiary:** the 160-bit address of miners/validators who validate the block, receiving the block reward.
- **Extra data**: arbitrary data that can be stored in the header.
- **Parent hash**: the Keccak256-bit hash of the previous block's header.
- **Logs bloom:** a 256-bit bloom filter derived from the logs of all transactions included in a block.
- **Ommers hash**: the Keccak256-bit hash of the ommers (uncles) block.

Finally, the block body holds a long list of transactions and list of ommers (Bashir, 2020; Aini *et al.,* 2022; Salomon, 2023). The diagram in Figure 4 illustrates the Ethereum block structure within the data layer.

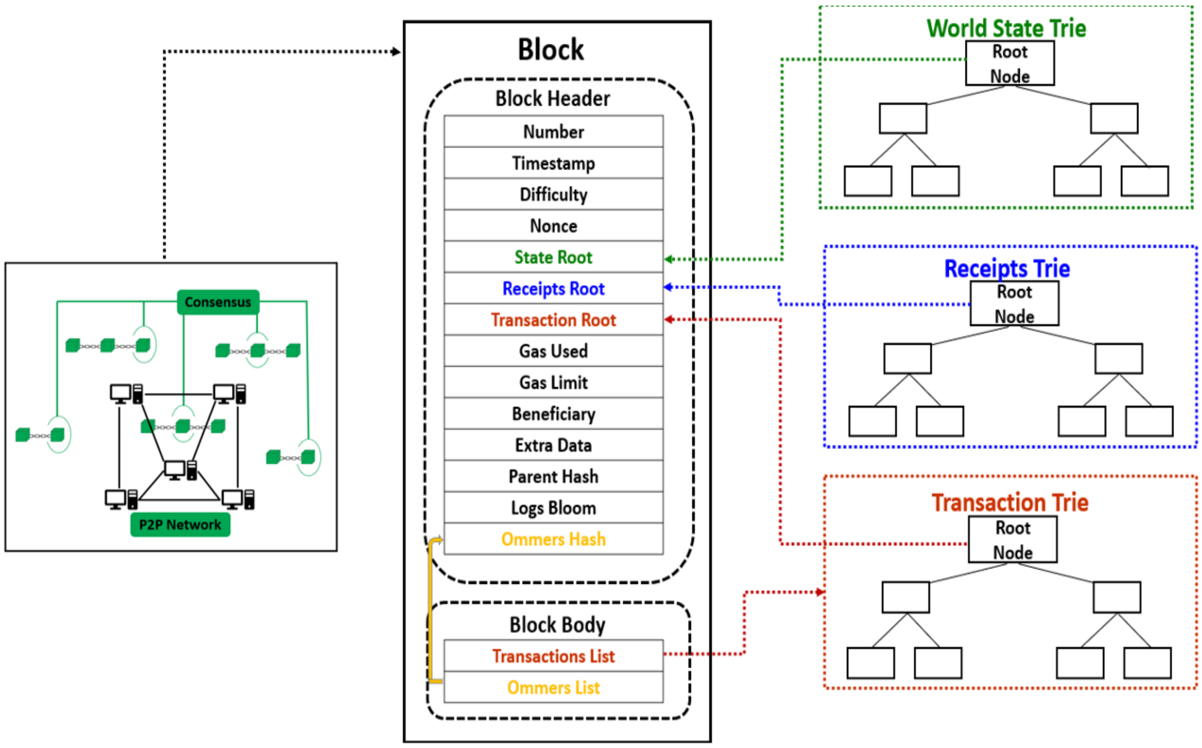


Figure 3 - Ethereum Block with a Block Header and Tries on a Peer to Peer Network (Adopted from Bashir, 2020; Aini et al., 2022; Salomon, 2023).

2.6.7 Physical Layer

The physical layer is the actual medium that transports the bits. The main components of this layer are the IoT devices, which connect to the internet and act as nodes on the Blockchain network. Smart contracts are responsible for the decentralisation of the Blockchain translating the existing contractual clauses into embedded hardware and software (Choo, Dehghantanha and Parizi, 2020). To establish a connection with a Blockchain-based system, all IoT devices need to interact with smart contracts and perform the digital signature and additional authentication processes (Yang *et al.,* 2020; Edgcombe, 2016). Integration of IoT devices with Blockchain enhances device security and data privacy.

2.7. Summary

This chapter provided an overview of Blockchain's key features, including decentralised distributed ledger, cryptography (digital signature, hashing and zero knowledge proof) and consensus algorithms. The Ethereum platform and smart contracts, being key elements of this research, have been explained. The Blockchain architecture as a seven-layer model is adopted which provides a better granularity to account for all possible security risks in each layer. The role and components of each layer have been described. A seven-layer Blockchain system architecture and the key vulnerabilities within each layer are discussed in more detail in Chapter 3.

Chapter 3: Security Analysis Within the Seven Layers of the Blockchain

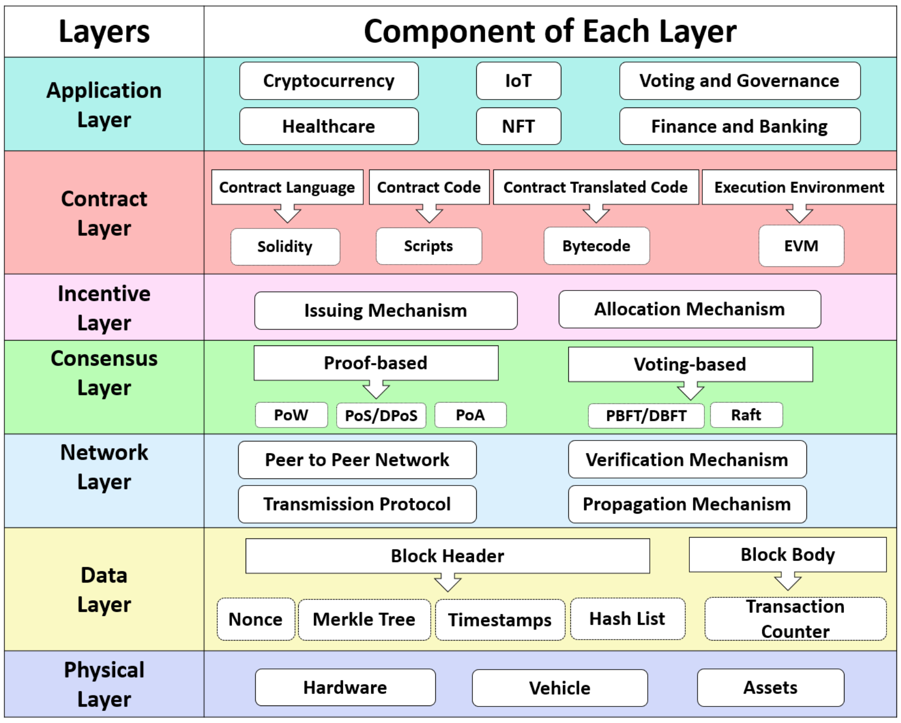
3.1 Introduction

This chapter provides a review of the different vulnerabilities and attacks associated with each layer of a seven-layer Blockchain. The potential consequences of these attacks are highlighted, yielding a taxonomy outlining, within each layer, the inter-relationships between the vulnerabilities, attacks and the corresponding potential consequences and suggested countermeasures.

3.2 A Seven-layer Blockchain

As highlighted above, the decentralised nature of Blockchain offers transparency, security, and decentralised decision-making, which can be advantageous to address security challenges. This will only work to deep dive into the architecture of Blockchain looking at each component of a seven-layer Blockchain and put in place adequate measures to counter the security vulnerabilities and threats. This section describes the initial research findings, providing a comprehensive overview of the different vulnerabilities associated with each of the seven layers.

In the current literature most Blockchain architectures are presented as comprising between four to six layers. An exception is the work in Huang *et al.,* (2019), as highlighted above. This, poses a high risk of missing the source, and therefore understanding the nature of the security threats. Having a more granular architecture enables a closer look at the components of the Blockchain, and a more detailed examination of security risks and their location within the architecture. Therefore, a more detailed architecture, comprising seven layers, was adopted as depicted in Table 1.



Layers	Component of Each Layer
Application Layer	Cryptocurrency; IoT; Voting and Governance; Healthcare; NFT; Finance and Banking
Contract Layer	Contract Language ↓ Solidity; Contract Code ↓ Scripts; Contract Translated Code ↓ Bytecode; Execution Environment ↓ EVM
Incentive Layer	Issuing Mechanism; Allocation Mechanism
Consensus Layer	Proof-based ↓ PoW, PoS/DPoS, PoA; Voting-based ↓ PBFT/DBFT, Raft
Network Layer	Peer to Peer Network; Verification Mechanism; Transmission Protocol; Propagation Mechanism
Data Layer	Block Header ↓ Nonce, Merkle Tree, Timestamps, Hash List; Block Body ↓ Transaction Counter
Physical Layer	Hardware; Vehicle; Assets

Table 1 – A Seven-layer Blockchain System Architecture, (Adapted from Wen et al., 2021; Yang et al., 2020; Homoliak et al., 2021; Deng, Huang and Wang, 2022; Huang et al., 2019; Chen et al., 2020).

3.3 An Overview of Vulnerabilities within a seven-layer Ethereum Blockchain

As Ethereum and smart contracts are not very mature, they add complexity to developing non-vulnerable smart contracts. In the following sections, Ethereum vulnerabilities and attacks are outlined based on their location. Their root causes and consequences are analysed, and the possible detection tools and preventative techniques, drawn from the literature, are discussed. Figure 5 provides a summary of attacks/vulnerabilities associated with each of the seven layers of the Ethereum Blockchain, which are described in detail in subsections 3.4 – 3.10. Existing works have been analysed and detection tools and preventive techniques listed for each of the layers. This work is detailed in Tables 2-8.

These findings are used as a basis for developing a taxonomy of the Ethereum vulnerabilities/attacks and their consequences, as discussed below. The basis of this taxonomy, is shown in Figure 5, summarising for each of the seven layers, existing work on detection tools and preventive techniques used for securing Ethereum systems. A fuller taxonomy depicting vulnerabilities, attacks and their consequences is later shown in Figure 10.

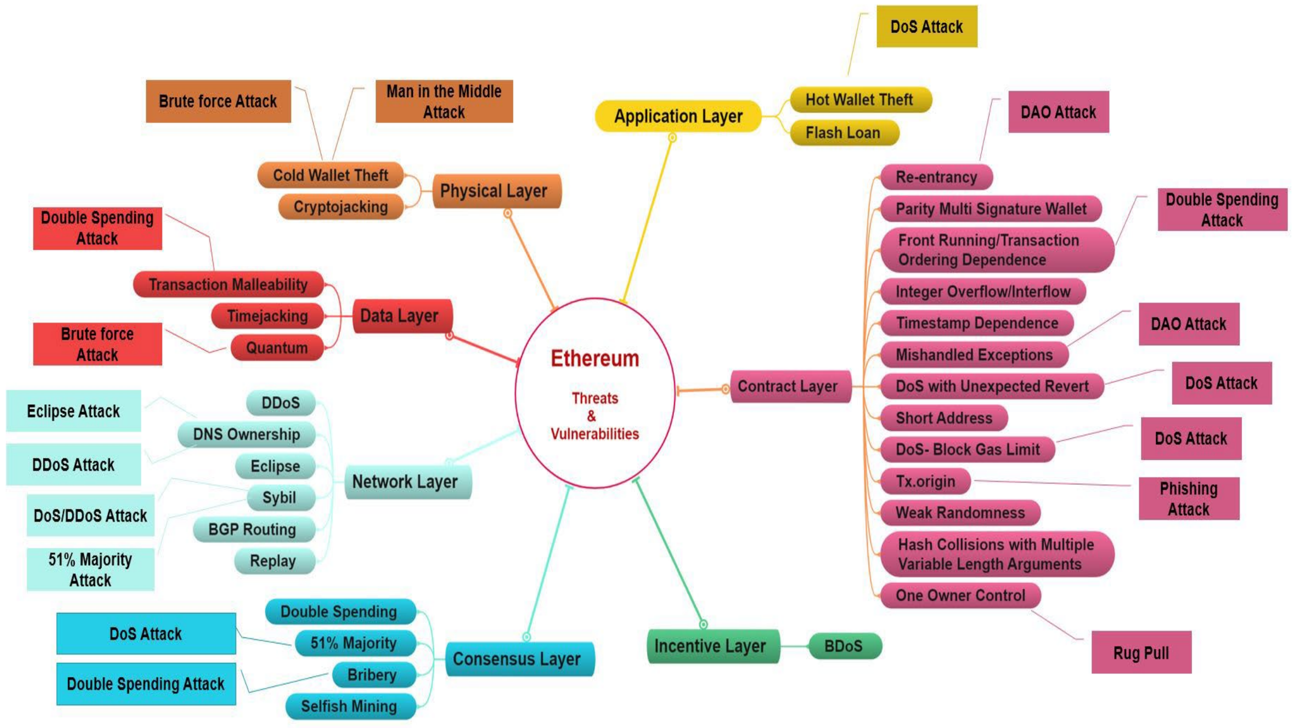


Figure 4. Vulnerabilities and Related Attacks within Each Layer of the Ethereum Blockchain

3.4 Vulnerabilities/Attacks on the Application Layer

3.4.1 Hot Wallet theft

A crypto wallet is used to store and manage the private keys. There are several crypto wallets with different security levels, such as hot wallet, cloud wallet, paper wallet and hard wallet (Rezaeighaleh and Zou, 2019). Ethereum remote clients (mobile wallets/browser wallet) are able to manage private keys, broadcast transactions and interact with smart contracts but not able to store the full Ethereum Blockchain like full node client (Antonopoulos and Wood, 2018). Since the cryptocurrency wallet is simply used for a key storage, when connecting to a transaction network, it is vulnerable for a key theft. Researchers highlighted a number of vulnerabilities in crypto wallets which cause private key leakage and loss of assets in wallets (Zamani, He and Phillips, 2020; Sung, 2021). Hackers can use different techniques to exploit a cloud server and steal and tamper with sensitive resources such as keys and transactions. Hackers also can pose DoS attack on servers. In addition, there are some reports from cryptocurrency exchanges, such as Bilaxy exchange and AscendEX, that tokens were lost from Ethereum via hot wallets (PARTZ, 2021; Thomas, 2021). Therefore, it is vital that exchanges keep most funds in cold storage.

Zamani et al. *(*2020) explained that this vulnerability can be minimised by using cold storage, offline wallets or even paper-based wallets documents and avoiding 'hot' wallets (Zamani, He and Phillips, 2020). Sung. (2021) introduced the key protocol for key exchange agreement between nodes which comprises a session key and Federated Byzantine Agreement (FBA) that protects the cryptocurrency wallet key from theft (Sung, 2021).

3.4.2 Decentralised finance (DeFi) flash loan attack

DeFi relies on smart contracts and uses automated protocols to provide financial services without intermediaries. A flash loan is uncollateralised and unsecured loan in DeFi system that allows borrowers to take loans without needing upfront collateral and then repay the loans with a single Blockchain transaction, guaranteed by a smart contract (Werapun *et al.*, 2022). DeFi poses security risks on the Ethereum Blockchain due to smart contract weaknesses and new unsecure protocols suh as MakerDAO. Flash loan attacks can lead to:

- Data leakage via phishing: attackers attempt to trick users and direct them to a fake website to access user's sensitive data such as private key (Werapun *et al.*, 2022).
- Market price manipulation: attacker borrows a large amount of digital assets via flash loan and use that funds to manipulate the price of that specific assets on a certain DeFi platform. Furthermore, malicious arbitrage or attacker create an arbitrage opportunity and manipulate token price. If greedy arbitrageurs do not have large sums of tokens in their wallet, they use flash loan service to borrow from a flash loan provider such as Aave to leverage their trading position sizes and gain more profit (Werapun *et al.*, 2022). There are a number of DeFi attacks that happened in 2020 and 2021 (Qin *et al.*, 2020; Thurman, 2021).
- Steal or redirect funds: smart contract plays crucial role in execution of transactions. Most of DeFi platforms run on Ethereum Blockchain whereas trading rules are governed by the underlying smart contract. Bugs or vulnerabilities within smart contract provide a great opportunity for attackers to steal or redirect funds (Qin *et al.*, 2020; Werapun *et al.*, 2022).

Researchers are using different analysers such as BLOCKEYE (Oracle analysis) to detect DeFi attacks on the Ethereum Blockchain (Wang *et al.,* 2021). Other researchers, Qin *et al.* (2021) proposed a framework to optimise the action parameters. The optimisations that enhance the ROI (return on investment*)* of loan-based attack (Qin *et al.*, 2020). Furthermore, Werapun *et al*. (2022) conducted the Flash loan Attack Analysis (FAA) framework to analyse DeFi attack based on different factors such as flash loan sizes, the adjustable collateral ratio, and market fluctuation (Werapun *et al.*, 2022).

Vulnerabilities/ Attacks Location	**Typical Vulnerabilities/Attacks**	**Authors Of key Works**	**Detection Tools/ Preventive Techniques**
Application Layer	Hot wallet theft	Zamani et al. *(*2020) Sung. (2021)	- Recommended use of cold storage, offline wallets or paper-based wallets. - Wallet key protocol using session key & Federated Byzantine Agreement (FBA) for the key-exchange agreement among users.
	Flash loan/DeFi	Qin et al. *(*2022) Wang et al. *(*2021) Werapun et al. (2022)	- Proposed a framework to optimise the action parameters. - BLOCKEYE to detect DeFi attacks. - Flash loan Attack Analysis (FAA) framework.

Table 2 - Current Work on Vulnerabilities/Attacks and Related Counter-measures on the Application Layer

3.5 Vulnerabilities/Attacks on Contract Layer

3.5.1 Re-entrancy Vulnerability

One of the features of Ethereum smart contracts is their ability to call and utilise code from other external contracts" (Antonopoulos and Wood, 2018 p.173). The attack happens when attackers create a contract at an external address which contains malicious code in the *fallback* function. As a result, attackers would be able to have control of this vulnerable contract and call back into the original function, invoke the same function again continually before the state has been updated. As a consequence, attackers can drain the contract's funds and the honest accounts lose Ether. DAO is a sample of re-entrancy attack on Ethereum smart contract which occurred in 2016 (Antonopoulos and Wood, 2018; Hooper Solorio and Kanna, 2019; Chen *et al.,* 2020).

The most vulnerable built-in functions contain *transfer*(), *call*(), *send*(). Between these three functions, *call* function is more vulnerable (Shahda, 2019). It is a good practice to identify vulnerable functions and monitor the operations that change the state variables of the smart contract. Furthermore, Antonopoulos and Wood. (2018) suggested to limit calls to external contracts or make it the last operation during the code execution as well as using Mutex (Antonopoulos and Wood, 2018). These solutions help to block malicious operations in the transaction. They inform the smart contract owner by storing the attacker's address in the contract and avoid re-entrant calls (Antonopoulos and Wood, 2018; Alkhalifah *et al.*, 2021). Researchers (Alkhalifah *et al.*, 2021; Khan and Siami, 2020; Feng, Torlak and Bodik, 2019) suggested various security analysis tools such as static (Oyente, Teether, Gasper, Vandal, Securify, smartcheck, Zeus), others, Khan and Siami (2020) proposed dynamic tolls such as Vultron, Sereum, Regaurd. (Khan and Siami, 2020, Feng, Torlak and Bodik, 2019) presented Fuzzing tool (ContractFuzzer). In addition, taint analysis and symbolic execution (OSIRIS, EasyFlow, SmartScopy) have been discussed by (Khan and Siami, 2020; Feng, Torlak and Bodik, 2019). Fang *et al.* (2021) proposed dynamic path profiling solution (Jyane). Furthermore, Sereum (Secure Ethereum) suggested to perform run-time monitoring of smart contract execution (Khan and Siami, 2020).

3.5.2 Parity Multi-Signature Wallet

This is a wallet that is used to manage crypto assets by users. The vital data such as user's personal information and daily withdrawal limits are stored on wallets and users should have multiple signatures, or multiple private keys, to own a multi-signature wallet to withdraw crypto assets from the wallet (Praitheeshan *et al.* 2019).

Since parity multi signature wallet depends on the public library, the centralised setup of this weak library coupled with the non-restricted calls to the external wallet library functions made the parity multi signature wallet a target for attacks. Parity multi signature wallet was hacked twice in 2017, which caused loss of ether for around $31m (Praitheeshan *et al.* 2019; Chen *et al.,* 2020).

In the case of Parity, some of the essential functions and contract logic, such as withdraw function, are implemented in a public library. In this library, functions have visibility specifiers that regulate how a function can be called. With developer’s mistake not to make the functions private or not forcing the function to be callable only within the contract itself, all public functions are callable by everyone including “initwallet” that kill the owner’s right and give the ownership right of an important contract to the attackers to take possession of its ether. The vulnerable parity multi signature wallet was divided into two contracts library, contract called “WalletLibrary and an actual “Wallet” contract to decrease the size of each wallet and save gas (Antonopoulos and Wood, 2018; Praitheeshan *et al.* 2019; Chen *et al.,* 2020).

WalletLibrary contract contains functions (*initWallet*, *changeOwner* (with external visibility), payable and withdraw (with external visibility) that allow anyone to deposit money into the wallet, but only the owner can withdraw its funds or change the owner of the wallet. However, Wallet contract (malicious contract) used delegatecall to call whatever function it provides**.** When Delegatecall is used, the code is executed within the context of the caller which allows a client contract to *delegate* the responsibility of handling a call to another contract. Consequently, an attacker would be able to call any function as long as s/he provides the signature of the function, and then changing the owner of the Wallet and withdraw funds. Malicious contracts can receive ether but will also be able to freeze any ether and stop transferring it to other accounts. This vulnerability has led to parity wallet attack (Wang *et al.,* 2020; Breidenbach *et al.,* 2017).

A good practice for Solidity developers is to adopt the private modifier by default. This will restrict the access for all contract functions and specify the visibility of all functions in a contract and have proper access control of functions even if they are intentionally public. Other measures to consider consists of adopting the private modifier by default (Praitheeshan *et al.* 2019), avoiding using “delegateCall” as a catch-all forwarding mechanism (Goldberg, 2018), build stateless libraries (Chen *et al.,* 2020), using static security analysis tools such as

Oyente to detect bugs (Vivar *et al.,* 2020) and using verification tool such as Artemis (Wang *et al.,* 2020).

3.5.3 Front Running/ Transaction-Ordering Dependence

Transaction ordering is a race condition attack whereby malicious nodes increase the transaction gas price and try to select and execute own transactions first (Hooper Solorio and Kanna, 2019). In Ethereum, miners can use their power to choose transactions and order them based on the highest gas price to get more profit and pose frontrunning attacks (Antonopoulos and Wood, 2018).

When a transaction broadcast to the Ethereum network, it goes to the Mempool. Then Miners/validators choose the transaction, use consensus algorithm to mine/validate a block. In this type of attack, malicious nodes observe transactions and all transaction details that are visible in the Mempool. Attackers will then be able to control the order of transactions, they will select their own transactions and frontrun orders with higher transaction fees to ensure they are mined/validated and it is beneficial for them. As a result, high fees paid for priority transaction ordering poses a security risk, including double spending attack (Daian *et al.*, 2019; Antonopoulos and Wood, 2018).

To mitigate against this vulnerability, it is better to minimise the miner's power to arbitrarily select transactions and put them in a queue (Eskandari, Moosavi and Clark, 2019). The other option is to use a cryptographic commit-reveal scheme to hide transaction details, which are visible on the Mempool (Eskandari, Moosavi and Clark, 2019). Finally, developers can enforce rules such as first in first out (FIFO) to reduce this vulnerability in the network (Praitheeshan *et al.* 2019; Najafi, 2020). Researchers suggested to monitor nodes' behaviour on networks by using Intrusion detection systems (IDS) and anomaly detection systems (ADS) and use static security analysis tools such as Oyente, Securify, Mythril to detect relevant vulnerabilities (Varun, Palanisamy and Sural, 2022; Praitheeshan *et al.* 2019, Mense and Flatscher, 2018).

3.5.4 Integer Overflow and Underflow

Each integer variable has a certain range of number of bits. The range of numbers that can be represented is limited. Both Solidity and EVM support up to 256 bits. The Integer Overflow and Underflow vulnerability happens when the number gets incremented higher than the maximum value or below the minimum value, respectively (Antonopoulos and Wood, 2018; Ma *et al.*, 2019). In 2018, BedToken faced integer overflow attack which cause of transferring a huge of amount to malicious accounts (Gao *et al.,* 2019).

There are some solutions to avoid under/overflow vulnerabilities which include: using the "SafeMath" library that can handle arithmetic calculation that are offered by OpenZeppelin (Ma *et al.*, 2019), checking that the output of math is valid (Ma *et al.*, 2019), creating dedicated mathematical libraries instead of using the standard operators for addition, subtraction and multiplication (Ma *et al.*, 2019), using smart contract analysis automated tools to detect the vulnerabilities from source codes such as static (Oyente, Zeus) (Praitheeshan *et al.* 2019) and dynamic (Vultron) (Khan and Siami, 2020). Furthermore, other researchers suggested to use taint analysis and symbolic execution such as OSIRIS, EasyFlow (Khan and Siami, 2020; Gao *et al.,* 2019).

3.5.5 Timestamp dependence

The block timestamp is a primary condition to run critical operations. In an Ethereum network, miners have the ability to process transactions and adjust transaction timestamps just for a few seconds, lock funds for period of times and entirely modify the output of the contract (Praitheeshan *et al.* 2019). The timestamp is usually set to the system time of miner's computer. When a block is mined successfully, the miner has to provide the timestamp for the block. The miner will check the timestamp of a new block after mining and carry out the verification process to make sure that the timestamp of the new block is larger than the timestamp of the last block and that the local machine timestamp is not greater than 900 seconds (Praitheeshan *et al.* 2019). The vulnerability happens in Ethereum when malicious miners can adjust the timestamp of a new block slightly to manipulate the outcome of timestamp dependent smart contracts (Praitheeshan *et al.* 2019; Antonopoulos and Wood, 2018; Hooper Solorio and Kanna, 2019). This vulnerability can increase the probability of frontrunning attacks (Antonopoulos and Wood, 2018).

Another vulnerability, which is similar to timestamp dependency, is the block number dependency, whereby the block number can be manipulated while it is used as part of critical operations in a smart contract (Jiang, Liu and Chan, 2018). This vulnerability can be avoided by not using block timestamps and block number in contract (Antonopoulos and Wood, 2018) or simply follow the 15 second rule (Hooper Solorio and Kanna, 2019). Developers should not rely on block.timestamp or blockhash as a source of randomness and it is a good practice to follow quality assurance test cases carefully before deploying a smart contract (Hooper Solorio and Kanna, 2019). Furthermore, the latest version of Solidity complier alerts developers of this vulnerability and uses state-reverting exceptions to handle errors (Praitheeshan *et al.* 2019). Other researchers suggested to use static security analysis tools such as static Oyente, Remix, Mythril, SmartCheck, Zeus (Praitheeshan *et al.* 2019; Khan and Siami, 2020), Fuzzing tool such as ContractFuzzer (Jiang, Liu and Chan, 2018) and use of SmartScopy as an attack synthesiser (Feng, Torlak and Bodik, 2019).

3.5.6 Mishandled exceptions

This Solidity vulnerability is known by other names in different literature, such as “Unchecked send”, “Unchecked External Call”, and “Exception Disorders” (Khan and Siami, 2020). An Ethereum smart contract performs an external call by using “*call*”, “*transfer*” and “*send*” functions to fulfil the required functionalities. The exception handling is based on the execution of callee contracts and the interaction between contracts (Khan and Siami, 2020; Chen *et al.,* 2020). Therefore, it is important how a function is called and how exceptions are handled. Out-of-gas exception is one of the famous exceptions in the Ethereum. If an exception occurs in the callee, it may or may not propagate to the caller. The calling transaction will therefore terminate entirely and revert the state and all gas is lost (Praitheeshan *et al.* 2019; Khan and Siami, 2020; Mosakheil, 2018).

Huang *et al.* (2019) and Ma *et al.* (2019) stated that a mishandled exception may cause Denial of Service (DoS) attack on the on-going contract (Huang *et al.,* 2019; Ma *et al.*, 2019). As the key problem is related to unchecked send errors, it is a good practice to handle the error manually in the caller statement to prevent attackers from executing malicious codes into the contract (Praitheeshan *et al.* 2019). Researchers suggested to use static security analysis tools such as Oyente, Remix, Mythril, SmartCheck, Securify, GasFuzzer to detect this vulnerability (Praitheeshan *et al.* 2019; Khan and Siami, 2020).

3.5.7 DoS with Unexpected Revert

This issue appears when a transaction is reverted due to improper handling of an incomplete transaction (Samreen and Alalfi, 2021). When ether is sent to a contract, the *fallback* function, or other functions, should execute. If the execution of the caller contract fails, the contract's *fallback* function only performs the *revert*() function which can disrupt the execution of the caller contract and cause a DoS state in the caller contract (Ma *et al.*, 2019; Tikhomirov *et al.*, 2018).

There are some techniques that prevent DoS attacks through transaction revert. These include using the withdraw design pattern, which places the responsibility of claiming and withdrawing funds on the users, or making the recipient pull funds out rather than the sender using push to send out funds (Ma *et al.*, 2019; Tikhomirov *et al.*, 2018). Another countermeasure is to isolate *if /for* statements with an external function call in the condition (Samreen and Alalfi, 2021). Samreen and Alalfi, (2021) proposed a framework called SmartScan that combines static and dynamic analysis to identify vulnerable pattern and detect DoS Unexpected revert vulnerability (Samreen and Alalfi, 2021).

3.5.8 Short Address – Parameter Attack

A weakness of the EVM is causing short address vulnerability which happens when a contract receives encoded parameters that are shorter than the expected parameter length. If EVM detects an underflow, it adds a zero to the end of the encoded parameters to make up the expected length (256-bits). A malicious user can take advantage of this vulnerability by removing the last zero from the ether (Antonopoulos and Wood, 2018; Sayeed, Marco-Gisbert and Caira, 2020). This can be mitigated by checking the length of a transaction's input and validating all input parameters in the external application before sending them to the network. Furthermore, as padding only happens at the end, parameter ordering in smart contract can minimise this issue (Antonopoulos and Wood, 2018). Researchers suggested to use a detecting system such as SmartScopy as an attack synthesiser to can automatically synthesize adversarial contracts to protect smart contract (Wen *et al.,* 2021; Feng, Torlak and Bodik, 2019). Others recommended to use SmarCheck as static security analysis tool (Vivar *et* *al.,* 2020) and Etherolic and SoliAudit as dynamic analysis tools to detect this vulnerability (Kushwaha *et al.,* 2022).

3.5.9 Denial of Service -Block Gas Limit

As mentioned earlier, Solidity uses *send*(), *transfer*() and *call*() functions to transfer ether to Externally owned accounts (EOAs) or between smart contracts and a contract would receive Ether by executing either the *fallback* or *receive* function *fallback*() external payable or receive() external payable). The *payable* modifier used in solidity to ensure that function can send and receive Ether (Samreen and Alalfi, 2021). EVM allocates gas at the start of execution. Each block in the Ethereum has an upper limit on the amount of gas that can be spent for computation. The gas limit per execution is 2300 and both send and transfer functions forward 2300 gas to the receiving contract to complete operation. The block gas limit prevents the security risk involved in executing expensive state changing code in the *fallback* function of the contract receiving the ether. However, if the gas usage of a transaction exceeds this limit, the transaction will collapse, which may lead to a DoS attack (Samreen and Alalfi, 2021). Nonetheless, there is no gas limit associated with the "call" function, making it more vulnerable (Samreen and Alalfi, 2021).

One way to counter this vulnerability is not to use loops over data structures (Chen *et al.,* 2020, keep track of the loop if it is necessary to use or Split the loop over multiple transactions to alleviate the risk of an unbounded loop (Chen *et al.,* 2020; Ghaleb, Rubin and Pattabiraman,2022). Ghaleb *et al*. (2022) suggested to implement access control to restrict the call of public functions to only the contract's owner or specific addresses. It is important to use function modifier to check the condition before function execution. They also proposed a static analyser named eTainter to detect this this vulnerability based on taint tracking in the bytecode of smart contract (Ghaleb, Rubin and Pattabiraman,2022). In addition, Grech et al. (2018) designed a static analysis technique called MadMax to automatically detect gas-focused vulnerabilities (Grech *et al.,* 2018).

3.5.10 Tx.origin

Tx.origin is a global variable on Solidity which returns the address of account that sent the call or transaction. Using tx.origin variable for authentication, makes the smart contract vulnerable to phishing attacks (Antonopoulos and Wood, 2018). A malicious contract can trick the victim by sending Ether, when the victim sends transaction to a malicious contract, it will invoke the "*fallback*" function and call the "*withdraw*" function of the phishable contract and transfer all the funds belonging to another address to itself through wallet (Antonopoulos and Wood, 2018).

Researchers argued strongly against using tx.origin for authentication and using msg.sender instead (Antonopoulos and Wood, 2018; Chen *et al.,* 2020). Another way is to use tx.origin == msg.sender which returns the user's contract address instead of the original address of the owner, thus preventing the external contract calls to the current contract (Antonopoulos and Wood, 2018). Tikhomirov *et al.* (2018) suggested to use a static security analysis tool such as SmartCheck to detect relevant bugs/vulnerabilities (Tikhomirov *et al.*, 2018).

3.5.11 Weak Randomness

Blockchain uses randomness to process cryptographical task (Bouichou, Mezroui and Oualkadi, 2020). Ethereum produces 256 random bits by using the underlying operating system's random number generator to create keys. Most of the Ethereum contracts are open source and variables are public on Blockchain. Therefore, it is vital to find a secure source of entropy or randomness to create keys otherwise attackers/malicious miners can easily predict the generated random number (Huang *et al.,* 2019). For example, malicious miners can control block.timestamp, block.difficulty, blockhash and block.number (Swcregistry, 2020). As randomness plays vital role in many real-world contracts such as gambling, gaming and when using proof of stake algorithm. A liable random number can be applied in proof-of-stake protocols for randomly select the miner who gets to add the next block or randomly choosing a subset of members for decentralise autonomous organisation to vote for every decision that need to make (Chatterjee, Goharshady and Pourdamghani, 2019).

Several techniques/ approaches have been used to generate pseudorandom numbers that can be used in Ethereum smart contract such as using block hash/timestamp as a seed, relying on off-chain resource like an oracle, using Commitment Schemes (two steps approach), RANDAO acts as a library (Chatterjee, Goharshady and Pourdamghani, 2019). Some of these methods are vulnerable due to trusting either untrustworthy owner of oracle (an external provider) or miners and incentive for the participants to submit random numbers. These issues can lead to centralisation risk (Chatterjee, Goharshady and Pourdamghani, 2019). Thus, reliable random data generation method is very important to limit prediction of random number (Chatterjee, Goharshady and Pourdamghani, 2019). Chatterjee *et al*. (2019) designed a well-incentivised and unmanipulable approach which provides a trustworthy source of randomness that is not rely on malicious miners or off-chain oracles. Amiet. (2021) suggested to use a secure random number generator for smart contracts such as RANDAO (Amiet, 2021).

3.5.12 Hash Collisions with Multiple Variable Length Arguments

Hash Collision happens if two separate input strings of a hash function produce the same hash output (Swcregistry, 2020). Data is encoded according to its type and Solidity provides some global functions to encode various data types. Application Binary Interface encoding functions (ABI) can be used to interact with contracts and the external contract call on Ethereum (Chittoda, 2019). abi.encodePacked() function is non-standard packed mode that performs packed encoding of the given arguments and returns the packed encoding of the data as bytes (Chittoda, 2019; Solidity Team, 2023). This function can lead to hash collision in specific situation whereas different parameters return the same value/encoding. Since the return values are the same, the signature will still match, making the attacker an admin (Swcregistry, 2020; Zipfel, 2020). In a signature verification situation, an adversary can exploit this by adjusting the position of elements in a previous function call to effectively bypass authorisation (Swcregistry, 2020).

There are different methods to prevent this vulnerability. The first option is to ensure that a matching signature cannot be achieved using different parameters. To do so, avoid using abi.encodePacked()and alternatively use abi.encode() instead (Swcregistry, 2020).

3.5.13 One Owner control - Centralisation

As previously stated, smart contracts can be written in various programming languages, such as Solidity for Ethereum, and deploy on the Blockchain. The combination of vulnerabilities in both Ethereum and Solidity programming language poses security challenges for the security checks in smart contracts development (Praitheeshan *et al.* 2019). One of the main challenges within Ethereum Blockchain is centralisation/one owner control. The following will explain some key vulnerabilities, which may overlook the significance of addressing centralisation risks during development.

Solidity is not designed with a permission-based security model in mind (Ghaleb, Rubin and Pattabiraman, 2023). Lack of stable security mechanism such as access control makes smart contracts vulnerable (Dai *et al.,* 2019). Therefore, smart contract developers implement access control checks based on their judgment and in an adhoc manner, which results in several vulnerabilities, called access control vulnerabilities/bugs (Ghaleb, Rubin and Pattabiraman, 2023).

The decentralised distributed nature of Blockchain enhances data privacy by removing third parties and solves the single point of failure issue. Using Blockchain technology provides opportunity to minimise centralised access control systems (Rouhani and Deters, 2019).
Access control is a security mechanism which is significantly important as part of security in smart contracts and all applications. This mechanism is used to restrict access to certain administrative functions, including who can view, or use, resources (Achour, Ayed and Idoudi, 2021). In traditional network applications, most of the access control solutions are deployed on centralised systems whereby a central entity decides the user's eligibility to access system functions, depending on pre-specified rules. These access control procedures cause low scalability, low fault tolerance and a lack of automation (Ghaffari *et al.,* 2021).

According to Praitheeshan et al. (2019), smart contracts act as autonomous agents in critical decentralised applications. Therefore, Blockchain, as a decentralised distributed ledger with smart contracts functionality can be a game changing technology in access control. Smart contracts can enforce access control rules through the use of cryptographic signatures (public key and private key) and decide who can execute specific functions, and how the eligible nodes can access specific transactions or data. however, smart contracts can be tailored based on the reequipments and develop in a way that owners of the data, access critical functions, perform sensitive operations and pose security risks (OpenZeppelin, 2022).

As we explained in section 2.5, the ownership of a smart contract, automatically will be assigned to the contract creator by the system during deployment. The address of the owner/contract creator will be stored on the Blockchain during the initialisation (Hooper Solorio and Kanna, 2019; Larson, 2022).

3.5.13.1 Current Methods to Implement Access Control

- *Implement Access Control by Developer*

This ownership of smart contracts is one of the most common forms of access control (OpenZeppelin, 2022). The owner of a smart contract can set up administrative privileges and defines access control rules (OpenZeppelin, 2022). Smart contract ownership can be transferred to different addresses through the use of smart contract functions, such as transferOwnership that transfer ownership to a new account and renounceOwnership, which is responsible to renounce the ownership right of the owner and transfer it to address (0) (Pierro and Tonelli, 2021; Mou, Coblenz and Aldrich, 2021). In order to provide access control to the owner of contract, developers use a function modifier, which allows them to apply a specific logic to any function that has this modifier. This modifier, called "onlyOwner", is a common access control mechanism that can be used to restrict access to certain functions or variables to the contract owner. It is also used to check if the address that calls the function is equal to the owner's address. Therefore, this makes the smart contract such that only the owner can execute this function and prevent non-owners from accessing specific functions on the smart contract (Hooper Solorio and Kanna, 2019, Larson, 2022).

This "owner control" means that the contract which is stored on the Blockchain can be accessed by any user provided they have the appropriate permissions. For example, if a smart contract needs an owner, an administrator, or another privileged user, this way of controlling access will be explicitly implemented by the developer (Ivanov and Yan, 2022). With having admin permission, the admin user would have been able to perform certain actions within a smart contract that are not available to other users such as modifying or upgrading the contract, changing its parameters or rules, adding or removing users (OpenZeppelin, 2022).

- *Implementing Access Control to a Contract Owner*

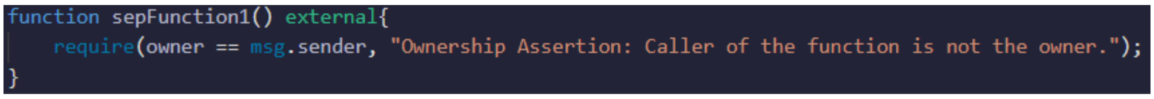
Step 1: developer creates address variable, private and call it owner. This is state variable which is stored inside the Blockchain (Larson, 2022).

```
address private owner;
```

Step 2: Creates a constructor and gives the initial value to the owner address variable. Constructor will be executed when a smart contract is deployed. Basically, we assign the address of sender of a transaction to the owner. In fact, this address is the address of creator/owner who deployed smart contract on Blockchain (Larson, 2022).

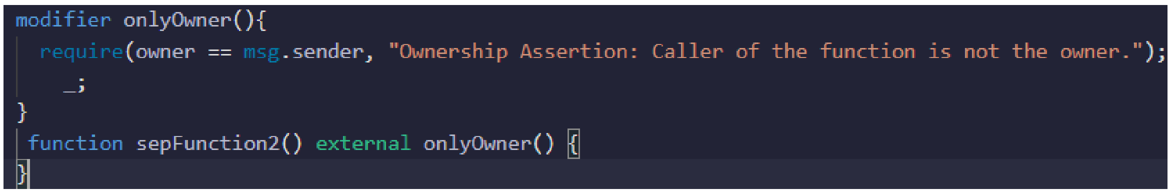
```
constructor() {
   owner = msg.sender;
}
```

Step 3: Creates a function with external view and add access control on it. Use a require statement to check if the "msg.sender" is the owner if contract. If the *require* statement is passed, it will execute and owner can call "*sepFunction1*" otherwise transaction will revert and displays an error message (EatTheBlocks, 2019).



```
function sepFunction1() external{
    require(owner == msg.sender, "Ownership Assertion: Caller of the function is not the owner.");
}
```

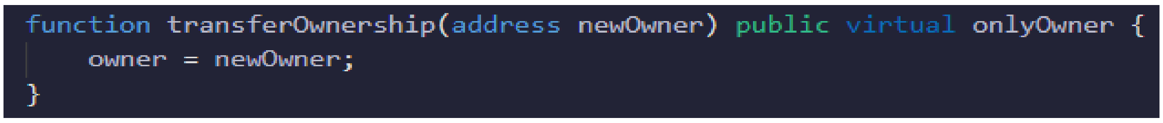
In order to prevent repeating the required statement for each function, the function modifier will be used. The developer defines a modifier called "onlyOwner" to restrict the use of function to the owner. "onlyOwner" modifier attached to the function which allows the owner of contract to call "*sepFunction2"* (Mou, Coblenz and Aldrich, 2021). Then use the *required* statement and also add underscore placeholder inside a modifier to specify when the function should be executed (Larson, 2022; EatTheBlocks, 2019).



```
modifier onlyOwner(){
  require(owner == msg.sender, "Ownership Assertion: Caller of the function is not the owner.");
    _;
}
 function sepFunction2() external onlyOwner() {
}
```

As described previously, owner of contract can be transferred to different addresses through the use of smart contract functions, such as transferOwnership and renounceOwnership (Pierro and Tonelli, 2021; Mou, Coblenz and Aldrich, 2021).

Mou *et al.,* 2021 stated that the previous owner of contract can reclaim the ownership unexpectedly due to access control bugs and unsecure smart contract configuration. These researchers proposed an analysis tool called AccessLockDetector to identify a smart contract with access control bugs (Mou, Coblenz and Aldrich, 2021). Using *renounceOwnership* function and transfer the ownership to address (0) would minimise the centralisation risk. However, the owner/developer can regain ownership after calling *renounceOwnership* (Solidgroup, 2021).



```
function transferOwnership(address newOwner) public virtual onlyOwner {
    owner = newOwner;
}
```

With the access control in the "hands of the owner/developer" they would be able to access critical functions, perform sensitive operations such as 'moderating smart contract', 'minting tokens', 'burning tokens, 'transferring ownership', 'setting any address as validator', voting on proposals, freezing funds and many other operations (OpenZeppelin, 2022; Code4rena, 2022). Access to these critical operations poses some security risks caused by the centralised ownership of the smart contract. The kind of risks and vulnerabilities that can be introduced by this centralised ownership include the possibility of the owner acting maliciously or making errors that compromise the contract's integrity. More details are given below.

- *Implementing Access Control Using* Openzeppelin

OpenZeppelin is an open-source framework for Ethereum Blockchain. It offers a set of tested and audited reusable code to create a secure smart contract. Using this library helps developers to save time and avoid code repetitions in creating Blockchain-based projects (OpenZeppelin, 2023). However, it can pose security risks in a Blockchain system through inexperienced developers who not fully understand the concept of decentralised applications. Inheriting contracts from OpenZeppelin can introduce vulnerabilities if developers use code without proper understanding, reviewing and testing. OpenZeppelin code is available on GitHub repository with different directories. "Access" directory includes different "AccessControl" contracts to provide a role-based access control mechanism and "Ownable" contract with a singer owner and the contract ownership (OpenZeppelin, 2023). In this project, the "Ownable" contract which is an access control contract imports from OpenZeppelin to offer ownership functionality.

3.5.13.2. Risks of Contract Ownership

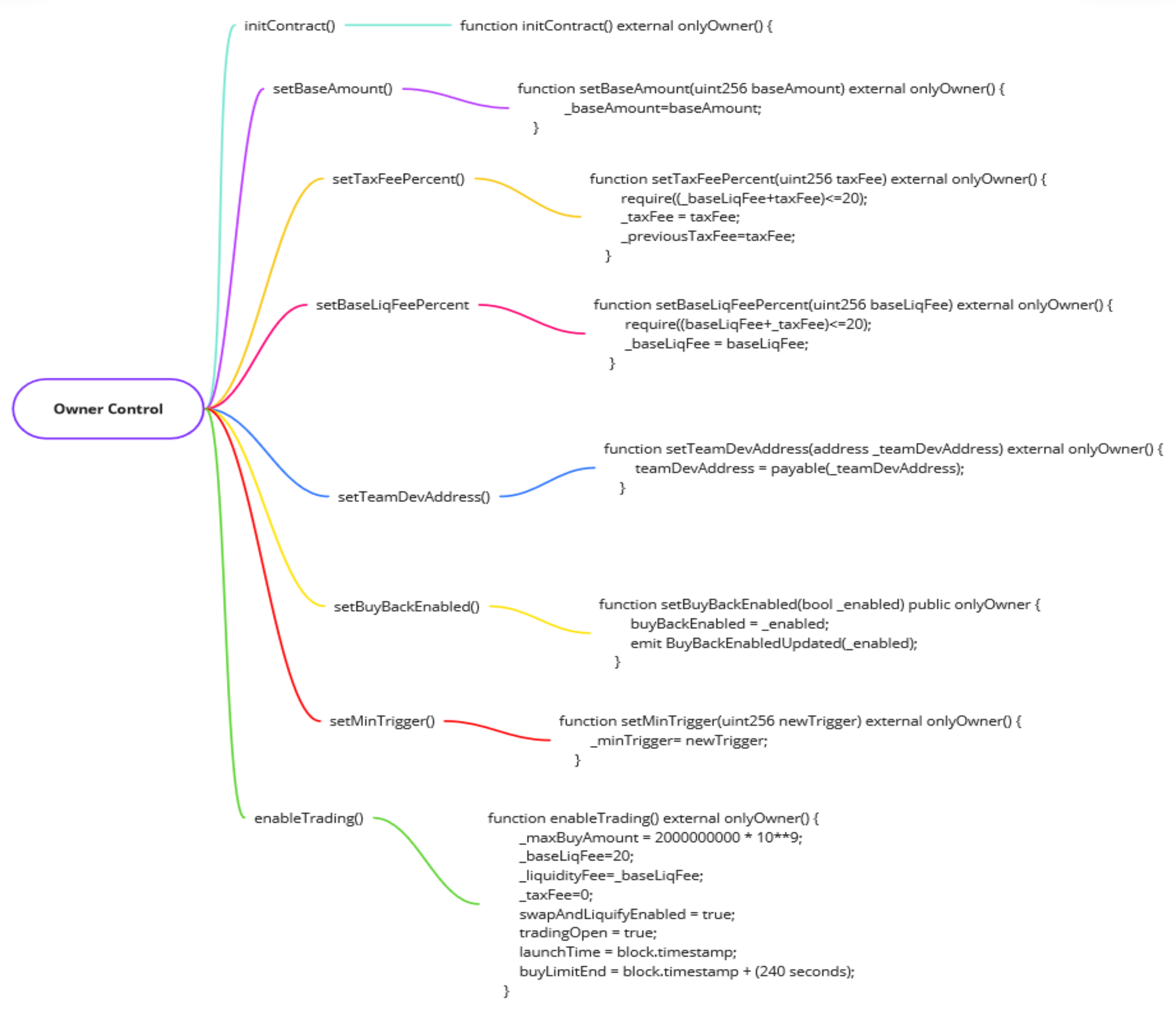
In addition to the well-known vulnerabilities /attacks that are listed in the contract layer, there are other security issues which pose centralisation risks caused by smart contracts. Smart contracts with centralised ownership pose major security issues and act as a single point of failure, which contradicts the very decentralised nature of Blockchain. The following sections highlight contract ownership/centralisation risks.

- *Centralisation/Ownership Control*:
  This happens when the contract owner has control over certain functions or variables. The contract would become more centralised, which contradicts the decentralisation paradigm in Blockchain and can lead to a single point of failure. This means that the power of decision making is exclusively in the hands of owners or developers. For example, an owner can abuse their power or act against the interests of the contract or its users. Hence, the role of "onlyOwner" modifier is extremely important to provide an appropriate access control to owners, developers and all nodes (Mou, Coblenz and Aldrich, 2021). Blockchain security firms such as Certik, Skynet, and Code4rena work with security experts, researchers, and developers to identify possible vulnerabilities in smart contracts. They use AI-based scanning tool, advanced formal verification techniques, static and dynamic analysis tools as well as performing manual (CertiK, 2023). Certik auditors identified 286 discrete centralisation risks within a sample of 1,737 audits that they performed (CertiK, 2022). Both Certik and Code4rena classified contract ownership (or centralisation) as "high risk" yielding single points of failure that external attackers, or malicious insiders, can exploit (Certik, 2021; Code4rena, 2022). Researchers and security experts stated that centralisation is a major security issue which is caused by malicious owners when acting maliciously, compromising the smart contract's integrity by controlling the entire contract balance and stealing all assets on that platform (Sai *et al.*, 2021; CertiK, 2022; Code4rena, 2022; Github, 2022).

There have been many real-world examples that indicated of centralisation risks due to owner control. in 2021, Certik perfumed a security review of the Inari Token, an Ethereum based project. Audit reports contain information about security vulnerably on smart contract. According to Certik's report, four major centralisation risks detected on Inari Token. Smart contract's code is also available on Etherscan (Etherscan, 2021; CertiK, 2021).

The Inari contract includes some privileged functions that are restricted by modifiers such as "onlyOwner". Contract "Ownable" sets a specific owner who has special privileges and control over the contract which is used through inheritance in this project. It uses onlyOwner" to restrict use of functions to the owner. In Inari project, the owner of "Ownable" contract has admin permission to control two functions "transferOwnership" and "lock" (CertiK, 2021).

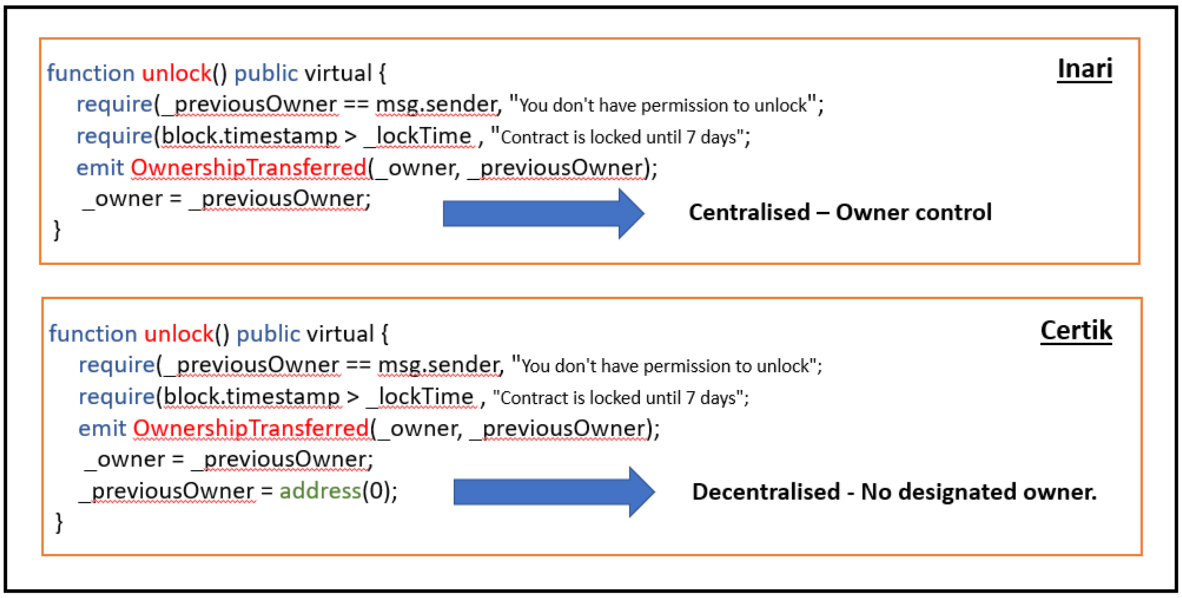
Furthermore, the owner has permission to control the following functions in Inari smart contract (CertiK, 2021). It means that the "onlyOwner" modifier on following functions allows only the owner of the wallet contract to call them. Hackers/ malicious insider can manipulate this project through these functions that owner has authority over them. The diagram in Figure 6 displays functions with owner control.



```
function initContract() external onlyOwner() {

function setBaseAmount(uint256 baseAmount) external onlyOwner() {
        _baseAmount=baseAmount;
   }

function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
        require((_baseLiqFee+taxFee)<=20);
        _taxFee = taxFee;
        _previousTaxFee=taxFee;
   }

function setBaseLiqFeePercent(uint256 baseLiqFee) external onlyOwner() {
        require((baseLiqFee+_taxFee)<=20);
        _baseLiqFee = baseLiqFee;
   }

function setTeamDevAddress(address _teamDevAddress) external onlyOwner() {
        teamDevAddress = payable(_teamDevAddress);
   }

function setBuyBackEnabled(bool _enabled) public onlyOwner {
        buyBackEnabled = _enabled;
        emit BuyBackEnabledUpdated(_enabled);
   }

function setMinTrigger(uint256 newTrigger) external onlyOwner() {
        _minTrigger= newTrigger;
   }

function enableTrading() external onlyOwner() {
        _maxBuyAmount = 2000000000 * 10**9;
        _baseLiqFee=20;
        _liquidityFee=_baseLiqFee;
        _taxFee=0;
        swapAndLiquifyEnabled = true;
        tradingOpen = true;
        launchTime = block.timestamp;
        buyLimitEnd = block.timestamp + (240 seconds);
   }
```

Figure 5 Centralisation Risk When Using "onlyOwner" Modifier in Inari Smart Contract (Adopted from Certik, 2021).

In addition, within the Inari project, tokens are minted to the centralised address “msg.sender” which is the owner’s address. There is a possibility that owner can regain the ownership of contract through *unlock()* function and transfer ownership to the owner. Both Inari’s code and analysis report are available on Etherscan and Certik’s website (Inari, 2021; CertiK, 2021). Figure 7 shows the vulnerable code, used by Inari developers, and the suggested code by Certik.



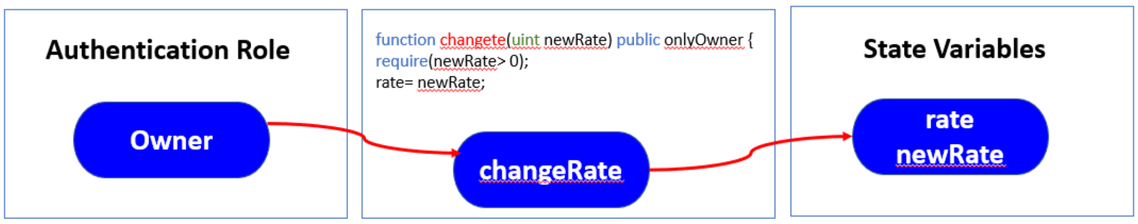
```
function unlock() public virtual {
    require(_previousOwner == msg.sender, "You don't have permission to unlock");
    require(block.timestamp > _lockTime , "Contract is locked until 7 days");
    emit OwnershipTransferred(_owner, _previousOwner);
    _owner = _previousOwner;
}
```

```
function unlock() public virtual {
    require(_previousOwner == msg.sender, "You don't have permission to unlock";
    require(block.timestamp > _lockTime , "Contract is locked until 7 days";
    emit OwnershipTransferred(_owner, _previousOwner);
    _owner = _previousOwner;
    _previousOwner = address(0);
}
```

Inari
Centralised – Owner control
Certik
Decentralised - No designated owner.

Figure 6 Solidity Code in a Centralised Vs Decentralised Application (Adopted from CertiK, 2021).

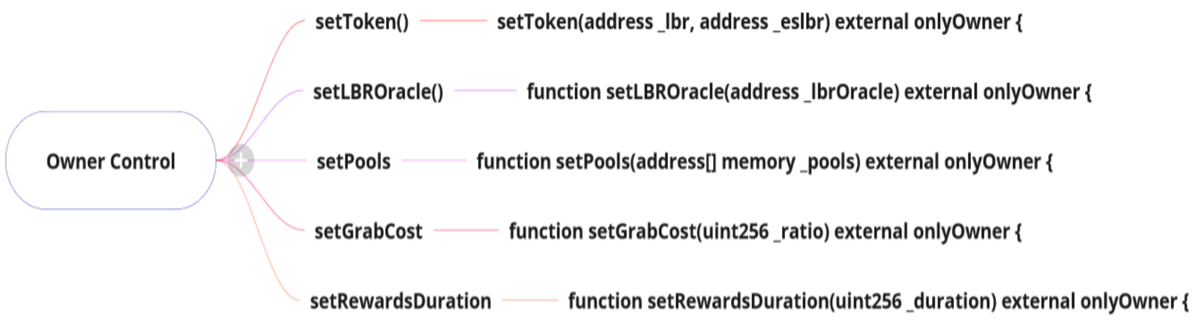
Another example is related to Vector Space AI project. Certik reported a major centralisation risk related to centralisation privilege where the contract owner has the authority over the functions. Certik recommended to minimise centralisation through decentralise mechanism such as Timelock and DAOs and enhance the security and minimise single point of failure via assigning privileged roles to multi-signature wallets (Certik, 2021). The diagram in Figure 8 shows owner's authority over the function.



```
function changete(uint newRate) public onlyOwner {
require(newRate> 0);
rate= newRate;
```

Figure 7 Centralisation Risks When Contract Owner Holds Authority Over the Functions, Adopted from Certik, 2021 (Example 1).

Code4rena analysed the Lybra Finance project which includes 21 smart contracts written in the solidity. Code4rena categorised risks on high, medium, and low/non-critical. Based on provided security analysis report, Lybra Finance project is vulnerable for the role of onlyOwner, where control or decision-making authority is centralised within contract creator or owner which pose high centralisation risk (Code4rena, 2023). The diagram in Figure 9 shows another example of owner's authority over the functions.



```
setToken(address _lbr, address _eslbr) external onlyOwner {
function setLBROracle(address _lbrOracle) external onlyOwner {
function setPools(address[] memory _pools) external onlyOwner {
function setGrabCost(uint256 _ratio) external onlyOwner {
function setRewardsDuration(uint256 _duration) external onlyOwner {
```

Figure 8 Centralisation Risks When Contract Owner Holds Authority Over the Functions, Adopted from Code4rena, 2023 (Example 2)

Blockchain security firms such as Certik, and Code4rena, have identified a major vulnerability on smart contracts which pose major centralisation risks. They suggested that appropriate measures should be implemented to minimise unnecessary control by owner and mitigate centralisation risks effectively.

- *Centralised Access controls*
  Authors in (Ghaffari *et al.,* 2021; Ghaffari *et al.,* 2020; Achour *et al.,* 2021; Nakamura *et al.,* 2019) discussed Blockchain-based access control mechanisms are classified into different categories such as Discretionary Access Control (DAC), Role-Based Access Control (RBAC), Capability-based Access Control (CapBAC), and Attribute-Based Access Control (ABAC).

  - **Discretionary Access Control (DAC):** In this method the owner of object who has the discretion to define access rules to resources that stored within the smart contract. The owner of objects defines and modify access rules and policies for the objects they own. As a result, smart contracts can be control and govern by the owner/s (Ghaffari *et al.,* 2020). The common example would be "onlyOwner" modifier or pre-built implementations like Ownable that is used by OpenZeppelin (Kuryłowicz, 2023; OpenZeppelin, 2023).
  - **Role-Based Access Control (RBAC)**: This approach manages access for subjects based on their roles. This model works based on different components such as

users, roles, operations and permission. In this centralised approach, access is determined by roles and permissions are associated with each role which will be grant and revoke to users (Achour *et al.,* 202).

- **Capability-based Access Control (CapBAC)**: Nakamura *et al.* (2019) proposed CapBAC for Ethereum Blockchain. A specific capability can be implemented to a user (subject) or group of users to access a certain resource (object). CapBAC scheme stores the token delegation relationship among subjects. This scheme allows object owners to verify the ownership and validate the tokens for access control (Nakamura *et al.,* 2019).
- **Attribute-Based Access Control (ABAC):** The researchers in (Ghaffari *et al.,* 2021; Achour *et al.,* 2021) explained that ABAC model allows for more dynamic and fine-grained access management. It allows the object owner define the access rules based on the attribute of user (object), subject, the environment and action attribute. There are four sets of attributes includes: Subject Attributes (username, token), Object Attributes (resources), Environment Attributes (time, location), and Action Attributes (read, write, execute).

3.5.13.3 Rug Pulls Scam

Rug pulls are a lucrative fraud in decentralised finance. Developers, or malicious owners, can create new crypto tokens, list them with decentralised exchanges and market them to investors to increase their value and overall liquidity (Maruf, 2022). They can manipulate smart contracts by using pre-set malicious functions or change critical conditions of the sale at any moment such as the price, the start time, the duration and the whole amount of tokens that are allowed to be sold, lock the contract, freeze the funds, stop users to sell their tokens and withdraw the funds from smart contracts (Code4rena, 2022; Huang *et al.,* 2022). As a result, malicious functions are performed to enable users to buy tokens but not sell their tokens to decentralised exchange and only the token creator would be able to sell tokens or drain funds and crash the token's value to zero before disappearing with the cash. Developers can remove all the cryptocurrencies (Ether) from the liquidity pool and make tokens untradeable without economic value (Huang *et al.,* 2022; Mazorra, Adan and Daza, 2022). Based on Code4rena report, the centralisation risk happened due to admin privilege of malicious owners. The owners would be able to carry out malicious operations to set address to mint any amount of ether, set any address as validator, take more ether than required, rug all ether in the contract and drain all contract funds (Code4rena, 2022; Github, 2022). Researchers stated that machine learning techniques can detect malicious, non- malicious tokens and potential rug pulls before they happen (Mazorra, Adan and Daza, 2022).

3.5.13.4 Private Key Compromise

A digital signature is used to sign a transaction in Blockchain. The owner's private key is required to access and modify certain functions or variables in the smart contract (Huang *et al.,* 2022). Losing the owner's private key or gaining access to the owner's private key through hacking can create a serious security risk for the smart contract. When the owner's private key is lost, the owner would not be able to control, or easily update, the contract. Thus, the entire contract will fail to operate which poses the single point of failure (Code4rena, 2022; Shanzson, 2022). When the owner's account gets hacked, the new owner becomes malicious and can control the contract and steal as much of the funds as possible (Code4rena, 2022). For example, in April 2021, the DeFi protocol EasyFi was hacked due to key management compromise. A hacker compromises the admin/owners MetaMask wallet and accesses the keys, he took control over the smart contract and steal funds for a worth of $80M (Future Learn, 2021).

3.5.13.5 Preventive Methods and Mitigation Steps

To mitigate centralisation risks, it is important to implement secure mechanisms and additional layers of security to ensure decentralisation and no single point of failure within the network.

Manual Analysis

There are different auditing tools that can be used to identify different vulnerabilities and errors within the smart contracts. However, based on the findings from current academic publications, searching on security audit websites and talking to experienced developers, the researcher could not find any specific tools for identifying, or detecting, the owner control vulnerability. This vulnerability can be detected by manual analysis through skilled developers or auditors. It is important to review the code and functions line-by-line to identify any bugs, or logic errors, risky functions, contract ownership and inconsistencies.

For example, it is vital to carefully consider the use of the “onlyOwner” modifier and to implement other mechanisms to reinforce decentralisation and therefore increase security.

Implementing Decentralised Access Control Mechanisms

There have been many real-world attacks due to access control vulnerabilities/bugs. Failing to implement appropriate access control may cause major security risk and significant financial loss (Ghaleb, Rubin and Pattabiraman, 2023). For example, in May 2021, Value DeFi was hacked due to coding mistake in smart contract. The coding error and lack of proper access control mechanism allowed a hacker to make themselves an owner of contract, re-initialise a liquidity pool and drain the staked tokens for a worth of $20M (Ghaleb, Rubin and Pattabiraman, 2023; Future Learn, 2021). Researchers proposed different access control approach/framework to detect access control vulnerabilities. Ghaleb *et al.* (2023) proposed a static analysis approach called AChecker to determine access control checks in smart contract (Ghaleb, Rubin and Pattabiraman, 2023). Other tools such as Mythril use different method such as symbolic execution to detect smart contract bug including access control bugs. However, not all smart contract vulnerabilities can be detected by available tools. They may detect vulnerabilities results in false negatives (Ghaleb, Rubin and Pattabiraman, 2023).

Using a DAO Structure

Within the decentralised distributed organisation, power is distributed among the members with no central entity holding the control and able to change the rules. All decisions should be approved by most of the DAO members (Santana and Albareda, 2022).

Use Rug Checker Tools

There are some tools that are available at poocoin.app/rugcheck, rugscreen.com**,** rugpulldetector.com, honeypot.is, solidityscan.com, rugdoc.io/honeypot to detect a rug pull and minimise/avoid this scam (Shanzson, 2022).

Set up Time-based Access Control on Privilege Operations

Implementing temporary lock feature helps to restrict access to a smart contract specifically sensitive functions such as like transferring ownership or minting tokens. The state of the smart contract would be locked for a specified length of time and delay the execution of a transaction until predetermined amount of time has passed. Setting up a timelock allows the owner relinquish ownership and prevent owner and anyone else from calling a contract during this time (Mou, Coblenz and Aldrich, 2021; CertiK, 2021).

Multi-Signature Accounts

Private key as part of digital signature plays important role in Blockchain. A single signature scheme allows only one user/owner to agree to a transaction whereas multi-signature scheme allows several owners to validate and sign a transaction (Di Angelo and Slazer, 2020; Han *et al.,* 2021). Multi-signature adds an extra layer of security because it requires a minimum number of addresses to sign a transaction before executing it. This means assigning of privileged roles to multi-signature wallets helps to prevent a single point of failure due to the private key, and even if one of the signatories is compromised, the funds in the wallet are still safe, as the attacker would need to gain access to at least one other signatory's private key in order to execute a transaction (CertiK, 2021). This method would improve access control by handing over privileged roles to multi-signature smart contracts (Le, Yang and Ghorbani, 2019; Destefanis *et al.,* 2018; Di Angelo and Slazer, 2020; Han *et al.,* 2021). Li, Ma, and Luo (2022) proposed an efficient asymmetric encryption scheme by combining homomorphic encryption and state-of-the-art multi-signature key aggregation and non-interactive zero knowledge proof to preserve privacy and verify valid transactions.

Vulnerabilities/ Attacks location	**Typical vulnerabilities/Attacks**	**Authors Of key Works**	**Detection Tools/ PreventiveTechniques**
Contract Layer	Re-entrancy	Antonopoulos and Wood. (2018)	- limit calls to external contract. - Mutex to lock some function states.

🟩		Hooper Solorio and Kanna. (2019) Shahda. (2019) Alkhalifah et al. (2021) Khan and Siami. (2020) Feng et al. (2019) Fang et al. (2021)	- Security analysis static tools such as Oyente, Teether, Gasper, Vandal, Securify, smartcheck, Zeus. - Security analysis dynamic tools such as Vultron, Sereum, Regaurd. - Fuzzing tool such as ContractFuzzer. - Use taint analysis and symbolic execution such ad OSIRIS, EasyFlow, SmartScopy. - Sereum (Secure Ethereum) to perform run-time monitoring of SC execution. - Jyane, a dynamic path profiling solution for SC.
🟩	Parity multi signature wallet	Praitheeshan et al. (2019) Vivar et al. (2020) Chen et al. (2020) Goldberg. (2018) Antonopoulos and Wood. (2018) Wang et al. (2020)	- Adopt the private modifier by default. - Avoid using “delegateCall” *as a* catch-all forwarding mechanism. - Build stateless libraries. - Use static security analysis tools such as Oyente. - Use verification tool such as Artemis.
🟩	Front running/Transaction ordering dependence	Praitheeshan et al. (2019) Eskandari et al. (2019) Najafi. (2020)	- Use cryptographic commit-reveal scheme to limit visibility of transaction details. - Enforce rules such as first in first out (FIFO) by adding a complex consensus-based solution.

		Varun, Palanisamy and Sural. (2022)	- Remove miner’s ability to arbitrarily order transaction by forcing queuing/ordering for the transactions. - Use static security analysis tools such as Oyente, Securify, Mythril. - Use IDS and ADS.
	Integer overflow/Underflow	Ma et al. (2019) Praitheeshan et al. (2019) Khan and Siami. (2020) Gao et al. (2019)	- Create dedicated mathematical libraries and use SafeMath. - Check the validity of math output. - Use static security analysis tools such as Oyente, Zeus. - Use dynamic security analysis tools such as Vultron. - Use taint analysis and symbolic execution such ad OSIRIS, EasyFlow.
	Timestamp dependence	Antonopoulos and Wood. (2018) Hooper, Solorio and Kanna. (2019) Praitheeshan et al. (2019) Jiang et al. (2018) Feng et al. (2019) Khan and Siami. (2020)	- Use static security analysis tools such as Oyente, Remix, Mythril, SmartCheck, Zeus. - Fuzzing tool such as ContractFuzzer. - Use SmartScopy as an attack synthesiser. - Use The 15-second Rule. - Not rely on block.timestamp or blockhash as a source of randomness - Avoid using block.number as a timestamp.

	Mishandled exceptions	Praitheeshan et al. (2019) Khan and Siami. (2020)	- Use static security analysis tools such as Oyente, Remix, Mythril, SmartCheck, Securify, GasFuzzer. - Handle the error manually in the caller contract and check the return value of functions.
	DoS with unexpected revert	Ma et al. (2019) Samreen and Alalfi. (2021)	- Propose a framework called SmartScan that combines static and dynamic analysis to identify vulnerable pattern and detect. - Isolate *if /for* statements with an external function call.
	Short address	Wen et al. (2021) Antonopoulos and Wood. (2018) Feng et al. (2019) Vivar et al. (2020) Kushwaha et al. (2022)	- Use SmartScopy as an attack synthesiser. - Validate input parameters in external applications before sending them. - Check parameter ordering. - Dynamic analysis tool Etherolic and SoliAudit. - Use static security analysis tools such as SmarCheck.
	DoS- Block gas limit	Chen et al. (2020) Ghaleb et al. (2022) Grech et al. (2018)	- Use static analyser tools MadMax, eTainter. - Avoid using loops over data structures. - Splitting the loop over multiple transactions to alleviate the risk of an unbounded loop. - Implement access control to restrict the call of the public

			function to only the owner of the contract.
	Tx.origin	Antonopoulos and Wood. (2018) Chen et al. (2020) Tikhomirov et al. (2018)	- Check the authorisation of ownership by using msg.sender' in place of `tx.origin'. - Use static security analysis tools such as SmartCheck.
	Weak randomness	Amiet. (2021) Chatterjee et al. (2019)	- RANDAO, a secure random number generator. - Designed a well-incentivised and unmanipulable approach which provides a trustworthy source of randomness that is not rely on malicious miners or off-chain oracles.
	Hash Collisions with Multiple Variable Length Arguments	Swcregistry. (2020)	- Ensure matching signature cannot be achieved using different parameters. - Avoid using abi.encodePacked()and alternatively use abi.encode() instead.
	One owner control (Centralised Ownership)	Certik. (2022) Mou et al. (2021) Li et al. (2022) CertiK. (2023) Ghaffari et al. (2021) CertiK. (2021) Shanzson. (2022)	- Manual analysis a. Check contract's ownership b. Correct permission to critical functions. c. Renounce the ownership / never claim the privileged roles. d. Remove the risky functionality.

			- Implement multi signature accounts, use an efficient asymmetric encryption scheme by combining homomorphic encryption and state-of-the-art multi-signature key aggregation and non-interactive zero knowledge proof to preserve privacy and verify valid transactions. - Implement access control mechanisms. - Set up time-based access control on privilege operations. - Implement DAOs. - Use rug checker tools.

Table 3 - Current Work on Vulnerabilities/Attacks and Related Counter-measures within the Contract Layer

3.6 Vulnerabilities/Attacks on the Incentive Layer

3.6.1 Blockchain Denial of Service (BDoS) Attack

A BDoS attack is the main security risk at the level of the incentive layer. Blockchain Denial of Service (BDoS) is an incentive-based attack, whereby the malicious actor manipulates the incentive mechanism (Mirkin *et al.,* 2020). The malicious attacker invests resources by generating a block and only publishes a proof that s/he mined it, without publishing the block itself. This, to the honest miners, is regarded as an advantage gained by the malicious actor, which leads to reducing miners' incentive to mine. As miners cease to mine, the entire Blockchain can grind to a halt. Incentive-based attack can force a certain order of transactions or transaction omission (Mirkin *et al.,* 2020). Mirkin et al (2020) present a fuller description of BDoS and its impact on the incentive mechanism and the Blockchain. They describe a mathematical model that increases the threshold of a partial shutdown of the system (Mirkin *et al.,* 2020).

A possible way to weaken this attack is to implement effective incentive mechanism and change miner behaviour (Mirkin *et al.,* 2020). Wang *et al*. (2018) designed a Blockchain-based privacy-preserving incentive mechanism in crowdsensing applications by using signcryption method to prevent malicious miners or attackers to pose privacy issue for users. Hou *et al.* (2019) proposed a framework called SquiRL for using deep reinforcement learning (DRL) to analyse attacks on Blockchain incentive mechanisms.

Vulnerabilities/ Attacks location	**Typical vulnerabilities/Attacks**	**Authors Of key Works**	**Detection Tools/ Preventive Techniques**
Incentive Layer	BDoS	Wang et al. (2018) Hou et al. (2019)	- Blockchain-based privacy-preserving Incentive mechanism in crowdsensing applications by using signcryption method to prevent malicious miners or attackers to pose privacy issue for users. - Proposed a framework called SquiRL to analyse attacks on Blockchain incentive mechanisms.

Table 4 - Vulnerabilities/Attacks and Related Counter-measures on Incentive Layer

3.7 Vulnerabilities/Attacks on the Consensus Layer

3.7.1 Double-Spending Attack

Double-spending refers to the risk of the cryptocurrency being spent twice. The attacker would send a copy of the currency transaction to make it look legitimate, thus disrupting the Blockchain network and, essentially, stealing the cryptocurrency. There are mainly three different types of double-spending attacks: the Race, the Finney, and the Vector (Wen *et al.,* 2021).

In a Race attack the malicious actor would send a token from their own address to the wallet address of the potential victim user. Then the malicious actor sends the same token to another of their own wallet address with higher transaction fees. The two transactions are logged in two blocks. This leads to the transaction with higher fees to be confirmed and the one sent to the victim to be orphaned and rolled back (Wen *et al.,* 2021).

A Finney attack is a more complex version of the Race attack, where a miner is involved in the transaction. In this case, the malicious actor, the miner, pre-mines a block with their payment to the "intended victim user, but creates another transaction before the pre-mined block is broadcast to the network, leading the network to reject the transaction sent to the victim user (Wen *et al.,* 2021).

A Vector attack is a combination of a Race attack and a Finney attack (Wen *et al.,* 2021).
Wen et al. (2021) argue that the most effective and convenient way to prevent double-spending attacks mostly race attack and vector 76 attack is by increasing the confirmation times (Wen *et al.,* 2021). As majority 51% attack has direct impact on double spending attack, it is important to avoid forming large-scaled mining pools (Wen *et al.,* 2021). Begum et al. (2020) have developed a model based on a change in governance protocol. In addition to the previous model which prevents double spending attacks with simple changing in governance protocol. Chen. (2021) proposed Blockchain access restriction (BAR) as a prevention mechanism to detect malicious behaviour and check the actual block request while transaction is recorded on a specific block. BAR can protect miner's privileges and provide fairness (Xing and Chen, 2021).

3.7.2 51% Majority Attack

Here the malicious actor is in a position to control (at least) 51% of the computing power to control the mining process (Wen *et al.,* 202). They would create a chain of blocks that is fully isolated from the real (honest) version of the chain. Using their 51% advantage they can process their blocks faster, and with time the isolated (malicious) chain is established as a genuine one. Many regards 51% majority as a form of double-spending (Wen *et al.,* 2021). Kitakami and Matsuoka devised an 'agreement algorithm' as a basis for a scheme to strengthen resilience against 51% attacks (Kitakami and Matsuoka, 2018). Mirkin et al. (2020)

stated that malicious miners can perform a full-fledged DoS attack through controlling a majority of mining power, generate empty block and ignore other blocks (Mirkin *et al.,* 2020). Wen et al. (2021) argue that the most effective way to prevent this attack is avoiding forming large-scaled mining pools to control mining process (Wen *et al.,* 2021). Akbar et al. (2021) suggested to combine two consensus algorithms PoW and PoS to provide a fair mining reward to miners and validators (Akbar *et al.,* 2021).

3.7.3 Selfish Mining Attack

Malicious miner can compromise Blockchain network to get higher block rewards (Saad *et al.,* 2019). One of the drawbacks of consensus mechanisms such as PoW, is that miners are able to collaborate with each other, use a set of selfish strategies to gain more rewards than they would otherwise do if they mine individually. Such miners are called selfish miners and their "illegitimate" mining collaboration is called selfish mining. This is not fair for the other honest miners who stick to the rules specified by the consensus mechanism used (Wen *et al.,* 2021]. Wen et al. (2021) provided a good review of previous work to strengthen consensus mechanisms against selfish mining. Saad et al. (2019) proposed an algorithm to enforce fair mining. The proposed solution is able to detect the behaviour of selfish miners and encourage the network to defence and disincentivises selfish miners (Saad *et al.,* 2019).

3.7.4 Bribery Attack

Adversary misuses Blockchain protocol and obtains the majority of computational power and to bribe miners in order to subvert the consensus agreement and achieve additional profits. Attackers can increase the probability of double-spending by bribing other miners (Sun, Ruan and Su, 2020). Several mechanisms for bribery have been proposed with various trust and risk properties (Bonneau, 2016; Liao and Katz, 2017). The evaluation of these different bribery mechanisms remains problematic due to the lack of systematic methods to quantify them. Bonneau. (2016) presented a few schemes to render bribery attacks ineffective. Such schemes, coupled with the fact that PoW makes it very costly for a bribery to be set, it will be fair to say that bribery attacks are not the worst "headache" for the consensus mechanism. Bonneau. (2016) stated that extra confirmation for large transaction make this attack difficult to succeed because of increasing the number of blocks in the attempted fork and its impact to increase the cost of bribe. Add block confirmation time would prevent double spending and bribery attacks (Bonneau, 2016; Wen *et al.,* 2021). Wen et al. (2021) suggested to avoid forming large-scaled mining pools (Wen *et al.,* 2021).

Vulnerabilities/ Attacks location	**Typical vulnerabilities/Attacks**	**Authors Of key Works**	**Detection Tools/ Preventive Techniques**
Consensus Layer	Double spending • Race • Vector 76 • Finney	Wen et al. (2021) Xing and Chen. (2021)	- Increase confirmation time. - Blockchain access restriction (BAR) prevention mechanism to detect malicious behaviour and check the actual block request while transaction is recorded on a specific block. BAR can protect miner's privileges and provide fairness.
	51% Majority (A)	Wen et al. (2021) Akbar et al. (2021)	- Avoid forming large-scaled mining pools. - Combine two consensus algorithms PoW and PoS to provide a fair mining reward to miners and validators.
	Selfish mining (A)	Saad et al. (2019)	- Algorithm to enforce fair mining.
	Bribery (A)	Bonneau. (2016) Wen et al. (2021)	- Extra confirmation for large transaction. - Add block confirmation time. - Avoid forming large-scaled mining pools.

Table 5 - Current Work on Vulnerabilities/Attacks and Related Counter-measures on Consensus Layer

3.8 Vulnerabilities/Attacks on the Network Layer

3.8.1 DDoS Attack

According to Saad et al. (2018) as with any network infrastructure, the Blockchain network layer is vulnerable to Distributed Denial of Service (DDoS) attacks. Such attacks can impact the memory pools (repository of unconfirmed transactions) and cause massive transaction backlog and trap users to pay higher mining fees. Saad and co-authors proposed two methods to optimise the mempool size and counter DDoS attacks on the Blockchain network layer by using a fee-based and an age-based methods. Saad et al. (2018) stated that the fee-based design is more suitable when the attack is not severe. This techniques filters spam transactions and reduce the size of mempool. On the other hand, if the attack is severe, the age-based design would be more useful. This technique rejects unconfirmed transactions that generated by attackers and accept the transactions of honest users by the mempool. This method increases the cost of attack and reduces the time window for a successful attack (Saad *et al.,* 2018).

3.8.2 Domain Name Service

The Domain Name System (DNS) plays a vital role in the internet. Nodes on peer to peer network are communicating with other contributors to transmit data through node discovery protocol. This protocol works based on DNS seed address that distribute the address of other active nodes on the network (Sai *et al.,* 2021). Researchers explained that the current DNS system is vulnerable to many attacks such as eclipse attack, DDOS attack, cache poisoning attack, single point of failure and centralisation (Li *et al.*, 2021).

Current DNS suffer security and privacy issues due to poor process of node discovery protocol, weak verification mechanism which leads to the cache poisoning attack and makes domain owners to observe nodes on network, claim their domain ownership and change the IP addresses of their domains. As Secure DNS are not yet in place, this would move ownership, control of the authentication keys to the user's security domain and poses centralised DNS services that can act as a single point of failure which makes legacy DNS vulnerable to DDoS attacks (Ren *et al.*, 2019; Li *et al.*, 2021). Blockchain-based DNS assist to minimise some of the security concerns. Blockchain-based ENS which is a distributed, decentralised naming system built on the Ethereum Blockchain, provides a decentralised ownership. However, because

Ethereum Name Service (ENS) is stored on smart contract and ENS registry contains a list of domain names, subdomains, important information about owner of domain name, the resolver of the domain and the caching time for all records under the domain (Liu *et al.,* 2019). ENS relies on smart contract to manage domain name ownership (Liu *et al.,* 2019). Therefore, it may be controlled/manipulated by a malicious developer/owner or attacker.

Researchers suggested different techniques to counter this vulnerability. Jin et al. (2021) proposed a Blockchain based naming service called DNSonChain. This technique works based on majority vote mechanism to validate the domain ownership in a decentralised manner which solve the DNS privacy issue (Jin *et al.,* 2021). The other researchers introduced Blockchain based domain name system called B-DNS which offers better protection against the DDoS attack and the cache poisoning attack (Li *et al.,* 2021). Ren et al. (2019) presented Blockchain-based decentralised naming system called blockDNS which enhances domain name ownership and data authenticity through a verification mechanism that helps to solve the centralisation issue (Ren *et al.,* 2019). Others suggested having a better node discovery protocol and encrypting DNS traffic will help with privacy issues (Jin *et al*., 2021; Li *et al.,* 2021; Sai *et al.*, 2021).

3.8.3 Eclipse Attack

In an eclipse attack the malicious actor attempts to own a plenty of IP addresses to take control of all honest node's connections. Adversary node isolates a node and manipulates it into illegitimate action. Attackers typically use botnet to compromise the node and seal it off. The victim node is isolated within an environment that is completely separate from the actual network activity. Because the attack relies heavily on exploiting the victim's neighbouring nodes, its success will depend on the structure of the Blockchain network (Wen *et al.,* 2021). Xu et al. (2020) proposed a detection model called ETH-EDS. This Ethereum detection model works based on random forest classification algorithm to detect malicious actor with high probability (Xu *et al.,* 2020). Wen et al. *(*2021) gathered countermeasures and tools that suggested by other researchers to detect and prevent this attack (Wen *et al.,* 2021).

3.8.4 Sybil Attack

In a Sybil attack, the malicious actor(s) can take over the entire network. Attackers may then be able to out-vote the honest nodes if they create multiple fake identities (or Sybil identities). They can then control the reception and transmission of blocks, effectively blocking other honest users from the network (Wen *et al.,* 2021). Malicious pool operator can add a large number of miners with zero power into mining pool and run sybil attack. These miners cannot mine any blocks, they can participate in data propagation for malicious user and stop propagating of honest user's data. therefore, only attacker's block would add to the network and attacker get higher rewards and decrease the throughput of network (Swathi, Modi and Patel, 2019). This attack may lead to several attacks like DoS, DDoS and 51% majority (Swathi, Modi and Patel, 2019). Swathi et al. (2019) proposed a solution to monitor the behaviour of each node, detect sybil nodes within the network and notify them to honest nodes (Swathi, Modi and Patel, 2019). Siddiqi and Ali. (2022) proposed a prevention technique that generate node's id, timestamp, password and encryption code which code uses RSA mechanism for node authentication. By using this technique, nodes would be identifying sybil nodes via data transmission process. Thus, this method helps to increase the throughput and improves network performance (Siddiqi and Ali, 2022).

3.8.5 BGP Routing Attack

The Border Gateway Protocol (BGP) is a routing protocol used to exchange routing information (IP packets) among autonomous systems (ASes) on the Internet (Saad *et al.,* 2022). BGP routing attack known as BGP hijacks or prefix hijack. This attack can happen when a malicious AS broadcasts fake IP prefix announcement, propagate wrong routing information. Thus, attacker can split the network into two or more disjoint components, controlling communication within components and outside of them, reroute the traffic and forks Blockchain into parallel chains (Wen *et al.,* 2021; Saad *et al.,* 2022).

Xing, *et al.* (2018) designed a system called BGPcoin which controls by a set of smart contracts to manage internet number resource (Internet address (IP) and autonomous system number (ASN)). This system authorises autonomous systems and provides a reliable origin advertisement/authentication source for BGP system (Xing, *et al.,* 2018). Apostolaki, *et al.*

(2019) built SABRE as a secure and scalable bitcoin relay network, which relays blocks worldwide through a set of connections that are resilient against BGP routing attacks. Saad, *et al.* (2022) proposed a secure Blockchain-based BGP routing system named RouteChain. This technique provides a temper proof route management through Blockchain validation source for all BGP announcements and use of consensus algorithms to achieve agreement between ASes over the prefix nature (Saad *et al.,* 2022).

3.8.6 Replay Attack

Replay attack happens more likely happens during a hard fork when the Blockchain is split into two when a malicious actor spoofs the communication between two valid nodes and gains access to the hashkey (Antonopoulos and Wood, 2018). Adversary captures a signed message and attempts to delay or retransmit data as a valid user to subvert the receiver (Hu *et al.,* 2019). One way to counter this attack, is an account nonce which is a transaction counter in each account and cannot be used again (Antonopoulos and Wood, 2018). Ramanan and Gebraeel. (2022) developed a Blockchain-based framework that relies on Bayesian inference to detect replay attack with full data privacy.

Vulnerabilities/ Attacks location	**Typical vulnerabilities/Attacks**	**Authors Of key Works**	**Detection Tools/ Preventive Techniques**
Network Layer	DDoS	Saad et al. (2018)	- Filter and reject unconfirmed transactions generated by attackers. ○ the fee-based design ○ the age-based design
	DNS/ ENS Ownership (Centralisation)	Sai et al. (2021) Jin et al. (2021) Li et al. (2021) Ren et al. (2019)	- Encrypt DNS traffic. - Enhance node discovery protocol. - DNSonChain (Blockchain based naming system). - B-DNS (secure and efficient Blockchain-based DNS).

			- BlockDNS (Blockchain-based decentralised naming system).
	Eclipse (A)	Xu et al. (2020) Wen et al. (2021)	- Detection model called ETH-EDS. - Enforce an upper limit on the number of TCP connection. - ADvISE, behavior detection tool.
	Sybil	Swathi et al. (2019) Siddiqi and Ali. (2022)	- Monitor node's behaviour and detect sybil nodes. - Node authentication mechanism and detect sybil nodes.
	BGP routing	Saad et al. (2022) Xing et al. (2018)	- Secure Blockchain-based BGP routing system called RouteChain. - BGPcoin, BGP Security Solution.
	Replay	Antonopoulos and Wood. (2018) Ramanan and Gebraeel. (2022)	- Blockchain-based framework that relies on Bayesian inference to detect. - Use an account nonce.

Table 6 - Current Work on Vulnerabilities/Attacks and Related Counter-measures on Network Layer

3.9 Vulnerabilities/Attacks on Data Layer

3.9.1 Transaction Malleability Attack

This is an attack that can be associated with either, or both, the Network layer and the Data layer (Wen *et al.,* 2021). A transaction consists of data that stored on Blockchain. To protect this data, Blockchain is using cryptography (hashing algorithm and digital signature). Transaction ID (TXID) or transaction hash is given to every transaction that is verified and added to the chain. It is an illegitimate modification to a transaction that is being broadcast, prior to being accepted in a block. In a Blockchain peer-to-peer network, transactions get passed from one node to another. A malicious node receives the transaction and creates a modified version of signature, by altering the transaction identifier (TXID), before passing it to other nodes in the Blockchain (Wen *et al.,* 2021; Sward, Vecna and Stonedahl, 2018). The consequence of a successful transaction malleability attack can result in additional attack such as double-spending (Khan, Arshad and Khan, 2020). Sward *et al*. (2018) proposed a method that increases the cost to the malicious author, thus making it a less attractive option to gain financial rewards (Sward, Vecna and Stonedahl, 2018). Ubaidullah et al. (2018) suggested a solution to detect transaction malleability attempts. The solution is to combine hash of transaction script without the signature (i.e. scriptSig) and the hash of the final transaction should be used as a transaction ID during the verification (Ubaidullah Rajput, Fizza Abbas, Heekuck, 2018).

3.9.2 Timejacking Attack

Timejacking happens due to the vulnerability of timestamp processing in a Blockchain. All participant nodes in a Blockchain network internally maintain a time counter, which displays the network time. Hackers can add multiple sybil nodes to the network and alter the node time at the same time. This can slow down the median time of the targeted node by sending inaccurate timestamps as well as splitting the network into several parts and isolate the targeted node from the network (Wen *et al.,* 2021). Thus, miners are wasting computational powers on stale block and network suffers of fake transactions (Sigurdsson, Giaretta and Dragoni, 2020). This issue can be prevented by using node's system time, instead of network time, to determine the upper limit of block timestamps, tighten the acceptable time ranges, use the median Blockchain time during block validation, use only trusted nodes (Conti *et al.,* 2018; Sigurdsson, Giaretta and Dragoni, 2020). Ma et al. (2019) suggested an optimised timestamp protocol to reach a consensus through (trusted) third party timestamp services (Ma *et al.*, 2019).

3.9.3 Quantum Attack

Attackers can launch a quantum attack on the cryptographic part of Blockchain to calculate the private key from public key by using Shor's algorithm. The level of the risk in Ethereum is high and quantum attackers can lunch this attack to do hash collision. They can take complete control of an account and drain all funds (Wen *et al.,* 2021). Researchers are working on post-quantum cryptography to protect Blockchain systems against quantum attack (Kearney and Perez-Delgado, 2021; Khalifa, Bahaa-Eldin and Sobh, 2019). Furthermore, Vitalik Buterin Proposed Hard fork strategy alongside with applying Winternitz signatures scheme and zero-knowledge proofs to improve the security of transactions and protect against quantum attack on Ethereum network (Swayne, 2024).

Vulnerabilities/ Attacks location	Typical vulnerabilities/Attacks	Authors Of key Works	Detection Tools/ Preventive Techniques
Data Layer	Transaction Malleability	Ubaidullah et al. (2018)	- Combine hash of transaction script without the signature (i.e. scriptSig) and the hash of the final transaction should be used as a transaction ID during the verification.
	Timejacking	Conti et al. (2018) Sigurdsson et al. (2020)	- Use system time, and narrow time range, use the median Blockchain time during block validation. - Use only trusted nodes, using node's system time, instead of network time.
	Quantum	Kearney and Perez-Delgado. (2021) Khalifa et al. (2019) Swayne. (2024)	- Use appropriate postquantum. - Use secure digital signature schemes. - Hard fork strategy, Winternitz signatures scheme and zero-knowledge proofs.

Table 7 - Current Work on Vulnerabilities/Attacks and Related Counter-measures on Data Layer

3.10 Vulnerabilities/Attacks on the Physical Layer

3.10.1 Cold Wallet Theft

With the interest of using hot wallets on portable devices, attackers attempt to use different techniques to disrupt confidentiality, integrity and availability of valuable assets on wallets.

As the software wallets store the keys on a computer or smartphone, there are more vulnerable to security breach. Therefore, the alternative option, which is an offline wallet or cold wallet, is introduced to users. A hardware wallet, which is a more secure wallet, has no internet connection and transfers keys and transactions through a USB stick, Bluetooth device or smart card with special embedded software to do cryptography functions (Conti *et al.,* 2018). However, hardware wallets are suffering from lack of a secure and convenient backup and recovery process of private keys. For example, a cold wallet can be vulnerable to Man-In-The-Middle attack. Some of the cold wallets use a terminal like a smartphone or a computer to communicate with the user. Therefore, hackers can capture NFC wireless communication or install malware on the terminal and make a Man-In-The-Middle attack. Another vulnerability would be a brute force attack. Hackers can use a brute force attack to work out what the passphrase is (Rezaeighaleh and Zou, 2019). Moreover, wallets are hosted in an operating system and the running environment may be exploited, hence the security threat posed to the crypto wallet (Hu *et al.,* 2021).

Rezaeighaleh and Zou (2019) proposed a secret sharing mechanism, multi signature and a key agreement protocol (Elliptic-Curve Diffie-Hellman) for secure backup and recovery (Rezaeighaleh and Zou, 2019). Due to the vulnerabilities in both wallets and operating environment, it is important to use multi factor authentication, update the operating system, antivirus, antimalware as well as enhancing the security awareness (Saleh, 2022).

3.10.2 Cryptojacking Malware

Cybercriminals employ various techniques to hijack the computational resources of target devices to mine cryptocurrency. Attackers use two types of cryptojacking malware. They can install an application on a target device (executable-type cryptojacking) that computes hashes secretly or they use browser-based cryptojacking. In this case, users visit the infected website and provide their CPU power to compute hashes. Thus, mining happens within the client browser (Tanana, 2020).

To minimise this security risk, Saad et al. (2019) analysed the static, dynamic, and economic aspects of browser-based cryptojacking and suggested a detection method and some countermeasures such as monitoring the usage of CPU, using malware scanner, using browser extensions to block cryptominers and avoiding suspicious websites (Saad, Khormali and Mohaisen, 2019). Carlin et al. (2018) offered a technique to detect browser-based cryptojacking using dynamic opcode analysis that uses the random forest machine learning algorithm. In contrast, Tanana. (2020) proposed a detection technique based on CPU load by an application that would be able to deal with both executable (type and browser-based) cryptojackers (Tanana, 2020).

Vulnerabilities/ Attacks location	**Typical vulnerabilities/Attacks**	**Authors Of key Works**	**Detection Tools/ Preventive Techniques**
Physical Layer	Cold wallet theft	Rezaeighaleh and Zou. (2019) Saleh. (2022)	- Use multi signature and a key agreement protocol. - Use multi factor authentication, update the operating system, antivirus, antimalware.
	Cryptojacking	Saad, Khormali and Mohaisen. (2019) Tanana. (2020) Carlin et al. (2018)	- Browser-based cryptojacking detection technique, monitoring the usage of CPU, using malware scanner, using browser extensions to block cryptominers and avoiding suspicious websites. - Use of dynamic opcode analysis to detect browser-based cryptojacking with Technique based on CPU load by an application that would be able to deal with both executable-type and browser-based cryptojacker.

Table 8 - Current Work on Vulnerabilities/Attacks and Related Counter-measures on Physical Layer

3.11 Towards a Conceptual Taxonomy and Classification

Sections 3.4 – 3.10 provided a comprehensive overview of the different vulnerabilities and attacks associated with each layer of a seven-layer Blockchain. For each vulnerability, an explanation is given about how it is exploited and the potential consequences of such exploitations. Defensive methods are described and countermeasures proposed. An overview of vulnerabilities, attacks and their consequences is depicted in the taxonomy shown in Figure 10.

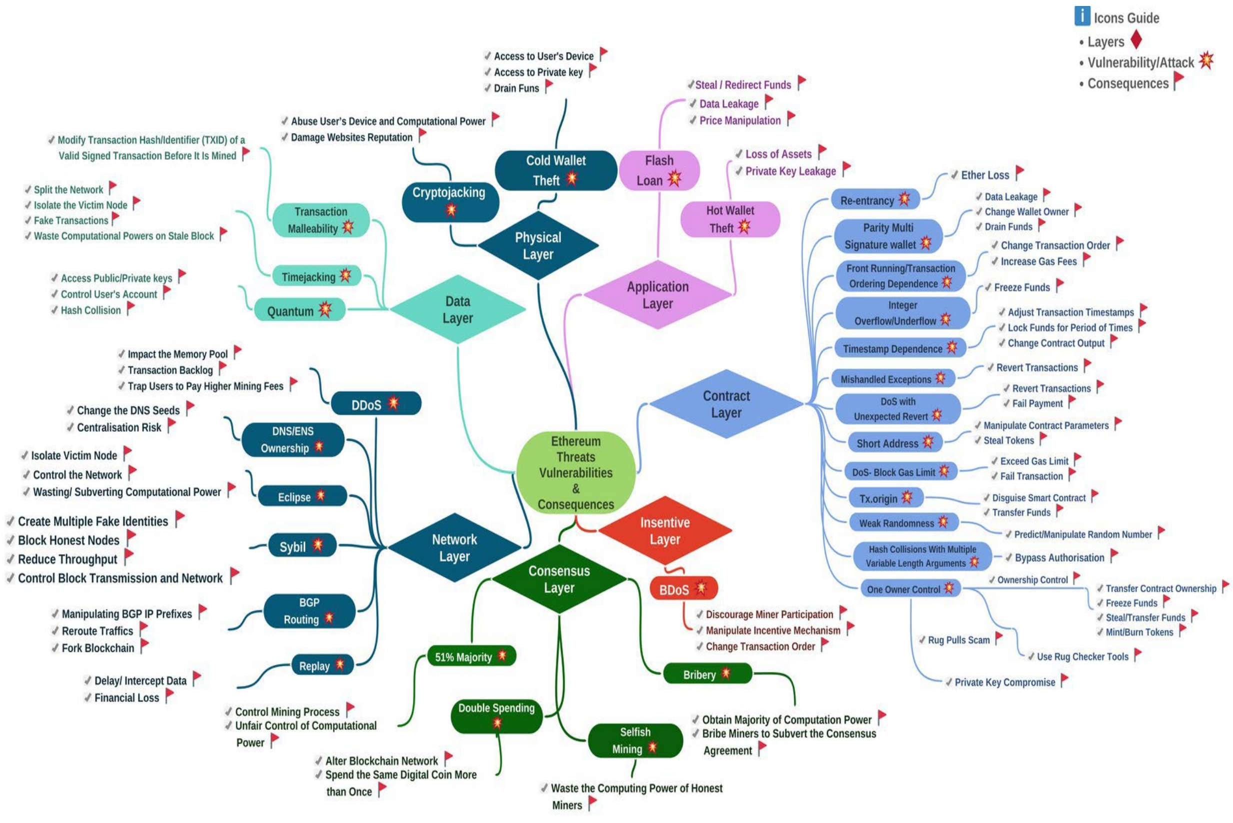


Figure 9 - Vulnerabilities, Attacks and Consequences: a Taxonomy for the 7-layer Architecture

3.12 Centralisation Risks:

Findings from existing works confirmed that centralisation risk is one of the major security concerns that recently raised in Blockchain. Centralisation risks emanate from a different layer of Blockchain. As explained in section 1.5, centralisation risks exist in five layers of Blockchain, such as the application layer, with centralised end user applications which are provided by centralised organisations known as exchanges (Sai *et al.*, 2021). Centralisation also poses a risk within the consensus and incentive layers, which control the consensus power and incentive distributions, which can cause 51% attack and selfish mining (Sai *et al.*, 2021; Xiao *et al.,* 2020). Even in the network layer, centralised DNS services control DNS seed addresses and can cause an eclipse attack (Sai *et al.*, 2021; Li *et al.,* 2021). Centralisation risks also affect the contract layer with “owner control” whereby developers and external attackers can exploit Blockchain through contract’s ownerships (Sai *et al.*, 2021; CertiK, 2022). For smart contracts written in Solidity, although they are meant to be decentralised, developers can exploit the network to inject centralisation into the smart contract. This is the case because when digital assets are in the control of developers/owners, and Blockchain is not sufficiently decentralised, the risk moves to the smart contract itself. This makes smart contracts one of the major areas of security concerns in Blockchain transactions (Xiao *et al.,* 2020; Sai *et al.*, 2021; CertiK, 2022).

3.13 Security Risks Associated with Smart Contracts in the Contract Layer

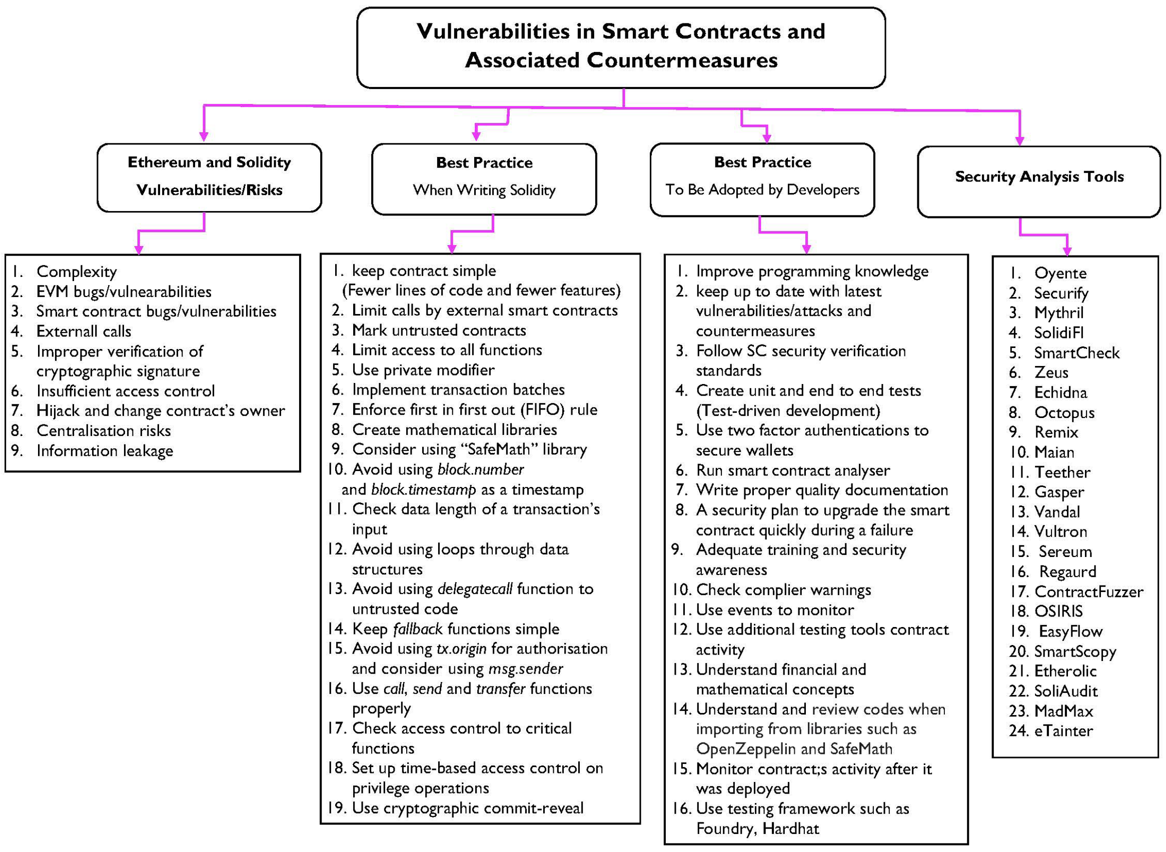
Considering the prevalence of smart contracts and their related security risks, and taking account of the vulnerabilities and attacks within each layer, as outlined in Section 2.7 above, the contract layer is, arguably, the most vulnerable layer in a Blockchain architecture. This is, in part, a consequence of the fact that smart contracts are prone to security vulnerabilities due to the high dependence on programmers and exposure to bugs (Hooper Solorio and Kanna, 2019).

A Smart contract, which runs on Ethereum Blockchain, enhances trust in Blockchain technologies, for it delivers higher trust in the decentralised ledger where data cannot be altered or deleted. Paradoxically, smart contracts are prone to security vulnerabilities due to high dependence on programmers and exposure to bugs (Ma *et al.*, 2019; Rezaeighaleh and Zou, 2019). Based on the nature of Blockchain-based programs, once smart contracts are deployed, they cannot be modified. Therefore, it is vital that developers be familiar with secure coding and best practices, testing tools to enhance the code before deploying smart contract on Ethereum network. It is argued, therefore, that particular attention should be paid to security risks emanating from smart contracts. This section provides a detailed account of current work on security risks and countermeasures associated with smart contracts. This is then used as a basis for developing a more detailed model application for smart contract security risks within the contract layer. The model is described here outlining the best practice towards developing more secure smart contracts.

As smart contracts are still recent, new bugs and security risks are constantly being discovered. This led to developers using several smart contract security tools to check and validate the code and detect some of the vulnerabilities. As described in Section 3.5 above, the literature review revealed that different techniques, or tools, exist to detect vulnerabilities within the contract layers.

To alleviate the risks associated with smart contracts, recommendations include manual code review to detect bugs, check access control to critical functions and the flow of function calls. Researchers also suggested to use testing frameworks like foundry and hardhat to run tests and debug solidity code (Sm4rty, 2022). Source code metrics can be used for quality assurance and performance of Blockchain oriented software (e.g., measure complexity, calculate smart contract resource consumption such as gas in the Ethereum system) (Ajienka, Vangorp and Capiluppi, 2020). SWC Registry provided smart contract weakness classification which includes real-world smart contracts as test cases for each vulnerability (SWC, 2020). It is vital that developers, researchers, and auditors use best practices for secure coding throughout the development process and use available techniques/tools, remediation steps, as suggested by CWE (SWC, 2020), ConsenSys and Mastering Ethereum (Antonopoulos and Wood, 2018) to avoid erroneous implementations and major/critical vulnerabilities.

From the analysis of the work in (Antonopoulos and Wood, 2018; SWC, 2020, and ConsenSys, no date) and the related literature cited in Table 3 above, a model application is developed depicting, in more detail, the security risks within the contract layer. For each vulnerability, the model proposes a best practice to adopt when writing the Solidity Code, best practice to be adopted by developers in general, and suggested analysis tools to use. This model is presented in Figure 11.



Vulnerabilities in Smart Contracts and Associated Countermeasures
Ethereum and Solidity Vulnerabilities/Risks
Best Practice When Writing Solidity
Best Practice To Be Adopted by Developers
Security Analysis Tools
1. Complexity
2. EVM bugs/vulnearabilities
3. Smart contract bugs/vulnerabilities
4. Externall calls
5. Improper verification of cryptographic signature
6. Insufficient access control
7. Hijack and change contract's owner
8. Centralisation risks
9. Information leakage
1. keep contract simple (Fewer lines of code and fewer features)
2. Limit calls by external smart contracts
3. Mark untrusted contracts
4. Limit access to all functions
5. Use private modifier
6. Implement transaction batches
7. Enforce first in first out (FIFO) rule
8. Create mathematical libraries
9. Consider using "SafeMath" library
10. Avoid using *block.number* and *block.timestamp* as a timestamp
11. Check data length of a transaction's input
12. Avoid using loops through data structures
13. Avoid using *delegatecall* function to untrusted code
14. Keep *fallback* functions simple
15. Avoid using *tx.origin* for authorisation and consider using *msg.sender*
16. Use *call*, *send* and *transfer* functions properly
17. Check access control to critical functions
18. Set up time-based access control on privilege operations
19. Use cryptographic commit-reveal
1. Improve programming knowledge
2. keep up to date with latest vulnerabilities/attacks and countermeasures
3. Follow SC security verification standards
4. Create unit and end to end tests (Test-driven development)
5. Use two factor authentications to secure wallets
6. Run smart contract analyser
7. Write proper quality documentation
8. A security plan to upgrade the smart contract quickly during a failure
9. Adequate training and security awareness
10. Check complier warnings
11. Use events to monitor
12. Use additional testing tools contract activity
13. Understand financial and mathematical concepts
14. Understand and review codes when importing from libraries such as OpenZeppelin and SafeMath
15. Monitor contract;s activity after it was deployed
16. Use testing framework such as Foundry, Hardhat
1. Oyente
2. Securify
3. Mythril
4. SolidiFI
5. SmartCheck
6. Zeus
7. Echidna
8. Octopus
9. Remix
10. Maian
11. Teether
12. Gasper
13. Vandal
14. Vultron
15. Sereum
16. Regaurd
17. ContractFuzzer
18. OSIRIS
19. EasyFlow
20. SmartScopy
21. Etherolic
22. SoliAudit
23. MadMax
24. eTainter

Figure 10 - A Model Application for Best Practice Towards a More Secure Smart Contract

3.14 Summary

This chapter provides a comprehensive overview of the security threats and vulnerabilities associated with each layer of a seven-layer blockchain architecture. The inter-relationships between these vulnerabilities, their exploitation and the related consequences are described, with a particular focus on the case of Ethereum Blockchain. With the research questions in mind, a systematic investigation is carried out, covering the mechanisms proposed by researchers to detect/prevent the vulnerabilities and attacks. The outcome of this investigation is summarised in a taxonomy for a seven-layer Blockchain architecture, describing the inter-relationships between vulnerabilities, attacks and the related consequences. Of the seven layers, particular attention is given to the Contract Layer, and more specifically the vulnerabilities associated with how smart contracts are written. A model is proposed to enhance the security of smart contracts, while enlisting the best practices and tools to use.

The security impact of centralisation on the Blockchain is discussed. Major security risks caused by centralisation, particularly within five specific layers of the Blockchain, are identified. An area of continuing interest is related to the potential centralisation that can be caused by smart contracts. Smart contracts with centralised ownership pose major security issues and act as a single point of failure, which contradicts the very decentralised nature of Blockchain. To mitigate against the risks associated with centralised control, decentralised autonomous organisations (DAOs) promise to alleviate some of these risks, by enforcing automated rules that are encoded in smart contracts thus reinforcing the community-based governance. With creating a decentralised decision-making process, the power of decision-making will be distributed and therefore preventing smart contract ownership and ensuring that no single individual, or team, has complete control over the network. For this research, the next step is to develop a method whereby smart contracts are written in such a way to prevent one-owner control and therefore enabling genuine DAO. The next chapter will explain the role of DAOs to reduce the centralisation risks.

Chapter 4: Research Methodology

4.1 Introduction

This chapter presents the research methodology employed in this study, focusing on the centralisation risks associated with smart contracts within Blockchain technology and exploring the merits of Decentralised Autonomous Organisations (DAOs) as a potential solution. The methodology outlined here builds on the research objectives discussed in earlier chapters and provides a detailed explanation of the methods used to design, implement, and test the "Genuine DAO" application. This chapter also sets out the key requirements that will be used to assess the success of this research, ensuring that the objectives are met through a rigorous and systematic approach.

4.2 Research Focus: Centralisation Risks and DAOs

Centralisation within Blockchain networks, particularly in the context of smart contracts, poses significant security risks that can undermine the very decentralisation that Blockchain aims to achieve. As discussed in previous chapters, smart contracts with centralised ownership act as a single point of failure, which contradicts the distributed nature of Blockchain. This research is focused on addressing these centralisation risks by leveraging the concept of DAOs, which decentralise decision-making and governance.

4.2.1 Centralisation Risks Caused by Smart Contracts

The inherent risks of centralisation within Blockchain technology, particularly at the Contract Layer, require careful monitoring, detection, and mitigation. Smart contracts, which are supposed to operate autonomously, often suffer from centralised control due to the way they are deployed and managed. Centralised ownership of smart contracts not only poses a security risk but also contradicts the fundamental principle of decentralisation. This research aims to address these issues by proposing a method to write smart contracts in a way that prevents one-owner control, thereby fostering a truly decentralised environment.

4.2.2 Decentralised Autonomous Organisations (DAOs)

DAOs offer a promising solution to the centralisation risks posed by smart contracts. A DAO is a self-governed, decentralised organisation encoded in smart contracts and deployed on a Blockchain. This self-governed organisation is encoded in smart contracts, deployed, and executed on Ethereum Blockchain (Singh and Kim, 2019). Autonomous governance of DAOs leveraged by automated rules coded within smart contract agreements to facilitate automated and transparent execution. This enhances efficiency and security (Santana and Albareda, 2022).

DAOs tend to differ from existing centralised organisational structures in several aspects. This is mainly due to the fact that DAOs distribute the power of decision-making among all participants with minimum central overall control. All decisions should be approved by most of the DAO members (Santana and Albareda, 2022). DAOs elements are described below.

By distributing decision-making power among all participants, DAOs reduce the risks associated with centralised control and enhance the security, transparency, and efficiency of the system.

- **DAO Rules:** These are the protocols and guidelines encoded in smart contracts that govern the operations of the DAO. They include conditions for membership, decision-making processes, token management, and voting mechanisms, ensuring that the DAO operates transparently and securely.
- **Tokens:** DAO tokens represent ownership or membership in the organisation. They grant holders the right to participate in governance and decision-making, thereby decentralising control within the organisation.
- **DAO Members:** Members of the DAO, represented by tokens, have the right to vote on proposals and contribute to decision-making processes, ensuring that power is not concentrated in the hands of a few.
- **DAO Smart Contracts:** These are the backbone of the DAO, encoding the governing rules and automating operations in a decentralised manner, thereby eliminating the need for a central authority.

4.2.3 Research Hypothesis: The Use of DAOs in Mitigating Centralisation Risks

Based on the literature review and the identified centralisation risks in smart contracts, the following hypothesis is formulated:

> *"The centralisation risks associated with smart contracts can be mitigated by adopting a DAO-based governance model."*

By transferring decision-making authority from a centralised entity to a diverse group of users, DAOs ensure that no single individual or team has complete control over the network.

Numerous studies have explored vulnerabilities within Ethereum and smart contracts in particular (see Chapter 3 above). Furthermore, and over the last three to four years, there have been a great number of works dedicated to examining DAOs. However, there are limited studies that pay particular attention to centralisation risks in DAOs. At the time of writing, the existing literature lacked any work showing how using DAOs would minimise centralisation risks on smart contracts. Therefore, there is a need to delve into the centralisation risks with particular focus on risks related to owner control. In this thesis a decentralised application that is governed by DAOs is proposed. "Genuine DAO" is a decentralised application that has the potential to offer transparency, security, and decentralised decision-making.

In the main, two methods are suggested to minimise centralisation risks, a) hand over privileged roles to multi-signature smart contracts, and b) implementing DAOs whereby the project will be managed by the community that actively participates in it (Kuryłowicz, 2023; Certik, 2021; Code4rena, 2023).

This research proposes the development of a decentralised application called "Genuine DAO," which aims to provide a transparent, secure, and decentralised decision-making process, thereby reducing the risks of centralisation in smart contracts.

4.3 Research Philosophy

Research philosophy encompasses the researcher's beliefs and assumptions about the nature of knowledge creation and the techniques and methods that are used to collect and analyse data (Saunders et al., 2020). Research philosophy deals with the sources of knowledge, its development, and the nature of that knowledge. Knowledge can develop in a particular field and can come from answering a specific problem in a certain context (Collins, 2019).

Al Zefeiti and Mohamad. (2015) believe that every view of research philosophy will then come with a different set of assumptions that are attached to it. With regards to the research onion, they stated that several of the philosophies described in the research onion model require a set of assumptions with different ontology, epistemology and axiology. Of these varying philosophies, positivism, realism, interpretivism and pragmatism are considered to be the most important, which the following section explains in more detail.

4.3.1 Positivism Philosophy

The positivism philosophy is founded on the belief that all knowledge stems from human experiences (Collins, 2019). It is because of this idea that it is also believed that positivism refers to a researcher's own philosophical viewpoint and position. The researcher must be aware of his or her own philosophical beliefs. Saunders et al. (2020) state that it requires working with an observable social reality, so that the final result of this type of research can be the creation of law-like generalisations derived from our own experiences. With this type of research philosophy, there is a need to evaluate results with statistical methods. In general, positivists believe in a single and measurable reality (Patel, 2015). Therefore, the main focus with this type of philosophy is on quantifiable observations and quantitative methods.

4.3.2 Realism Philosophy

The realism philosophy is based on the separation of reality from the human mind (Saunders et al., 2020). The belief is that something can exist even in the absence of us thinking about them. With this philosophy, there is a common understanding and belief that the development of knowledge requires a scientific approach. Realism is often divided into two groups, which are direct and critical. Direct realism believes that what you see is in fact in existence, whereas critical realism believes in certain fixed and set philosophical positions Saunders et al., 2020).

4.3.3 Pragmatism Philosophy

This philosophy believes in using different ways and methods to solve a problem. With a pragmatic philosophy, the researcher is looking for the best possible way to solve the problem. Therefore, a pragmatic researcher may use mixed methods and techniques associated with qualitative and quantitative research (Saunders et al., 2020).

4.3.4 Interpretivism/Constructivism Philosophy

According to Saunders et al. (2020, p. 168), in an interpretivist philosophy “the researcher needs to make sense of subjective and socially constructed meaning expressed about the phenomenon being studied”.

Interpretivism has a conceptual relationship to the philosophical position of idealism and is a term that unites a range of approaches, including social constructivism (Collins, 2019). With the interpretivism philosophy, the main focus is on qualitative results. Interpretivists believe that there is no single reality or truth, and therefore reality needs to be interpreted (Patel, 2015). With this type of research philosophy, the research will be highly dependent on the researcher to observe and interpret a subject or event. It requires good training in overcoming biases and in observational methodology. In interpretivism, common methodologies include observations and interviews.

4.3.5 The Adopted Research Philosophy

The research philosophy underpinning this study is pragmatism, which supports the use of a combination of quantitative and qualitative methods. Pragmatism allows for a flexible approach to research, enabling the researcher to choose the methods that best suit the research problem and objectives.

- **Positivism**: This perspective is applied during the experimental phase of the research, where quantitative methods are used to test hypotheses and collect empirical evidence. The positivist approach is essential for evaluating the outcomes of the "Genuine DAO" application in a systematic and measurable way.
- **Interpretivism**: This perspective is applied during the qualitative phase of the research, where the focus is on understanding the subjective experiences and insights of experts in the field. Interpretivism is crucial for gathering and interpreting the qualitative feedback that informs the development and refinement of the "Genuine DAO" application.

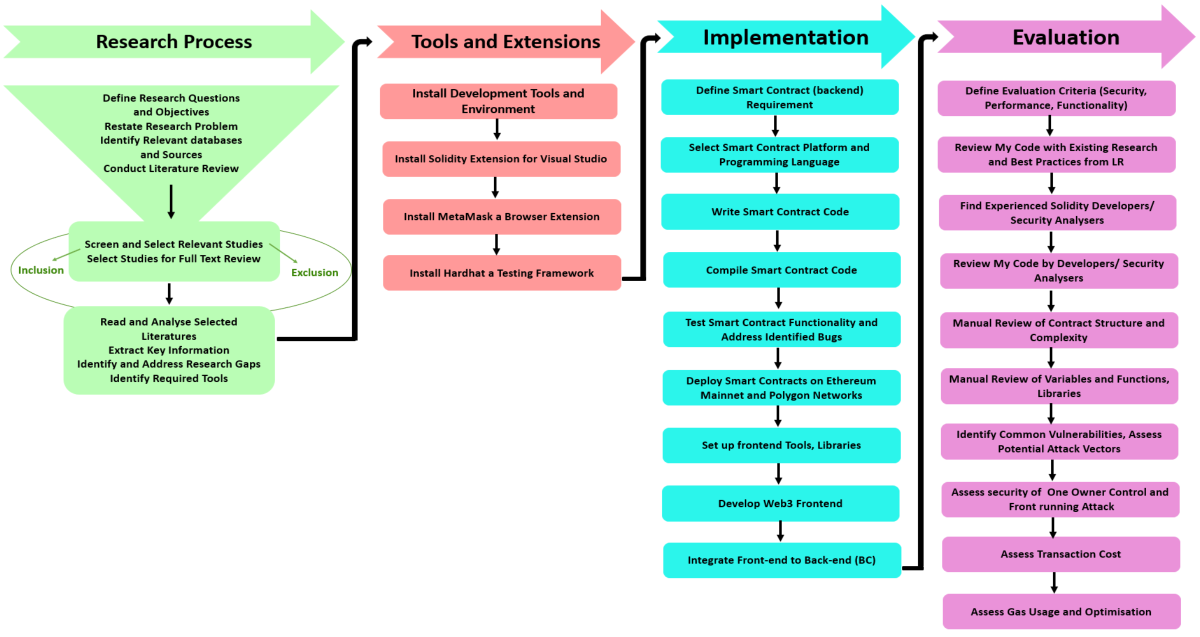
By combining these perspectives, the research can address both the technical and human aspects of the problem, providing a comprehensive understanding of the centralisation risks and the effectiveness of the proposed solution.

4.4 Research Approach

The research approach adopted in this study is deductive, aligning with the goals of scientific research. The deductive approach involves starting with a hypothesis based on existing knowledge and then systematically testing this hypothesis through experiments and data analysis.

- Formulating Hypothesis: Based on the literature review and the identified centralisation risks in smart contracts, a hypothesis is formulated regarding the potential effectiveness of the "Genuine DAO" in mitigating these risks.
- Testing Hypothesis: The hypothesis is tested through a combination of quantitative experiments (e.g., testing the smart contracts within the "Genuine DAO") and qualitative evaluations (e.g., expert feedback).
- Data Analysis: The data collected from the experiments and evaluations are analysed to determine whether the hypotheses are supported or refuted. This approach ensures that the research findings are based on solid empirical evidence.

Figure 11 illustrates a flowchart depicting the research methodology and the detailed steps and key stages involved in conducting the research and achieving the research objectives



Research Process
Define Research Questions and Objectives
Restate Research Problem
Identify Relevant databases and Sources
Conduct Literature Review
Screen and Select Relevant Studies
Select Studies for Full Text Review
Inclusion
Exclusion
Read and Analyse Selected Literatures
Extract Key Information
Identify and Address Research Gaps
Identify Required Tools
Tools and Extensions
Install Development Tools and Environment
Install Solidity Extension for Visual Studio
Install MetaMask a Browser Extension
Install Hardhat a Testing Framework
Implementation
Define Smart Contract (backend) Requirement
Select Smart Contract Platform and Programming Language
Write Smart Contract Code
Compile Smart Contract Code
Test Smart Contract Functionality and Address Identified Bugs
Deploy Smart Contracts on Ethereum Mainnet and Polygon Networks
Set up frontend Tools, Libraries
Develop Web3 Frontend
Integrate Front-end to Back-end (BC)
Evaluation
Define Evaluation Criteria (Security, Performance, Functionality)
Review My Code with Existing Research and Best Practices from LR
Find Experienced Solidity Developers/ Security Analysers
Review My Code by Developers/ Security Analysers
Manual Review of Contract Structure and Complexity
Manual Review of Variables and Functions, Libraries
Identify Common Vulnerabilities, Assess Potential Attack Vectors
Assess security of One Owner Control and Front running Attack
Assess Transaction Cost
Assess Gas Usage and Optimisation

Figure 11 Research Methodology Approach and the Key Stages.

4.5 Data Collection and Analysis Methods

The methodology for data collection and analysis in this study is guided by the Research Onion Model, which provides a systematic framework for selecting and implementing research methods.

4.5.1 Data Collection

The data collection process in this study involves both secondary and primary sources, using a mixed-method approach.

- Secondary Data: A systematic literature review (SLR) was conducted to gather information on Blockchain technology, smart contracts, DAOs, and centralisation risks. The review involved analysing 502 academic papers, conference proceedings, technical reports, and other relevant publications. After applying inclusion and exclusion criteria, 315 articles were selected for detailed analysis, focusing on vulnerabilities, attacks, and preventive techniques related to centralisation in Blockchain.
- Primary Data: Primary data was collected through the development and testing of the "Genuine DAO" application. This involved coding smart contracts, deploying them in a test environment, and gathering feedback from experts. The experts' feedback was collected through structured questionnaires and interviews, providing qualitative data that complements the quantitative analysis.

The steps used in the overall research strategy adopted for selecting the right publications are shown in Figure 12.

Some of the articles within the short list were excluded because they focused on a layering architecture or security threats within platforms other than Ethereum environments. The key inclusion and exclusion criteria are listed in below.

Inclusion criteria:

- The paper must be peer-reviewed and published in research databases.
- The technical report must be reviewed by reputable Blockchain security analysis companies.
- The paper must contain information associated with Blockchain technology or related to Blockchain layering, key components, vulnerabilities and attacks on Ethereum.

Exclusion criteria:

- Papers focusing on business or legal impacts of Blockchain applications.
- Papers focusing on other Blockchain platforms other than Ethereum.
- Papers written in a language other than English.

A thematic analysis was adopted in the identification of meaningful patterns linking threats to each of the Blockchain layers. Although no specific coding was used, the thematic analysis approach was supported by a content analysis, yielding a classification of the key categories around threats, attacks, countermeasures and how they are related to each layer. All data extracted helped to develop a more comprehensive and an in-depth classification of security threats and attacks within the different layers of Blockchain. A complete classification of the available detection tools and preventive techniques was provided for each vulnerability. The main factors that caused centralisation risks are also identified.

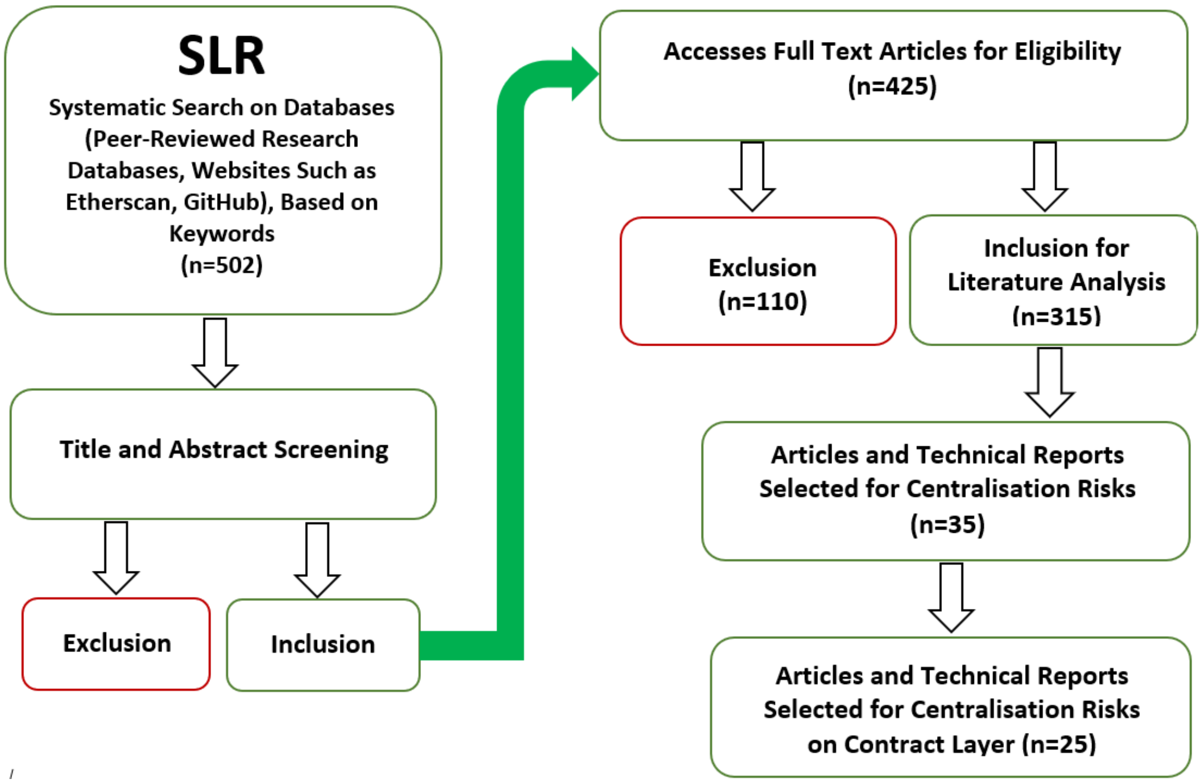

SLR
Systematic Search on Databases (Peer-Reviewed Research Databases, Websites Such as Etherscan, GitHub), Based on Keywords (n=502)
Title and Abstract Screening
Exclusion
Inclusion
Accesses Full Text Articles for Eligibility (n=425)
Exclusion (n=110)
Inclusion for Literature Analysis (n=315)
Articles and Technical Reports Selected for Centralisation Risks (n=35)
Articles and Technical Reports Selected for Centralisation Risks on Contract Layer (n=25)

Figure 12 - Flowchart of Research Strategy Used for Articles and Reports Selection.

4.5.2 Data Analysis

The data analysis process in this study involves both quantitative and qualitative methods, ensuring a comprehensive evaluation of the research findings.

- Quantitative Analysis: The quantitative analysis focuses on the outcomes of the "Genuine DAO" application, particularly its effectiveness in preventing centralisation in smart contracts. Metrics such as the distribution of decision-making power, the security of smart contracts, and the transparency of governance processes are used to evaluate the application.
- Qualitative Analysis: The qualitative analysis involves thematic analysis of the feedback provided by experts. This analysis identifies patterns and insights that inform the refinement of the "Genuine DAO" application, ensuring that it meets the needs of users and addresses the identified centralisation risks.

4.6 Design of the Genuine DAO Application

The data analysis phase informs the design phase, setting out the requirements for the Genuine DAO application and the architecture to adopt in order to address the identified centralisation risks. The focus in this phase is on the design requirements, architecture, and novel aspects of the system. The Genuine DAO is designed to address the centralisation risks associated with smart contracts by enforcing a decentralised governance model and enhancing security through innovative design features. The design is based on this research hypothesis that *“The centralisation risks associated with smart contracts can be mitigated by adopting a DAO-based governance model: Genuine DAO”.*

4.7 Implementation and Testing

The implementation phase of the research involves the development and testing of the "Genuine DAO" application. This phase is critical for demonstrating the practical applicability of the proposed solution and for validating the research findings.

4.7.1 Tools and Extensions

The development of the "Genuine DAO" application involved the use of various Open-Source tools and platforms. GitHub and Etherscan were used to collect and manage the smart contracts' source code, while other tools were used for coding, deployment, and testing.

4.7.2 Implementation Process

The implementation process involved several key steps:

- Designing the Smart Contracts: The smart contracts were designed to enforce decentralised decision-making, with specific measures to prevent one-owner control.
- Deploying the Smart Contracts: The smart contracts were deployed in a test environment within the Ethereum Blockchain, allowing for controlled experimentation and testing.

- Testing the Application: The application was tested for its ability to meet the security, transparency, and decentralisation requirements. This involved both functional testing (to ensure the smart contracts work as intended) and security testing (to identify and mitigate potential vulnerabilities).

4.7.3 Expert Evaluation

Once the testing phase was completed, the "Genuine DAO" application was evaluated by experts in the field of Blockchain and smart contracts. The experts were selected based on their experience and expertise, ensuring that the feedback provided was both relevant and insightful.

- Feedback Collection: Feedback was collected through structured questionnaires, focusing on the effectiveness of the application, potential improvements, and any observed weaknesses.
- Analysis of Feedback: The feedback was analysed to identify common themes and insights, which were then used to refine the "Genuine DAO" application.

4.8 Evaluation Criteria

The success of the "Genuine DAO" application is evaluated based on the following criteria:

- Decentralisation: The degree to which the application prevents one-owner control in smart contracts, ensuring that decision-making is distributed among participants.
- Security: The effectiveness of the application in mitigating security risks associated with centralisation, such as single points of failure and susceptibility to attacks.
- Expert Feedback: The insights and suggestions provided by experts during the evaluation phase, which will be used to assess the practicality and usability of the application.

These criteria are used to measure the effectiveness of the "Genuine DAO" application and to determine whether the research objectives have been met.

4.9 Mapping the Methodology Against the Research Questions

The research methodology applied in this study is designed to comprehensively address the key research questions outlined in Chapter 1. Below is a detailed explanation of how the methodology aligns with each research question.

- **RQ1: What are the current security concerns in Ethereum Blockchain transactions?**
  To address this question, a systematic literature review (SLR) was conducted, focusing on the existing security concerns within Ethereum Blockchain transactions. The SLR involved analysing a wide range of peer-reviewed articles, technical reports, and other relevant sources published between 2015 and 2024. By reviewing and synthesizing the findings from these sources, the study identified the most pressing security issues, such as transaction malleability, double-spending, and vulnerabilities in consensus protocols. The deductive approach used in this study enabled the formulation of hypotheses related to these security concerns, which were then tested through empirical research.

- **RQ2: What types of vulnerabilities are inherent to smart contracts in Blockchain?**
  This research question was addressed by further extending the systematic literature review to focus specifically on the vulnerabilities inherent in smart contracts within Blockchain technology. The review identified various types of vulnerabilities, including re-entrancy attacks, integer overflow/underflow, and logic errors within the contract code. Additionally, the study examined how these vulnerabilities can be exacerbated by centralised control, making them critical targets for the proposed solutions. The deductive research approach facilitated the testing of these identified vulnerabilities through the development and analysis of smart contracts within the "Genuine DAO" application, allowing for a detailed examination of how these issues can be mitigated.

- **RQ3: How can a DAO-based framework enhance the decentralisation and security of smart contracts?**
  To explore this question, the research focused on designing, implementing, and evaluating a DAO-based framework, specifically the "Genuine DAO" application. The methodology involved both quantitative and qualitative approaches. Quantitatively,

the application was tested for its ability to decentralise decision-making processes and enhance security by preventing one-owner control and reducing centralisation risks. Qualitatively, expert feedback was gathered to assess the effectiveness of the DAO-based framework in real-world scenarios. The insights gained from this feedback were instrumental in refining the application and validating its potential to enhance both decentralisation and security in smart contracts.

These methodological steps ensured that each research question was addressed systematically, providing a comprehensive understanding of the security issues within Ethereum Blockchain, the vulnerabilities inherent to smart contracts, and the potential of DAO-based frameworks to mitigate these challenges.

4.10 Summary

This chapter has outlined the research methodology used in this study, focusing on the development of a decentralised application ("Genuine DAO") to mitigate centralisation risks in Blockchain technology. The methodology combines quantitative and qualitative approaches, guided by a pragmatic research philosophy, to ensure a comprehensive and effective investigation. The next chapter will present the detailed implementation steps and the results of the testing and evaluation phases, demonstrating the application’s ability to enhance the security and transparency of Blockchain systems.

Chapter 5: Design and Implementation of the Decentralised Application: Genuine DAO

5.1 Introduction

This chapter presents the design, implementation environment, and testing of the "Genuine DAO" application, a novel approach to mitigating centralisation risks in smart contracts on the Ethereum Blockchain. Drawing on the findings from the literature review and the methodology outlined in Chapter 4, this chapter details the design requirements and the architecture of the Genuine DAO. The design emphasises decentralisation, and security while providing a robust framework for decentralised governance. The chapter also presents an in-depth analysis of system components (front-end and back-end interactions) of the developed application. The end of the chapter is dedicated to describing how the developed application is tested against the design requirements.

5.2 Design Requirements of the Genuine DAO

Starting from the hypothesis that "The centralisation risks associated with smart contracts can be mitigated by adopting a DAO-based governance model: Genuine DAO" the design of the Genuine DAO application is based on a set of critical requirements that stem from the core principles of Blockchain technology decentralisation, security, and transparency. These requirements are designed to address the centralisation risks associated with smart contracts and to provide a robust framework for decentralised governance.

5.2.1. Decentralisation

Requirement 1: Distributed Decision-Making

The application must ensure that decision-making authority is distributed among a broad and diverse group of users, preventing any single individual or entity from exerting undue control. This decentralised governance is essential to maintaining the integrity and democratic nature of the Blockchain, as it mitigates the risks associated with centralised ownership of smart contracts. The architecture of Genuine DAO ensures, through a voting system, that no single party can make unilateral decisions. This ensures that the decision-making authority is distributed among a diverse group of users, preventing any single individual or entity from exerting undue control. This is crucial to maintaining the decentralised nature of the Blockchain and mitigating the risks associated with centralised ownership of smart contracts.

Requirement 2: Elimination of Single Points of Failure

The design must eliminate any single points of failure within the system. A single point of failure, such as a centralised control point in a smart contract, can compromise the security or functionality of the DAO, potentially leading to catastrophic consequences for the entire network. To prevent this, the ownership and governance of smart contracts must be distributed across multiple participants. This decentralisation of power ensures that the system remains resilient even if one or more participants attempt to act maliciously.

This includes the decentralisation of smart contract ownership and the distribution of governance responsibilities across multiple participants.

5.2.2 Security

Requirement 3: Secure Smart Contract Execution

The application must ensure that all smart contracts are executed securely, with rigorous mechanisms in place to prevent vulnerabilities such as one owner control, double proposals and front-running attacks. Front-running, where malicious actors exploit the timing of transactions to gain an unfair advantage, is a significant threat in Blockchain systems. The Genuine DAO design integrates time-based mechanisms to thwart such attacks, ensuring that transactions are executed as intended without interference.

5.3 The Structure of Genuine DAO

Based on the three requirements above, the Genuine DAO application is designed to address the centralisation risks identified in Chapter 3 by enforcing a decentralised governance model through the use of DAOs. This section describes the architecture of the Genuine DAO, highlighting how the design meets the requirements outlined above. The implementation environment is described in detail in section 5.4.

5.3.1 Motivation for Genuine DAO

With the research hypothesis in mind, the development of the Genuine DAO was motivated by the need to mitigate the risks associated with centralised control in Blockchain-based systems. Centralised smart contracts pose significant security risks, acting as single points of failure and contradicting the decentralised ethos of Blockchain technology. To address these issues, the Genuine DAO was designed to ensure that smart contracts are managed through a distributed decision-making process, where power is shared among users and no single entity can dominate the system.

5.3.2 Architecture of Genuine DAO

The architecture of the Genuine DAO is depicted in Figure 13, which illustrates how the application is implemented on the Ethereum network. The architecture is designed to enforce decentralisation and enhance security by integrating the following components:

- **Smart Contracts:** The core of the Genuine DAO is a set of smart contracts that encode the rules and governance processes of the DAO. These contracts are designed to prevent single-owner control, ensuring that all decisions are made collectively by the DAO members.
- **Governance Mechanism:** The governance mechanism of the Genuine DAO is based on a voting system, where members (represented by tokens) can submit proposals and vote on them. The voting process is transparent and recorded on the Blockchain, ensuring accountability.
- **Preventing Front Running Attacks:** To further enhance security, the Genuine DAO detect and prevent front-runners from attempting to prioritise their transactions over other users' transactions and gaining advantage by doing so.

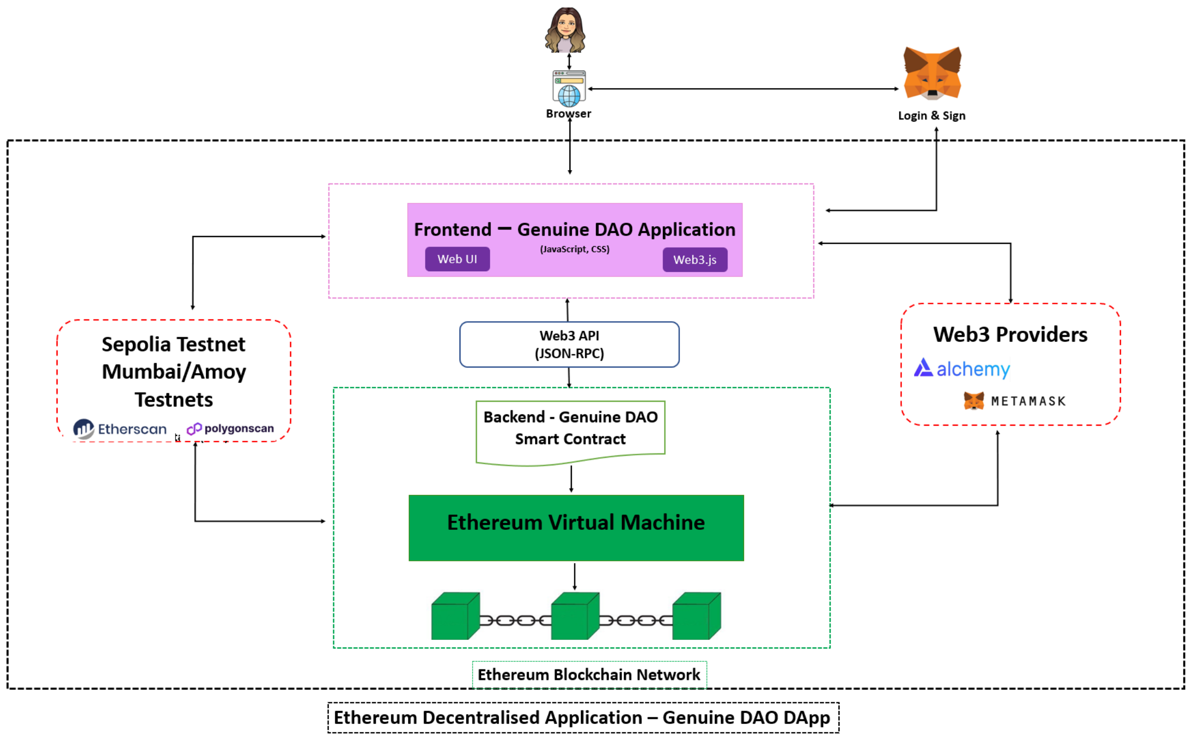


Figure 13 The Architecture of the Proposed Genuine DAO Application.

5.3.3 Novelty of the Genuine DAO Application

The Genuine DAO application introduces two novel elements that distinguish it from existing DAO implementations:

- **Enforced Decentralisation:** Unlike many existing DAOs that still allow for some degree of centralised control, the Genuine DAO is designed to enforce decentralisation by preventing single-owner control of smart contracts. This is achieved through a combination of multi-signature mechanisms and a distributed governance model.
- **Transparent and Auditable Governance:** All governance processes within the Genuine DAO, from proposal creation to voting and execution, are transparent and recorded on the Blockchain. This level of transparency is essential for maintaining trust and ensuring the integrity of the DAO.
- **Focus on Security:** The Genuine DAO places a strong emphasis on security, integrating features such as preventing frontrunning attacks commonly found in smart contracts. This focus on security ensures that the DAO is more resilient to attacks and other threats.

5.4 System Environment Construction and Code Implementation

This section introduces the implementation environment that was used to develop and test the Blockchain-based DAO application “Genuine DAO”. This environment includes a number of tools and platforms that are described below.

The operating environment for developing and testing “Genuine DAO” consists of 16G memory, 11th Gen Intel core i7 processor, and the Microsoft Windows operating system. The proposed decentralised application “Genuine DAO” used Visual Studio Code software as the development environment and the Solidity programming language to write smart contracts for the back-end. The smart contracts are tested using the Hardhat Framework. In addition, JavaScript programming language is utilised within the same development environment for writing the front-end components.

5.4.1 The Flow to Develop Genuine DAO (Back-end and Front-end)

This section describes the steps, the specific tools and the extensions that were used to develop the back-end and the front-end for the proposed blockchain-based DAO application and to enable interaction between them.

5.4.1.1 Develop Genuine DAO Smart Contracts (Back-end)

a. Install required software and extensions such as Visual Studio, IDE and Solidity, and Hardhat extensions.
b. Define the Smart Contract requirements such as data structures, functions, and logic for writing back-end of “Genuine DAO”.
c. Use Solidity compiler to compile smart contracts code into bytecode, which generates the ABI in JSON format.
d. Set up Ethereum network such as web3.js and connect “Genuine DAO” to the Sepolia and Mumbai/Amoy testnets.
e. Deploy the “Genuine DAO” smart contracts on both Ethereum mainnet network (Sepolia testnet) and Polygon network (Mumbai and Amoy testnets).
f. Test and debug “Genuine DAO” smart contracts.

5.4.1.2 Develop Genuine DAO Application (Front-end Interface)

a. Install Node.js extensions to create the back-end environment to interact with front-end.
b. Install web3.js library in the Node.js environment to interact with the deployed Genuine DAO smart contracts.
c. Install Tailwind CSS to easily style and design the user interface.
d. Install MetaMask wallet for developing and testing transaction flow of Genuine DAO smart contract as well as creating accounts to interact with the Genuine DAO application (DApp).
e. Write JavaScript functions to interact with the deployed smart contracts.
f. Create functions in the back-end that use web3.js to read and write data from/to the Genuine DAO smart contracts.
g. Develop Front-end (decentralised application) using JavaScript, React and Tailwind CSS.
h. Set up web3.js to enable communication between the Genuine DAO application (DApp) with the deployed Genuine DAO smart contracts. When the front-end sends data requests to the Genuine DAO smart contracts or makes a transaction, the back-end (Genuine DAO smart contracts) will handle the request using web3.js and response to the front-end (Genuine DAO application). (web3.js retrieves data from the deployed Genuine DAO smart contracts).
i. Set up, or import, the API of Genuine DAO smart contracts into the front-end code to enable the Genuine DAO application to interact with the Genuine DAO smart contracts.
j. Use Wagmi, a Design System, to facilitate interactions with the MetaMask wallet and Genuine DAO smart contracts.
k. Deploy and host the front-end on Ethereum network (Sepolia testnet) and Polygon network (Mumbai/ Amoy testnets).

g. Test Genuine DAO to ensure that the front-end communicates with the back-end correctly through the defined API endpoints and Genuine DAO functions as expected. To check if Genuine DAO handles user input securely and prevent common vulnerabilities.

5.4.2 Development Tools and Frameworks

For the development of the Genuine DAO decentralised application, the following tools and frameworks were used.

5.4.2.1 Programming Languages

- **Solidity to write Genuine DAO smart contracts (back-end)**

  Solidity is a programming language similar to JavaScript, C++ Python. It is the most widely used language for Ethereum smart contract (Solorio *et al.,* 2019). There are three types of variables in solidity that hold values in a program namely, local variable, global variable and state variable. Local variables store temporary data and hold the values during the function execution whereas state variables hold the values permanently, which are stored in smart contracts' storage. State variables have four possible states of visibility (public, private, internal, and external) to access functions and read the value of the state variable. Global variables are accessible from any function within the contract (in the global namespace) and holds specific information about transaction and block such as block number, block timestamp (Bashir, 2020). In this work, the back-end code is written in a Solidity language using Web3.js, ABI and Hardhat.

- **JavaScript for building Genuine DAO Application (Front-end)**

  Javascript is known for being the programming language of choice for web development. The combination of JavaScript and Web3 provides the opportunity for developers to build decentralised applications on the Ethereum Blockchain. To create the Genuine DAO application (DApp), JavaScript was used with help of some libraries and frameworks such as React, web3.js, Node.js, Hardhat and other supporting extensions (Mendes, 2023).

5.4.2.2 Tools and Libraries

A variety of tools and libraries have been used for testing and interacting with Ethereum and Blockchain networks.

Web3.js

Web3.js is a collection of libraries that provides an interface that developers can interact with smart contracts on the Ethereum Blockchain using JavaScript Node.js (Kenneth, 2019). Web3 contains of module constructors such as web3-eth, web3-shh, web3-bzz, web3-net and web3-utils that have specific functionalities to allow users to interact with Ethereum Blockchain and smart contract. Behind the system, Web3 uses JSON-RPC for encoding data, making it easily readable and interoperable across different programming languages and platforms (Solorio *et al.,* 2019; Infura, 2023). JSON-RPC is a remote procedure call mechanism that uses of JSON (JavaScript Object Notation) data format to encode its calls. It allows Blockchain based application interact with smart contracts and Blockchain nodes (Bashir, 2020).

When a developer creates a new contract object, they can provide it with the JSON interface of the respective smart contract. By doing so, Web3.js gains an understanding of the contract's structure and can interpret its functions and data. By passing the JSON interface to Web3.js, it becomes aware of the contract's ABI (Application Binary Interface). This ABI defines the contract's functions, events, and data structures. With this knowledge, Web3.js is able to automatically handle the conversion of high-level function calls into low-level ABI calls over RPC.
This allows developers to enable the library to understand the contract's structure and automatically handle the necessary ABI conversions for function calls in order to build an efficient and robust Blockchain based applications. The Web3.js library can be installing through Node Package Manager (npm)and use “npm install web3” command on terminal console (Kenneth, 2019).

Ethers.js
Ethers.js is a JavaScript library like Web3.js that offers Ethereum interface providers with four modules (Ethers.provider, Ethers.contract, Ethers.utils and Ethers.wallets). These modules interact with Ethereum nodes smart contracts and Ethereum network through JSON-RPC, Alchemy, Etherscan and MetaMask (Ethers, 2023; Infura, 2023).

Node.js
Node.js is a well-known JavaScript runtime environment that can be used in conjunction with Ethereum development to create decentralised applications. There

are several Node.js - related extensions available in the Visual Studio Code, which can be used to improve the development workflow and enables efficient interaction with the Ethereum network (Nguyen, 2023).

React
React is a JavaScript library for building user interfaces (UIs) for decentralised applications. The front-end communicates with the back-end (smart contracts) using the Ethereum Virtual Machine (EVM) and the Application Binary Interface (ABI) (Solorio *et al.,* 2019; MetaOpenSource, 2023).
The ABI acts as a communication bridge between the front-end application and the smart contract. It defines the structure, functions, and events of the contract and how data is read and returned. ABI also specifies the encoding and decoding rules for converting data between the contract's internal representation and external formats, such as bytes or JSON (Hoang Minh, 2022).

Tailwind CSS
This is a popular CSS framework that provides a set of pre-designed low-level utility classes for developers to build modern and responsive user interfaces faster and easier (Ukey, 2022). This CSS framework has gained popularity among front-end developers for its flexibility, reliability, simplicity and ability to speed up the development process by writing less code (Tailwindcss, 2023).

Wagmi
Wagmi is a collection of React Hooks that are used to build a fully functional front-end using React to interact with Ethereum Blockchain. It offers an easy and efficient way to facilitates interactions with crypto wallets and smart contracts and allows to access real-time data updates on changes in the wallet, block and Ethereum network (Wagmi, 2022; QuickNode, 2023).

Chai.js
Chai.js is an assertation library that used for unit testing for any JavaScript testing framework like Hardhat. It provides a set of plugins, functions and methods that allow developers to test the behaviour of code in small, independent units. Chai comes with

different assertion styles such as *should*, *assert* and *expect*. To install Chai in Node.js project, the researcher has used npm and has written the following command “npm install chai” (Chai Assertation library, 2023).

5.4.2.3 MetaMask Wallet to Interact and Deploy Genuine DAO

The MetaMask wallet is a Chrome browser extension that provides the ability for users to create accounts to interact with Blockchain based applications through Json- RPC (JavaScript Object Notation-Remote Procedure Call). MetaMask, enables users to store their digital assets and cryptographic keys, allowing them to securely sign transactions and authenticate their identity when interacting with decentralised applications (Solorio *et al.,* 2019). MetaMask wallet uses the ChainList to connect to the EVM powered networks and Alchemy website to receive Ethereum Sepolia and Polygon Mumbai and Amoy Faucets.

5.4.3 The Genuine DAO Back-end Implementation

5.4.3.1 Setting the Scene for the Back-end

The function *visibility* is a crucial part in the development of a secure smart contract. Functions in smart contracts can be accessible *externally, internally, privately and publicly*. Public functions can be called both internally within the contract or from outside. External functions are visible to other smart contracts and external accounts (EOAs) via transactions. Private functions are only visible within the same contract that defined them. Internal functions are accessible within the contract and any other smart contracts that inherit from it. In addition to *visibility*, solidity offers function modifiers such as *pure, view, payable* and *constant* to enforce specific restrictions on how functions can be accessed within a smart contract. The view modifier disables any modifications to state and allows the function reads the contract's state (Read-only access and returns values). The pure modifier executes locally and prohibits reading and modifying the state (returns calculations). Payable functions allow the function receives and manages Ether with a call. Finally, the Constant Modifier is similar to the View Modifier which disallows access or modification to the state (Bashir, 2020). As a result, functions and variables can be restricted with some “key words” and change the rules of scope to implement access control mechanism to enhance security, improve code’s clarity and reduce gas consumption for read-only operations (gas efficiency).

After Installing the necessary software, tools and extensions, the new project directory has been created on Visual Studio Code, called Genuine DAO Contracts. Figure 16 depicts the proposal structure that is developed by Genuine DAO smart contracts.

This project implemented an application based on a DAO structure where rules are established within the smart contracts. Back-end includes five smart contracts named Age.sol, DAOLib.sol, NFTContract.sol, GenuineD.sol and GenuineDStorage.sol.

- Age.sol is a contract to set the proposal's transaction fees that will be controlled by the contract owner.
- DAOLib.sol is a contract that responsible to provides "structs" and "enums" for GenuineD smart contract.
- NFTContract.sol is responsible for creating and minting tokens for users and give them permission to contribute to voting process.
- GenuineD.sol is the main contract that includes creating proposal, voting, timelock and execution process.
- GenuineDStorage.sol is interacting with GenuineD contract for managing proposals and checking the state of each proposal.

The above five smart contracts are described in more detail in Section 5.5.2 below. With the implementation of a decentralised governance model (DAO), decisions are made by a group of participants rather than a central authority which minimises centralised ownership structure. Governance rules are coded into smart contracts that enforce automated and transparent process. This project, not only has added DAO structure to the Blockchain based application, it is developed in a way that transfers control of smart contracts from contract creator/owner to another smart contract to mitigate one owner control and centralisation risks. More importantly, time-based security feature (timelock) has been implemented in this project. As we explained in chapter 3, section 3.5.13.5, implementing time-based mechanism adds an extra layer of security, temporary locks feature helps to restrict access to a smart contract specifically sensitive functions (Mou, Coblenz and Aldrich, 2021, CertiK, 2021).

The following figure 14 illustrates the process of the proposal's completion within the proposed decentralised application "Genuine DAO".

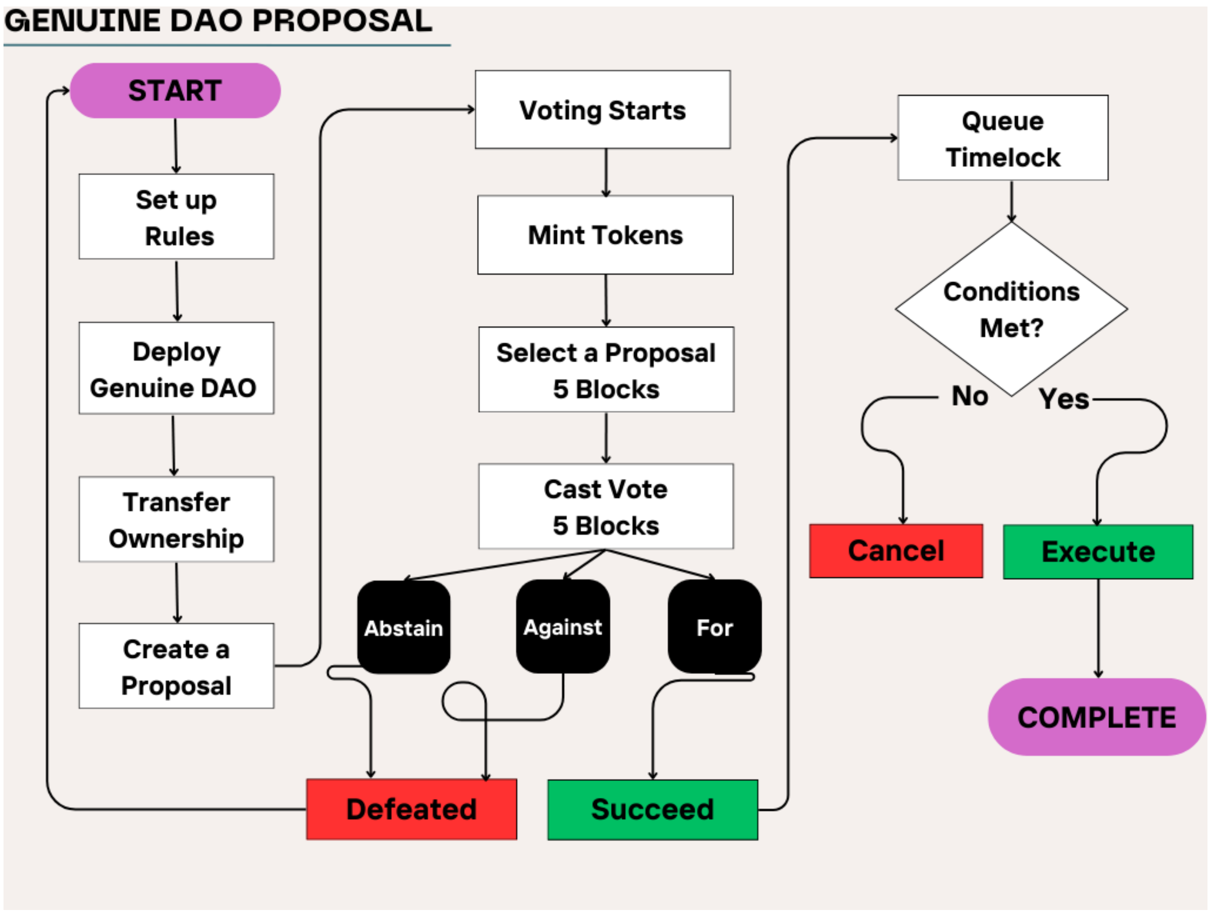


Figure 14 - A Proposal Cycle Within a Genuine DAO Structure.

After developing, adding all rules and terms and conditions and deploying the "Genuine DAO" on Ethereum networks (Ethereum Mainnet and Polygon), the decentralised application is ready to broadcast messages, mint tokens and create proposals for different projects.

The key part is the ownership of the deployed decentralised application which by default would be the contract developer or whoever has deployed it. In the "Genuine DAO" decentralised application, there is a contract called "Age" which includes *onlyOwner* modifier for a function that changes the transaction rate fee. If the owner is the contract developer who deployed the smart contract onto the network, or someone else who received ownership rights, they would be able to control the system and change the transaction rate fee. To mitigate against the one owner control and minimise the risk of enforced centralisation, this decentralised application is designed in a way to transfer ownership to the “GenuineDAO”. Therefore, the system automatically enforces “GenuineDAO” as an owner to change the transaction rate fee through a fairer voting process.

The following steps explain the voting process within “Genuine DAO”, a process which participants need to follow in order to change the transaction rate fee for each project. A time-based restriction is placed on the voting period and applies from the time the proposal starts until voting ends. Each proposal will expire after 200 blocks. It means that the proposal is expected to go through its lifecycle within a maximum of 200 blocks. If the proposal process takes longer than 200 blocks, it will be deactivated automatically.

Process of Working with Genuine DAO

- Connect the crypto wallet, MetaMask to Genuine DAO DApp
- Mint a token for voting on a proposal as a decision-making capability is powered by a native token within the DAO.
- Create a proposal: participants or token holders need to submit their proposals to change the transaction rate fee for a specific project. They have to wait until the status of created proposal become active from pending.
- Delegate: participants or token holders have the option to delegate their voting power to another address before starting voting process. Voting delay will apply for five blocks from creating a proposal until to start voting.
- There are three options to cast votes which include “for”, “against” and “abstain”. Voting delay will apply for five blocks during the voting until queueing starts.
- Voting ends by one of two options, “succeed” or “defeated”.
- Successful proposal moves into the queue step and will be queued in the timelock.
- A proposal can be executed after the Timelock waiting period has elapsed and all conditions are met.
- The proposal will be terminated after 200 blocks.
- The suggested transaction rate fee will be accepted after voting process and execution.
- Any change in the transactions rate fee will require the submission of a new proposal.

5.4.3.2 Key Components (Smart Contracts) to Enhance Security and Minimise Centralisation

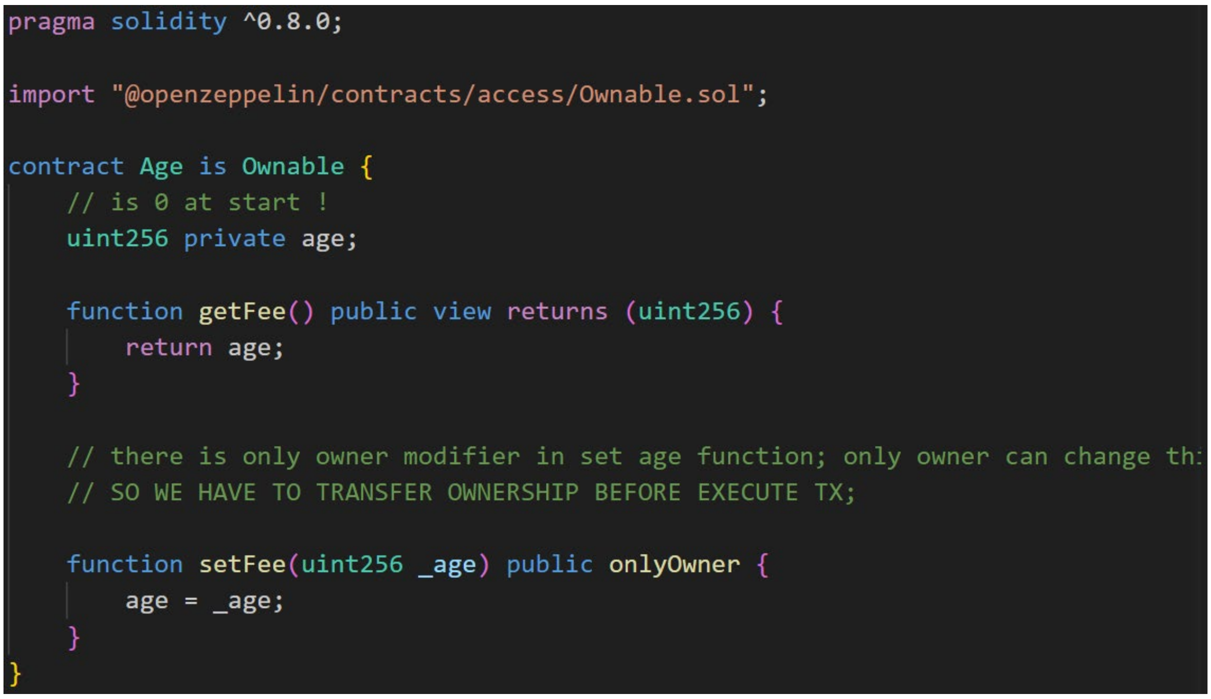
The following sections explain the most important functions of five smart contracts named Age, DAOLib, NFTContract, GenuineD, GenuineDStorage, which are written to enhance security and minimise centralisation.

Age.sol

This contract is responsible to change the transaction rate fee in this project. It includes *Age* or *fee* variable which by default is 0. There are two functions *getFee* and *setFee* that are responsible to return fee and change fee respectively. *OnlyOwner* modifier has been used in *setFee* function. It means only the owner/developer can change this value (fee).

In this smart contract, the *Ownable* feature imports from OpenZeppelin for implementing ownership of all smart contracts. Based on available code in this contract, the owner of ownable contract is the account that deployed this blockchain-based application on Ethereum networks (Ethereum Mainnet and Polygon) which would be able to control the decentralised application by changing the ownership, and the value of transaction rate, creating and minting tokens.

With this contract, the owner can abuse the power to control the system and adjust transaction fees. Manipulating fees, not only discourage participation in the network, also may cause centralised risk. The figure 15 shows two mains function within Age contract.

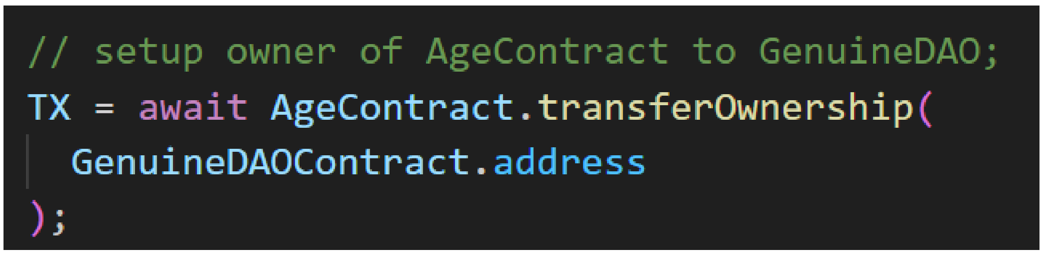


```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/Ownable.sol";

contract Age is Ownable {
    // is 0 at start !
    uint256 private age;

    function getFee() public view returns (uint256) {
        return age;
    }

    // there is only owner modifier in set age function; only owner can change th:
    // SO WE HAVE TO TRANSFER OWNERSHIP BEFORE EXECUTE TX;

    function setFee(uint256 _age) public onlyOwner {
        age = _age;
    }
}
```

Figure 15 - Age Contract with Main Functions.

To mitigate this risk, this program is written in a way that not only use DAO structure, also enforces to transfer the ownership of this contract to Genuine DAO contract after deploying it on Sepolia and Polygon networks. it means the "Genuine DAO" take the ownership control (one owner control) from developer/owner and transfer it to another smart contract to ensure the decentralised structure of Blockchain.

The following figure 16 shows how the ownership from "AgeContract"" would transfer to "GenuineDAOContract". It means after transferring the ownership, the developer or anyone who deployed AgeContract is not the owner anymore and will not be able to change the transaction rate fee. GenuineDAOContract takes the ownership control and would be able to change the transaction rate fee through voting process.



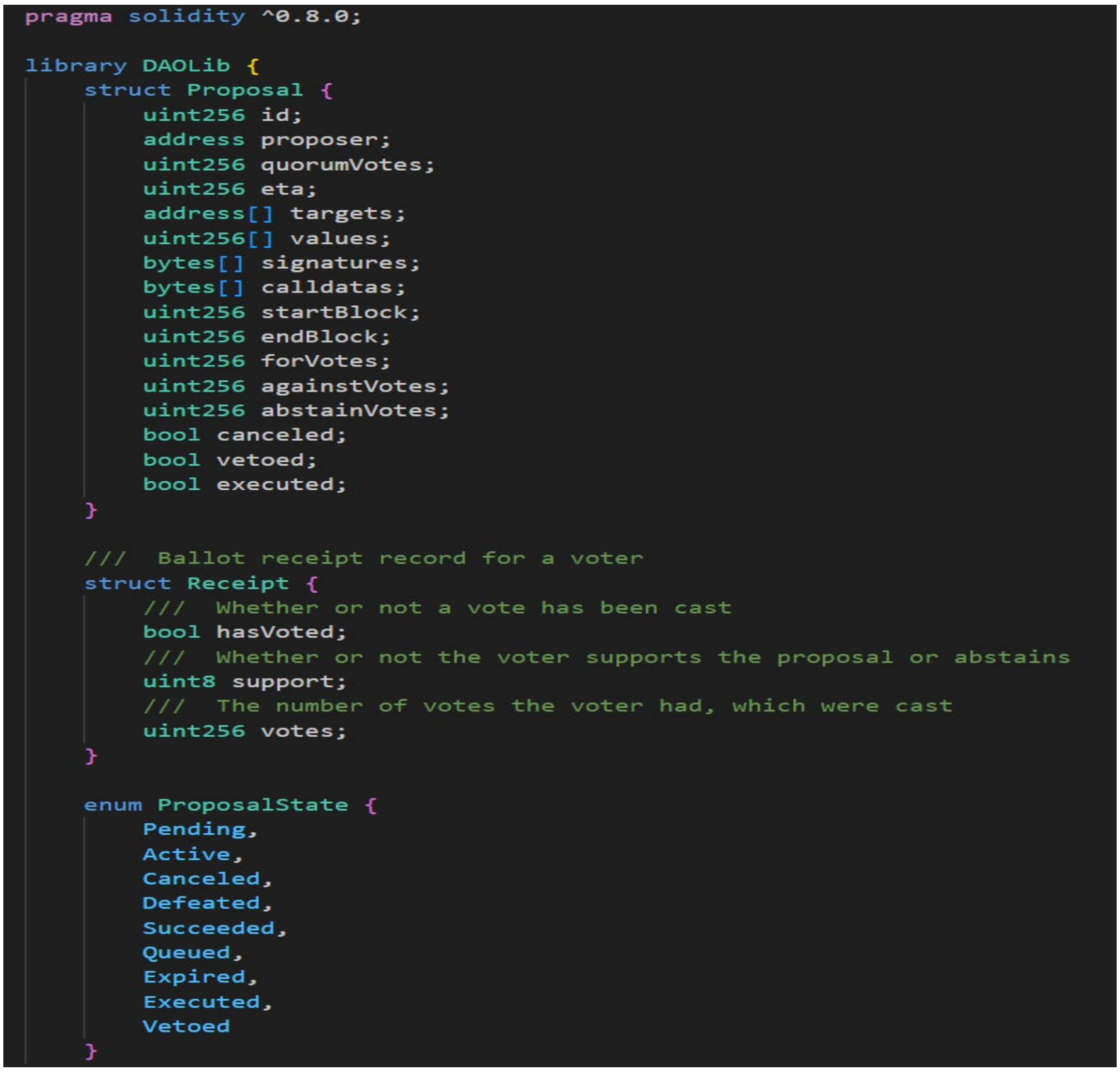
```
// setup owner of AgeContract to GenuineDAO;
TX = await AgeContract.transferOwnership(
  GenuineDAOContract.address
);
```

Figure 16 - Transfer Ownership from AgeContract to GenuineDAOContract

DAOLib.sol

This contract includes “struct” and “enum” and is that responsible to represent a data structure and create a set of named values that restrict a variable to have only one predefined value which enhance code readability and maintainability. The “struct Proposal” represents a data structure and hold multiple variables with different data types related to a proposal. The “struct Receipt” holds a voter’ receipt for voting on a proposal. To check if a vote has been cast, if the voter supports the proposal or not and checking the number of votes.

“enum ProposalState” represents a set of possible proposal states such as “pending”, “active”, “cancelled”, “defeated”, “succeeded”, “queued”, “expired”, “executed” and “voted” that can be used for managing and tracking the state of specified proposal with its proposal Id. Figure 17 represents “struct” and “enum” for GenuineD smart contract.



```
pragma solidity ^0.8.0;

library DAOLib {
    struct Proposal {
        uint256 id;
        address proposer;
        uint256 quorumVotes;
        uint256 eta;
        address[] targets;
        uint256[] values;
        bytes[] signatures;
        bytes[] calldatas;
        uint256 startBlock;
        uint256 endBlock;
        uint256 forVotes;
        uint256 againstVotes;
        uint256 abstainVotes;
        bool canceled;
        bool vetoed;
        bool executed;
    }

    ///  Ballot receipt record for a voter
    struct Receipt {
        ///  Whether or not a vote has been cast
        bool hasVoted;
        ///  Whether or not the voter supports the proposal or abstains
        uint8 support;
        ///  The number of votes the voter had, which were cast
        uint256 votes;
    }

    enum ProposalState {
        Pending,
        Active,
        Canceled,
        Defeated,
        Succeeded,
        Queued,
        Expired,
        Executed,
        Vetoed
    }
```

Figure 17 - DAOLib Contract with Struct and Enum.

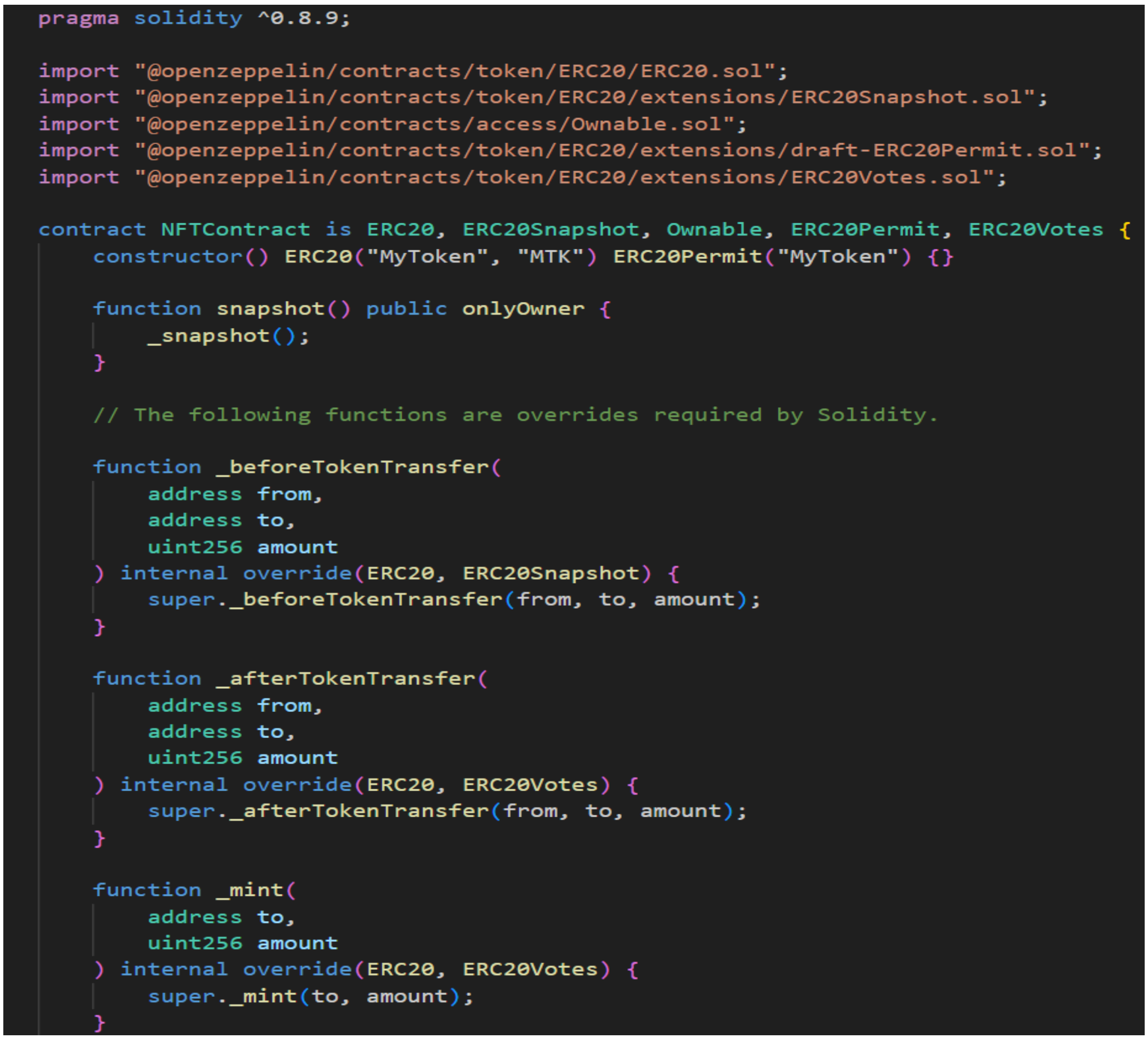
NFTContract.sol

This contract is responsible for creating and minting tokens for users and gives them permission to contribute to voting process. As part of DAO structure, voters need to have token to participate in the decision-making process.

This contract imports contracts from OpenZeppelin that are extensions of the standard ERC20 token to create and manage ERC20 token for staking and voting rights.

There are some keywords in solidity that can be used of contract inheritance such as "override".

"*_beforeTokenTransfer*" and "*_afterTokenTransfer*", "*mint*" functions include the override keyword which represent they are inheriting the IERC20, ERC20Snapshot and ERC20Votes contracts. These functions are intended to override a function in parent contracts with internal access level. It means the function can only be called from within the current contract (NFTContract) and its derived contracts. Figure 18 represents the NFTContract with functions that are inherited from other smart contracts.



```
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Snapshot.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";

contract NFTContract is ERC20, ERC20Snapshot, Ownable, ERC20Permit, ERC20Votes {
    constructor() ERC20("MyToken", "MTK") ERC20Permit("MyToken") {}

    function snapshot() public onlyOwner {
        _snapshot();
    }

    // The following functions are overrides required by Solidity.

    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal override(ERC20, ERC20Snapshot) {
        super._beforeTokenTransfer(from, to, amount);
    }

    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal override(ERC20, ERC20Votes) {
        super._afterTokenTransfer(from, to, amount);
    }

    function _mint(
        address to,
        uint256 amount
    ) internal override(ERC20, ERC20Votes) {
        super._mint(to, amount);
    }
```

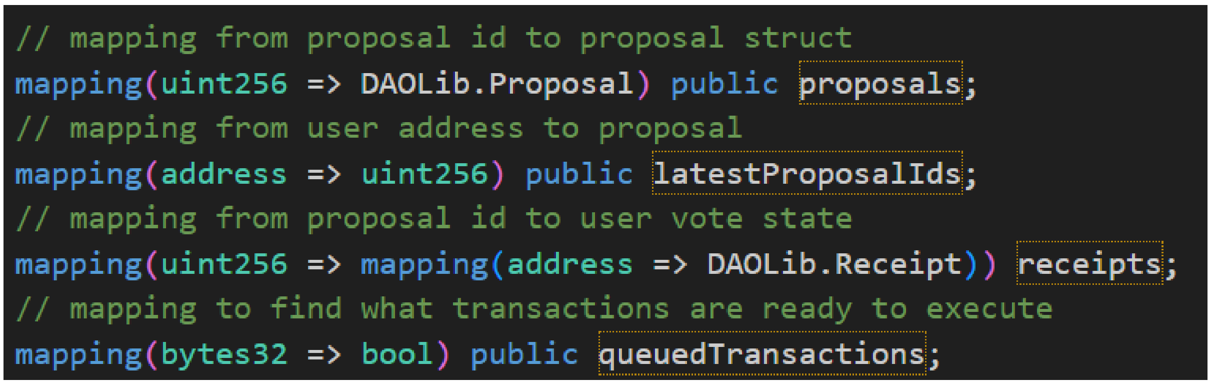
Figure 18 - NFTContract with Override Functions.

GenuineDStorage.sol

This contract works as a data storage for the main GenuineD contract. It is interacting with GenuineD contract for managing proposals and checking the state of each proposal. It also contains various functions such as setter and getter functions, state variables related to the storage and management of data.

Getter functions are used to retrieve the value of state variables directly from storage where as setter functions are used to update or modify the values of state variables in a smart contract (Ethereum.org Team, 2023). Mappings are used to store data as key-value pairs that allow efficient data retrieval based on a specific key. Each key is associated with a single value (Academy, 2023).

Figure 19 shows mappings that are used in GenuineDStorage smart contract.



```
// mapping from proposal id to proposal struct
mapping(uint256 => DAOLib.Proposal) public proposals;
// mapping from user address to proposal
mapping(address => uint256) public latestProposalIds;
// mapping from proposal id to user vote state
mapping(uint256 => mapping(address => DAOLib.Receipt)) receipts;
// mapping to find what transactions are ready to execute
mapping(bytes32 => bool) public queuedTransactions;
```

Figure 19 - GenuineDStorage Contract – Mapping.

The proposals mapping stores details about proposals based on their unique IDs. Users can use the proposal ID to retrieve details about a specific proposal.

The second mapping helps users to find the ID from the most recent proposal they submitted.

The receipts mapping keeps track of users' voting states (receipts) for different proposals.

Lastly, the queuedTransactions mapping is used to identify transactions that are ready to be executed.

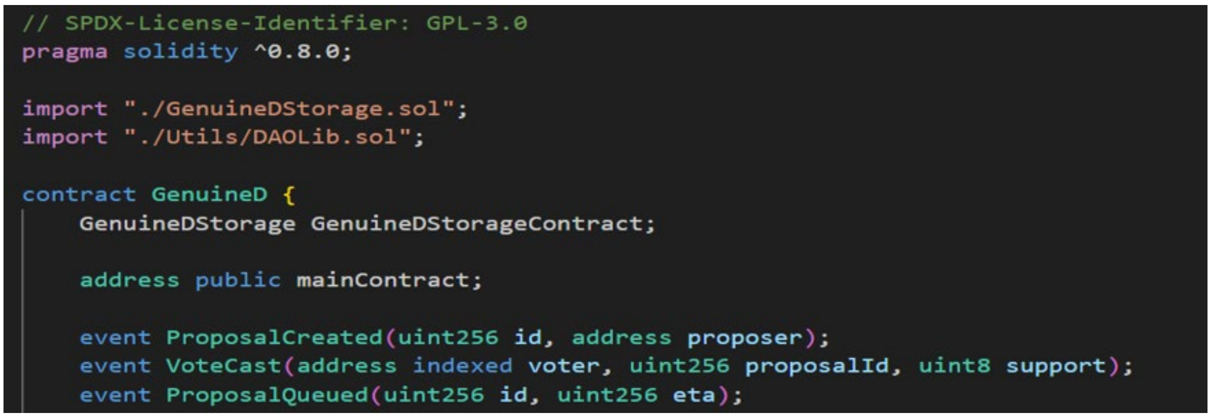
GenuineD.sol

This contract is the main contract that includes creating proposals, voting, timelock and execution process. It starts with three "events" that used to log specific occurrences within the smart contract. Event is an inheritable member of a contract and essential mechanism to communicate with external application or user interfaces. Events can be call just like functions and the "emit" keyword is used to call/dispatch events. This allow developers to observe when an event or function is being called and being informed about important state changes. In addition, events like functions accept arguments. However, arguments are stored in the transaction's log that is not accessible to smart contracts whereas functions store data in the smart contract. The EVM has a logging function that is used to store the data emitted by events which makes it accessible to external applications or user interfaces (Alchemy, 2022).

Event "*ProposalCreated"* emitted when a new proposal is created. It includes ID (identifier of the newly created proposal within the smart contract) and address of proposer (the Ethereum address of the account that created the proposal). This event notifies external application when a new proposal is created in the smart contract.

Event *"VoteCast"* emitted when a vote has been cast on a proposal. It includes three parameters such as proposal ID which is unique identifier of a proposal, voter parameter that stores the Ethereum address of the account that cast the vote and support is a parameter used to count votes. It represents as 0 for against vote and 1 for vote and 2 for Abstain. As a result, users can monitor the voting activities and outcomes via this event.

Event *"ProposalQueued"* emitted when a proposal has been queued in the Timelock. It includes two parameters ID which is used to store the identifier of the queued proposal and eta (Estimated Time of Arrival) that represents a timestamp when the proposal is expected to be executed or processed. Therefore, users can monitor the status of queued proposal. Figure 20 shows available events that are written in GenuineD contract.



```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

import "./GenuineDStorage.sol";
import "./Utils/DAOLib.sol";

contract GenuineD {
    GenuineDStorage GenuineDStorageContract;

    address public mainContract;

    event ProposalCreated(uint256 id, address proposer);
    event VoteCast(address indexed voter, uint256 proposalId, uint8 support);
    event ProposalQueued(uint256 id, uint256 eta);
```

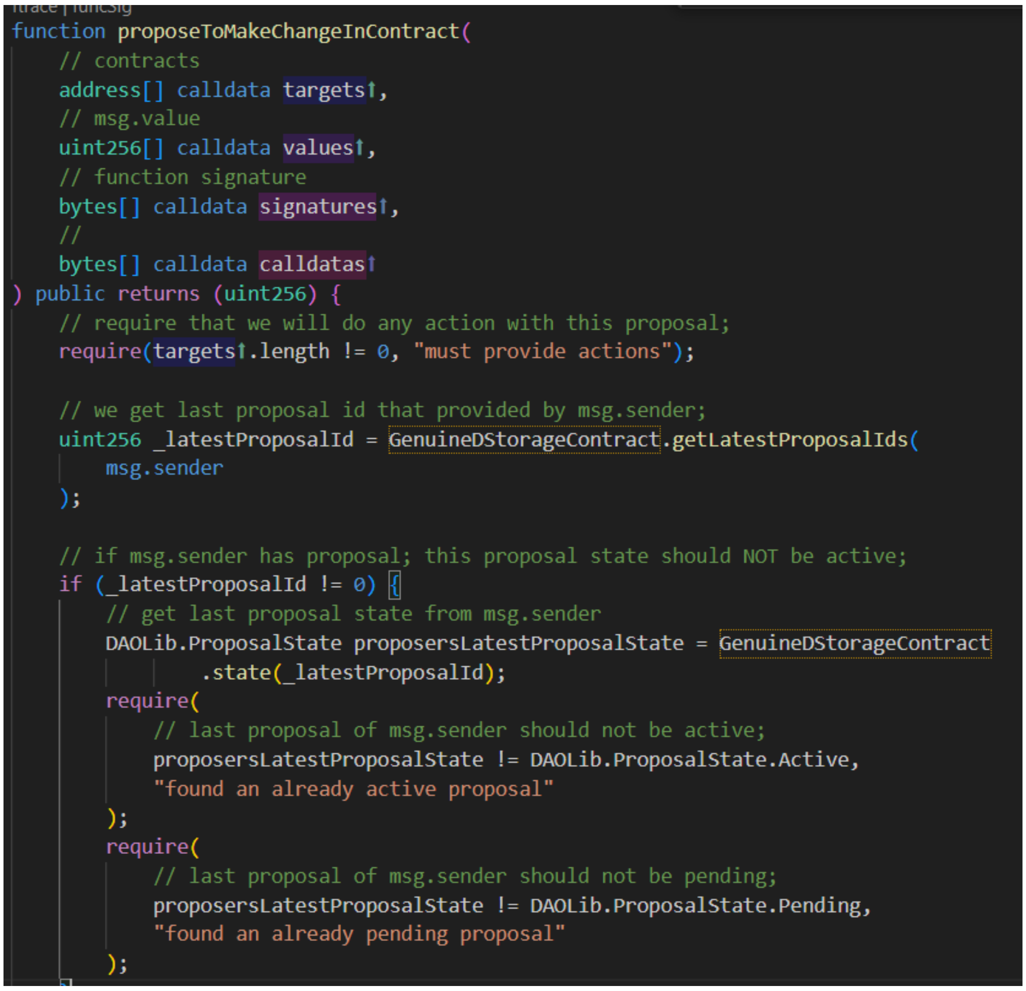
Figure 20 - GenuineD contract – Events.

5.4.4 Creating a Proposal

The following explains the process of a proposal creation to make changes in the value of transaction rate per a project. Each proposal has its own unique ID which help with tracking the status of a proposal.

The GenuineDAO as an owner would assign a specific value for a transaction rate. Participants/proposers will submit their proposals about changing the transaction rate. This will go through voting process to accept to reject the proposal. The voters need to mint tokens first in order to be able to contribute to voting process.

The system automatically checks the state of the latest proposal by the proposer's address, *msg.sender,* to ensure that the current address has not an active and pending proposal. Then the system let the current address to create a proposal. The codes are available in figure 21.



```
ftrace | funcSig
function proposeToMakeChangeInContract(
    // contracts
    address[] calldata targets,
    // msg.value
    uint256[] calldata values,
    // function signature
    bytes[] calldata signatures,
    //
    bytes[] calldata calldatas
) public returns (uint256) {
    // require that we will do any action with this proposal;
    require(targets.length != 0, "must provide actions");

    // we get last proposal id that provided by msg.sender;
    uint256 _latestProposalId = GenuineDStorageContract.getLatestProposalIds(
        msg.sender
    );

    // if msg.sender has proposal; this proposal state should NOT be active;
    if (_latestProposalId != 0) {
        // get last proposal state from msg.sender
        DAOLib.ProposalState proposersLatestProposalState = GenuineDStorageContract
                .state(_latestProposalId);
        require(
            // last proposal of msg.sender should not be active;
            proposersLatestProposalState != DAOLib.ProposalState.Active,
            "found an already active proposal"
        );
        require(
            // last proposal of msg.sender should not be pending;
            proposersLatestProposalState != DAOLib.ProposalState.Pending,
            "found an already pending proposal"
        );
```

Figure 21 - Create a proposal and check the proposal status.

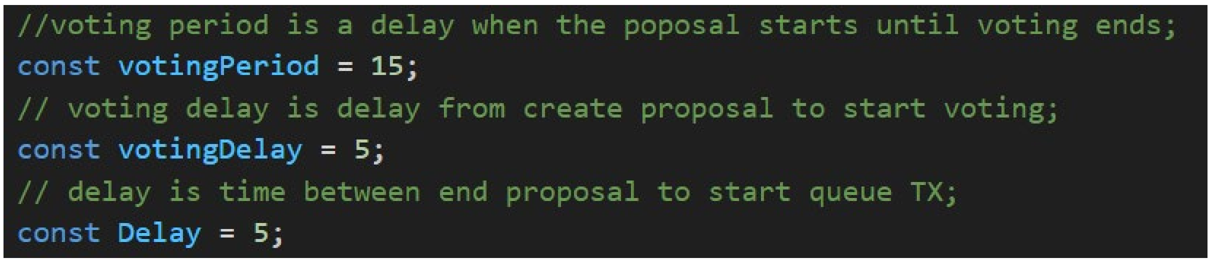
In Genuine DAO DApp, Timelock has been implemented which introduces delays in different steps of voting process. This mechanism is explained below.

Genuine DAO – Timelock

- **votingDelay**, a delay between the proposal creation and the actual start of the voting period.
- **votingPeriod**, a delay when the proposal starts until voting ends.
- **delay**, is the time between the end of proposal (the voting period for a proposal ends) and the start of queuing the transaction.

By specifying an “Timelock” for certain critical functions including fund transfers or significant governance decisions, a delay time between the submission of the action and its execution provides the opportunity for token holders/participants to review their actions and exit the system if they are not agreed with the decision before it is executed. Timelock adds an additional layer of security for any malicious activities such as rug pull (Shevko *et al*., 2023; Traore, 2023).

For the purpose of this research, VotingDelay and Delay have been set to 5 blocks and votingPeriod have been set to 15 blocks which are available in figure 22.



```
//voting period is a delay when the poposal starts until voting ends;
const votingPeriod = 15;
// voting delay is delay from create proposal to start voting;
const votingDelay = 5;
// delay is time between end proposal to start queue TX;
const Delay = 5;
```

Figure 22 - TestDAO.test.js- Timelock.

Each proposal has its own start time and end time which will be calculated by block numbers. The proposal start time includes the current block number plus the voting delay.

$$ProposalStartTime = CurrentBlockNumber + VotingDelay$$
$$ProposalEndTime = ProposalStartBlockNumber + VotingPeriod$$

- *ProposalStartTime* is the timestamp when the proposal starts.
- *ProposaEndTime is the time the vote for a proposal will get close.*
- *CurrentBlockNumer* is the number of the current block which will be available on “GenuineDAO” from Etherscan Testnet.
- *ProposalStartBlockNumber* is the block number when the proposal starts.
- *VotingDelay* is the delay before the proposal starts being voted on. It sets for 5 blocks in this project.
- *VotingPeriod* is a delay when proposal starts until voting ends. It sets for 15 blocks in this project.
- *Delay* is the time when proposal ends to start queuing. It sets for 5 blocks in this project.

E.g., if the current block number is 10, and voting delay is 5 blocks, then start block for this proposal is block 16.

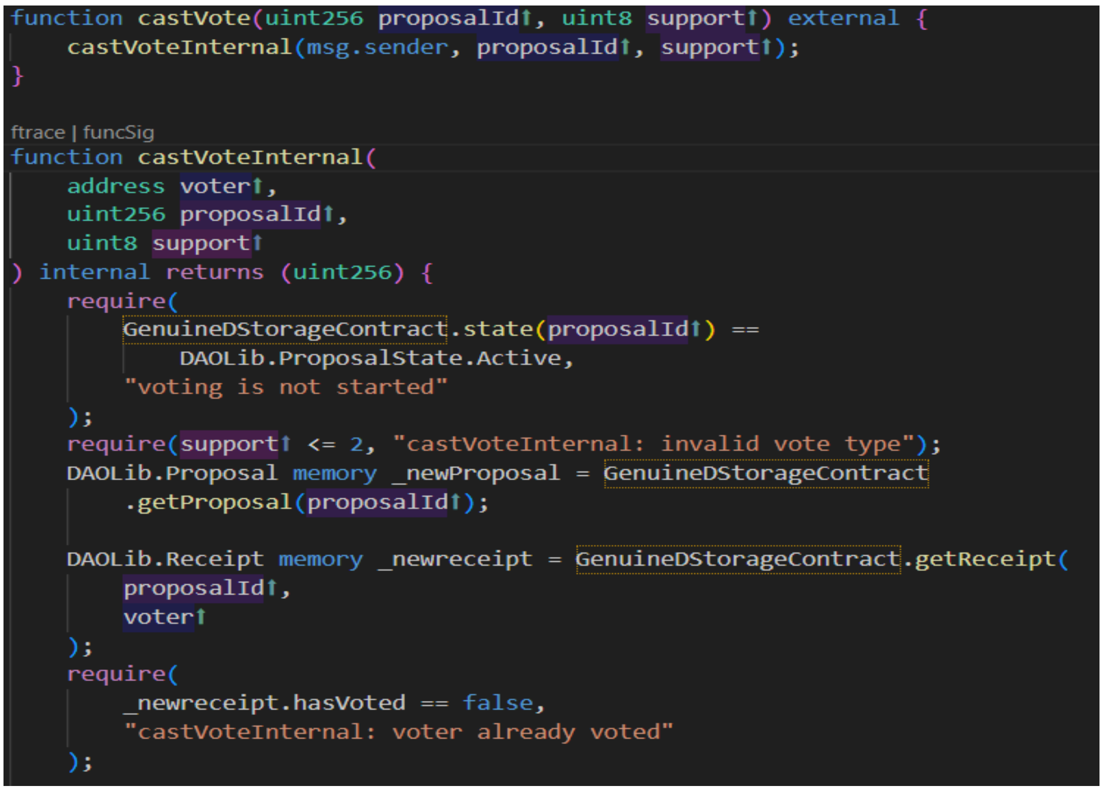
5.4.5 Voting for a Proposal

For the voting process, the "Genuine DAO" checks the following conditions and if all conditions are met, the system will let the voter starts voting process for a period of time.

- The validation of proposal ID
- The block numbers (the block numbers and delay time explained in section 5.4.4)
- The state of proposal to ensure if the created proposal is in an active state or pending
- The voter has already voted or not

The next step is to mint a token if voters/token holders do not already have it to be able to participant in voting process.

Figure 23 shows voting functions include Function *castVote which* is an external function and can be called by voters to cast their votes on a specific proposal. Voters can delegate their vote internally by using *castVoteInterna*l.



```
function castVote(uint256 proposalId↑, uint8 support↑) external {
    castVoteInternal(msg.sender, proposalId↑, support↑);
}

ftrace | funcSig
function castVoteInternal(
    address voter↑,
    uint256 proposalId↑,
    uint8 support↑
) internal returns (uint256) {
    require(
        GenuineDStorageContract.state(proposalId↑) ==
            DAOLib.ProposalState.Active,
        "voting is not started"
    );
    require(support↑ <= 2, "castVoteInternal: invalid vote type");
    DAOLib.Proposal memory _newProposal = GenuineDStorageContract
        .getProposal(proposalId↑);

    DAOLib.Receipt memory _newreceipt = GenuineDStorageContract.getReceipt(
        proposalId↑,
        voter↑
    );
    require(
        _newreceipt.hasVoted == false,
        "castVoteInternal: voter already voted"
    );
```

Figure 23 - Voting Process and Checking Conditions.

As part of a time-based mechanism, different delays implemented to offer opportunity to participants with a window of time to buy tokens, delegate their vote and review proposals during voting process.

There are setter and getter functions that are written in “GenuineDStorage” smart contract. These functions are used to set the value and retrieve the values of variables stored in a smart contract.

Setter functions such as *setLatestProposalIds, setDelay, setVotingDelay, setVotingPeriod and getter functions including getLatestProposalIds, getDelay, getVotingDelay, getVotingPeriod.* including *setVotingDelay* and *getVotingDelay* are written in “GenuineDStorage” smart contract.

Example 1, *setVotingDelay* function updates the votingDelay parameter in the “GenuineDStorage” contract. It takes a new value for votingDelay as an argument and assigns it to the votingDelay state variable. After updating, the function returns true to indicate a successful update. *getVotingDelay* function is a public view function. It is designed to return the value of the votingDelay variable, which represents the delay period used in the GenuineDAO's voting mechanism.

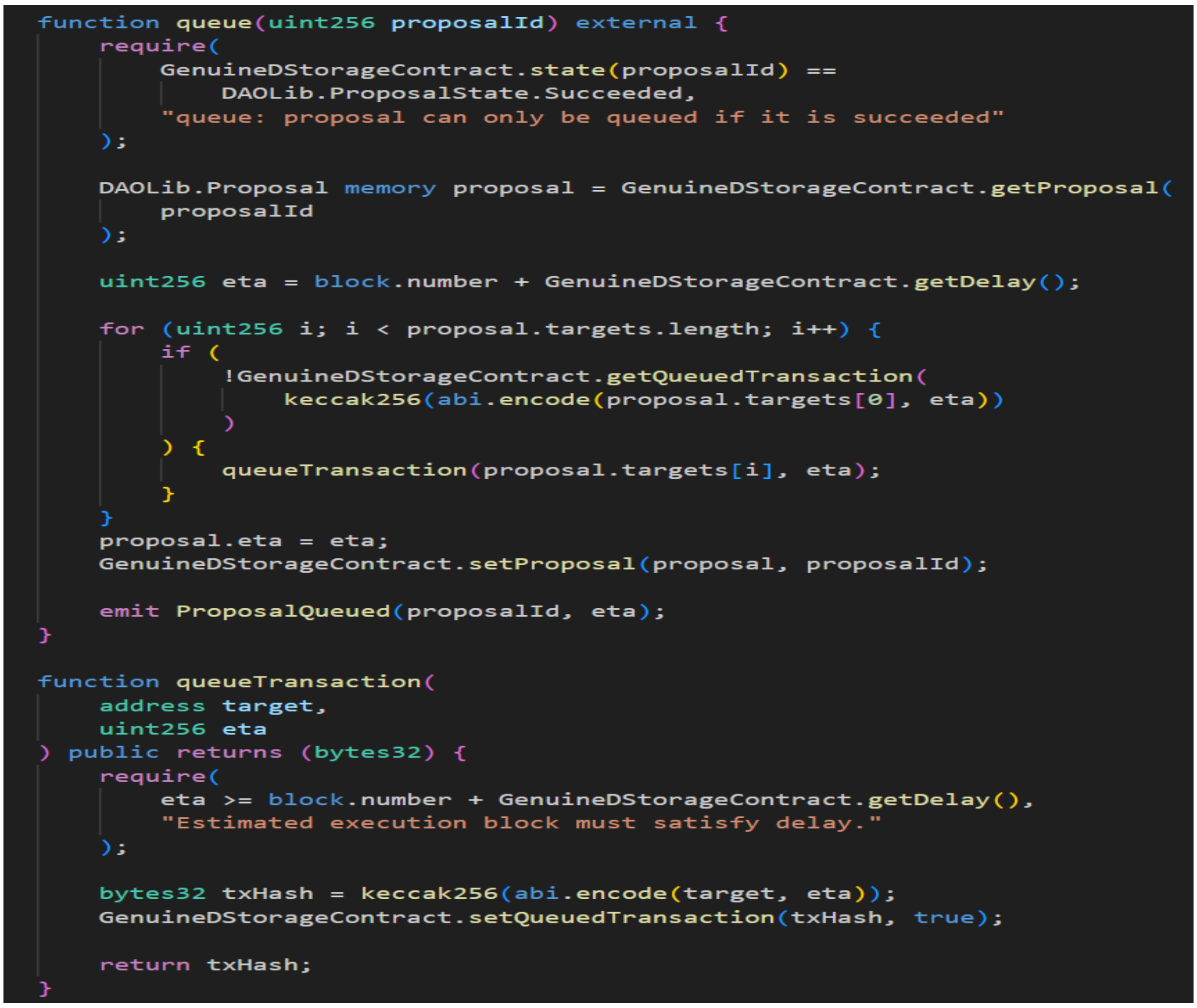
Example 2, *setVotingPeriod* function updates the votingPeriod parameter in the “GenuineDStorage” contract. It takes a new value for votingPeriod as an argument and assigns it to the votingPeriod state variable. After updating, the function returns true to show a successful update. getVotingPeriod function is a public view function. It is designed to return the value of the votingPeriod variable.

As part of voting process, the system checks the number of votes per proposal during the voting time. If a proposal has maximum votes to pass the proposal (*forVotes > againstVotes*), the state of proposal will be changed to “succeeded” otherwise it shows “defeated”. The succeeded proposal is ready to move into the queue step.

5.4.6 The Queue and Timelock

After a proposal has succeeded, it is moved into the queue and timelock step, where there is delay running a certain functionality of a smart contract until a specific amount of time has passed. The proposal state will be changed to queue.

Functions in figure 24 are written for Queuing step and Timelock. Function *Queue* checks the proposal ID and the state if proposal to ensure it is succeeded*.* Function *queueTransaction* is responsible for queuing a proposal for future execution at a specified time (eta).
getDelay() from GenuineDStorageContract, estimated time of arrival (eta) and require statements.
ProposalQueued event is emitted and indicate that a proposal has been successfully queued for the execution.



```
function queue(uint256 proposalId) external {
    require(
        GenuineDStorageContract.state(proposalId) ==
            DAOLib.ProposalState.Succeeded,
        "queue: proposal can only be queued if it is succeeded"
    );

    DAOLib.Proposal memory proposal = GenuineDStorageContract.getProposal(
        proposalId
    );

    uint256 eta = block.number + GenuineDStorageContract.getDelay();

    for (uint256 i; i < proposal.targets.length; i++) {
        if (
            !GenuineDStorageContract.getQueuedTransaction(
                keccak256(abi.encode(proposal.targets[0], eta))
            )
        ) {
            queueTransaction(proposal.targets[i], eta);
        }
    }
    proposal.eta = eta;
    GenuineDStorageContract.setProposal(proposal, proposalId);

    emit ProposalQueued(proposalId, eta);
}

function queueTransaction(
    address target,
    uint256 eta
) public returns (bytes32) {
    require(
        eta >= block.number + GenuineDStorageContract.getDelay(),
        "Estimated execution block must satisfy delay."
    );

    bytes32 txHash = keccak256(abi.encode(target, eta));
    GenuineDStorageContract.setQueuedTransaction(txHash, true);

    return txHash;
}
```

Figure 24 - GenuineD smart contract – Queue and Timelock.

Queue function is marked as “external” which can be called from outside the contract. It checks the state of the proposal with the given ID, if the proposal has been succeeded. As part of the GenuineDAO system, a proposal can be queued if it is succeeded. Then retrieves the details of a proposal and calculates the proposal’s eta. It followed by “require” statement that checks if the condition of proposed proposalID is met otherwise display the error message “proposal can only be queued if it is succeeded”. Eta introduces a time delay before the proposal can be executed and can be calculated based on the current block number (block.number) to the delay value obtained from GenuineDStorageContract.getDelay().

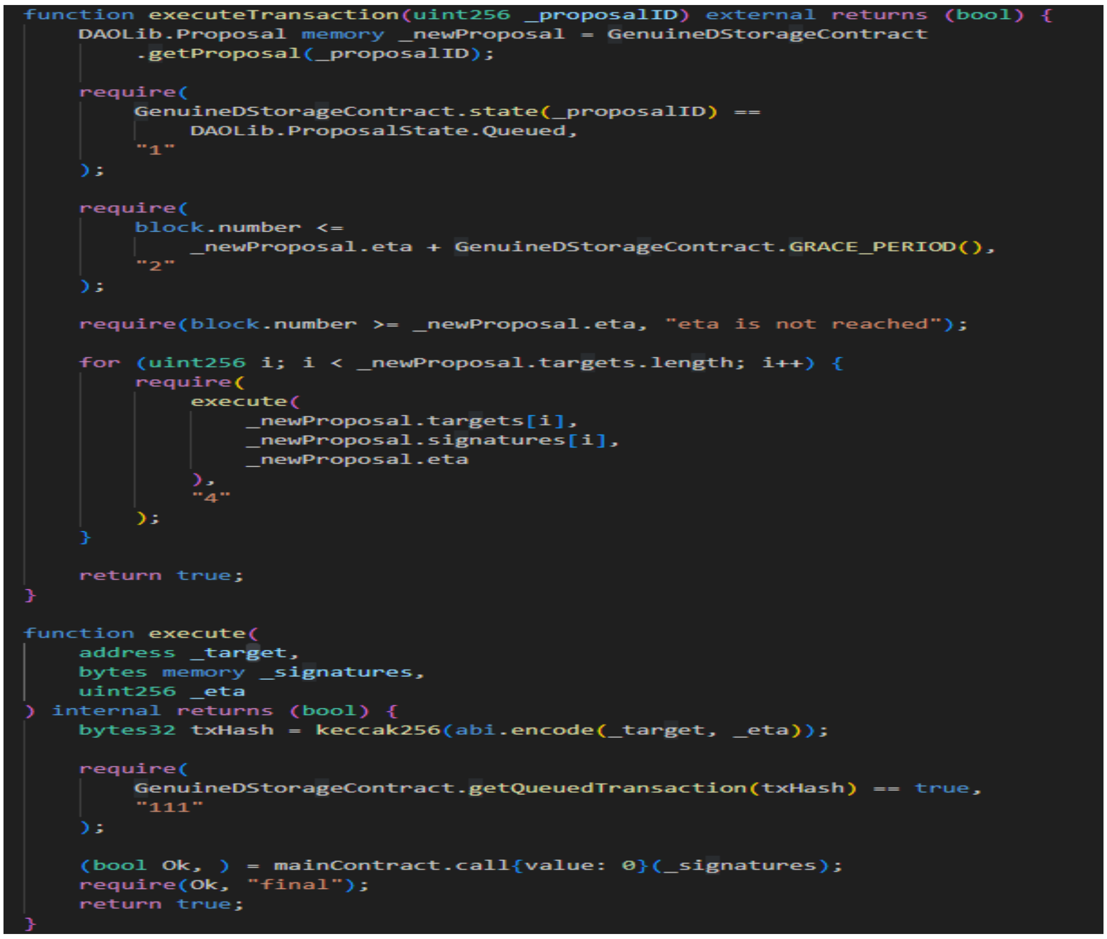
Within the loop, each address from proposal.targets is used to generate a unique identifier (hash) for the proposed transaction using the keccak256 hash function. The identifier includes both the target address and the eta value calculated earlier. This loop also calls “DStorageContract.getQueuedTransaction” to check whether the proposed transaction with the given identifier or targets has already been queued or not. Then If the transaction has not been queued, the function queueTransaction is called to queue the transaction.

The “*queueTransaction*” function is a public function that anyone can call it and take two parameters *target* (the address of account to execute the transaction) and *eta*. The “require” statement checks if the estimated execution block number (eta) is equal to the current block number plus the delay value obtained from GenuineDStorageContract.getDelay(). “Genuine DAO” enforces the time delay to ensure that transaction cannot be executed before the specified ETA. Otherwise it displays an error message “Estimated execution block must satisfy delay”.

The KWccak256 hash function has been used to create the hash of transaction (combination of target address and ETA) and update the GenuineDStorage contract.
As explained the time-based mechanism has been implemented in different steps including queuing. Both *queue*() and *queueTransaction()* functions have used ‘getDelay’.
It is notable that only successful proposals can be queued. When a transaction is in queue stage, ETA is zero.

5.4.7 Executing a Proposal

During the execution step, two functions *executeTransaction* and *execute* have been designed to execute transactions associated with a queued proposal if certain conditions have been met. The code is shown in figure 25. The "executeTransaction" function is callable externally and takes a _proposalId as an argument. It retrieves the details of proposal by using GenuineDStorageContract.getProposal(_proposalId). Then it checks the state of queued proposal and if the current block number is within the *grace period* for execution (200 blocks), *eta*, *targets* and *signatures*. After that, it calls the "*execute*" function to execute the transactions. The "execute" function can only be called from within the smart contract. It takes the target address, signatures, and eta as parameters and generates a transaction hash based on the target and eta. It again checks if the transaction is queued by calling GenuineDStorageContract.getQueuedTransaction(txHash). If the transaction is queued, it attempts to execute the transaction by invoking the GenuineD contract using mainContract.call which reviews the success of the transaction execution and returns true if it is successful.



```
function executeTransaction(uint256 _proposalID) external returns (bool) {
    DAOLib.Proposal memory _newProposal = GenuineDStorageContract
        .getProposal(_proposalID);

    require(
        GenuineDStorageContract.state(_proposalID) ==
            DAOLib.ProposalState.Queued,
        "1"
    );

    require(
        block.number <=
            _newProposal.eta + GenuineDStorageContract.GRACE_PERIOD(),
        "2"
    );

    require(block.number >= _newProposal.eta, "eta is not reached");

    for (uint256 i; i < _newProposal.targets.length; i++) {
        require(
            execute(
                _newProposal.targets[i],
                _newProposal.signatures[i],
                _newProposal.eta
            ),
            "4"
        );
    }

    return true;
}

function execute(
    address _target,
    bytes memory _signatures,
    uint256 _eta
) internal returns (bool) {
    bytes32 txHash = keccak256(abi.encode(_target, _eta));

    require(
        GenuineDStorageContract.getQueuedTransaction(txHash) == true,
        "111"
    );

    (bool Ok, ) = mainContract.call{value: 0}(_signatures);
    require(Ok, "final");
    return true;
}
```

Figure 25 - GenuineD smart contract - Execution Functions.

5.4.8 The Front-end Genuine DAO Implementation

This section focuses on the comprehensive development process and implementation of the front-end of Genuine DAO application. It delves into the intricacies of using Visual Studio Code and JavaScript to develop a decentralised application. Creating the front-end of a decentralised application with Visual Studio Code and JavaScript allows the researcher to create a user-friendly, and dynamic interface that interact with Ethereum Blockchain and host Genuine DAO application and all data/transactions on decentralised storage system securely. It also gets benefit of Blockchain network to enhance transparency, user autonomy and efficiency. The following explains the process of implementing front-end of "Genuine DAO". The following sections explain the main files and folders that are used to implement the front-end of Genuine DAO application.

- **Node_modules**

It is a directory that contains all the dependencies such as installed required libraries. It is commonly linked with projects that use JavaScript-based package managers, such as npm or Yarn.

- **Public**

It is a directory in a React project that store static files such as images and fonts that do not change during the application's runtime.

- **Src**

In a React project, this directory contains the front-end source code that use to be modified during the coding process. It is where all of JavaScript code exist to control behaviour of decentralised application. This folder includes the entry point for Genuine DAO application, such as JavaScript files, index.js, CSS files, ABI and other essential information to develop front-end application.

- **Package.json**

It is a file that is used with Node.js and npm to manage the project's dependencies, scripts, project's name and version and other relevant information.

5.4.8.1 Front-end Source Code in “src” Folder

As part of all the folders explained above, the focus lies on the “src” folder, mainly because it stores front-end source code which includes the following parts.

- **Assets**

This folder used to organise static files such as images, fonts, audio files that are used in this project (Kumar, 2021).

- **Components**

This folder contains reusable atomic and molecular components that encapsulate specific functionality and split the user interface into independent. Each component folder represents a specific part of the user interface which make it more organised to manage and maintain code (Kumar, 2021; React, 2023).

- **Layouts**

This folder used to store any layout components which can be used to define the structure of each page and organise the layout-related components and files (Onyeije, 2023).

- **Views**

Components in this folder are responsible for fetching data, managing state, and rendering the layout of the page. They can also be used to define the overall structure of a page, such as the header, main content, and footer sections.

- **App.tsx**

It is the main react component of the application. It represents the top-level component of application (Pagan, 2021).

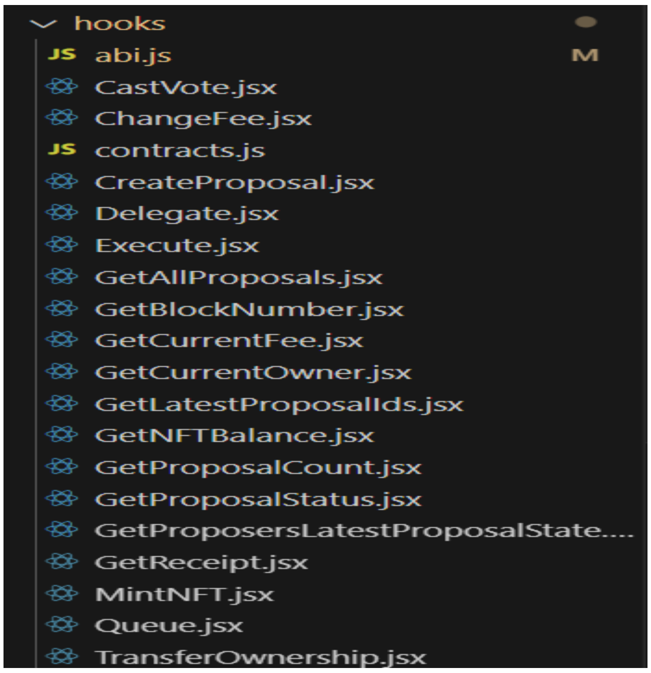
- **Index.tsx**

In a React project, it is the entry point file. It is the first file that will be executed when the application starts. Index.tsx and is responsible for rendering the App component (the root component) of application (Pagan, 2021).

- **Hooks**

in a React project, hooks are JavaScript functions to build user interface to connect to back-end (smart contracts). It provides functionality to work with stateful logic in functional components (React, 2023). The following shows the Hooks folder structure.

5.4.8.2 Front-end Source Code in “Hooks” Folder

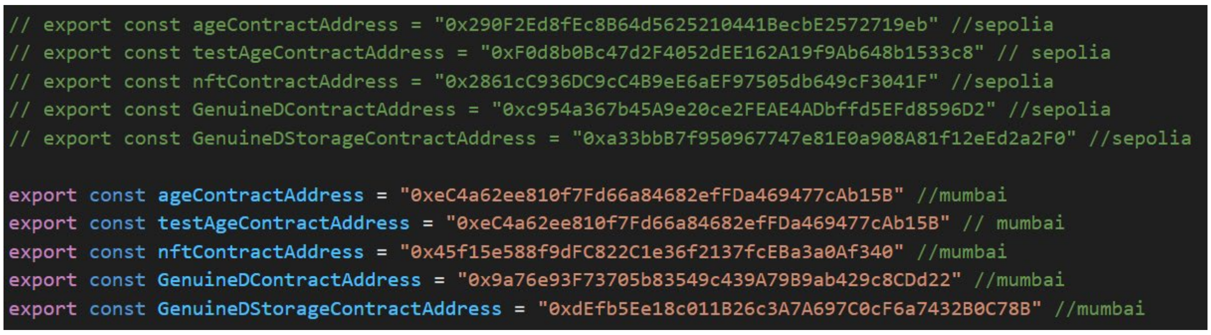


```
hooks
  abi.js                      M
  CastVote.jsx
  ChangeFee.jsx
  contracts.js
  CreateProposal.jsx
  Delegate.jsx
  Execute.jsx
  GetAllProposals.jsx
  GetBlockNumber.jsx
  GetCurrentFee.jsx
  GetCurrentOwner.jsx
  GetLatestProposalIds.jsx
  GetNFTBalance.jsx
  GetProposalCount.jsx
  GetProposalStatus.jsx
  GetProposersLatestProposalState....
  GetReceipt.jsx
  MintNFT.jsx
  Queue.jsx
  TransferOwnership.jsx
```

Figure 26 - Front-end - Hook Folder

Abi.js

ABI stands for Application Binary Interface. Abi.js is a JavaScript file and contains the contract functions, their inputs and outputs, and contract’s addresses that are necessary for interacting decentralised application with smart contracts. The following figure displays important contract’s addresses that used in this project with both Ethereum mainnet and Polygon networks.



```
// export const ageContractAddress = "0x290F2Ed8fEc8B64d5625210441BecbE2572719eb" //sepolia
// export const testAgeContractAddress = "0xF0d8b0Bc47d2F4052dEE162A19f9Ab648b1533c8" // sepolia
// export const nftContractAddress = "0x2861cC936DC9cC4B9eE6aEF97505db649cF3041F" //sepolia
// export const GenuineDContractAddress = "0xc954a367b45A9e20ce2FEAE4ADbffd5EFd8596D2" //sepolia
// export const GenuineDStorageContractAddress = "0xa33bbB7f950967747e81E0a908A81f12eEd2a2F0" //sepolia

export const ageContractAddress = "0xeC4a62ee810f7Fd66a84682efFDa469477cAb15B" //mumbai
export const testAgeContractAddress = "0xeC4a62ee810f7Fd66a84682efFDa469477cAb15B" // mumbai
export const nftContractAddress = "0x45f15e588f9dFC822C1e36f2137fcEBa3a0Af340" //mumbai
export const GenuineDContractAddress = "0x9a76e93F73705b83549c439A79B9ab429c8CDd22" //mumbai
export const GenuineDStorageContractAddress = "0xdEfb5Ee18c011B26c3A7A697C0cF6a7432B0C78B" //mumbai
```

Figure 27 - Contract’s Addresses for Ethereum Mainnet and Polygon.

CastVote.jsx

Contains the function that handles casting a vote for a specific proposal. As part of decentralised governance process, voters need to have tokens to be able to participant in voting process. Having tokens ensures that participants have a stake in the organisation's success, encourages them to participant actively, enhances security of decision making and prevents attacks such as Sybil attacks (creating fake identities).

ChangeFee.jsx

Includes function that change the transaction rate fee. If the owner is developer on anyone who deployed the smart contracts on Blockchain but not the GenuineD contract. Therefore, the owner/developer would be able to change the transaction rate fee.

Contracts.js

Contains combination of smart contracts that will be used by hooks functions.

CreateProposal.jsx

Contains function that makes changes in the smart contract such as proposal fee which can be changed by voting.

Delegate.jsx

Includes a function that delegate vote of the user. Token holders can delegate their voting powers to another address to participate in voting process for specific proposals. The voting powers can be tracked via checkpoints which keeps the history of each account.

Execute.jsx file

Contains functions for executing proposals. This function can be called by any Ethereum address.

GetAllProposals.jsx

Provides a view of all created proposals on the website.

GetBlockNumber.jsx

Is a function that fetches the current block number.

GetCurrentFee.jsx

Includes function that returns the transaction rate fee that can be changed by DAO proposals.

GetCurrentOwner.jsx

Returns the owner of the smart contract.

GetLatestProposalIds.jsx

Returns the created proposals of a specific voter based on the address.

GetNFTBalance.jsx

Returns the token value of a specific voter based on the address.

GetProposalCount.jsx

Includes a function that returns total number of proposals.

GetProposalStatus.jsx

Returns the status of given proposal.

GetProposersLatestProposalState.jsx

Returns the state of the latest proposal.

GetReceipt.jsx

Returns information of each proposal.

MintNFT.jsx

Is responsible for minting NFT token for participants (voters).

Queue.jsx

Is responsible for queuing proposals for specific period of time.

TransferOwnership.jsx

Includes a function that responsible to change the ownership of smart contract.

5.5 Testing the Genuine DAO Application

After writing front-end code, the next step includes interacting with the back-end smart contracts that deployed on Blockchain by using Web3 library. Smart contracts are deployed on both Ethereum network (Sepolia testnet) and Polygon network (Mumbai and Amoy testnets) using MetaMask wallet. Blockchain testnets offer a safe environment to test the

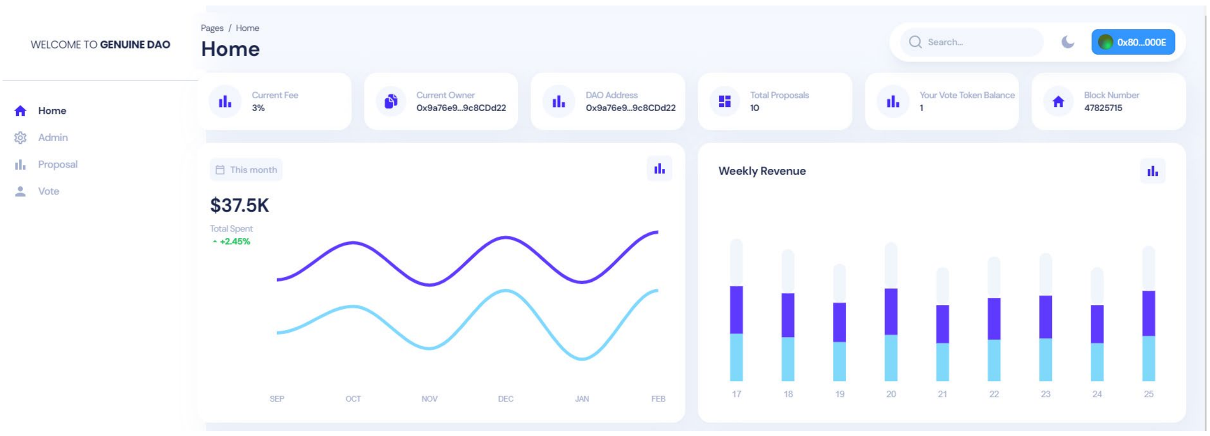
"Genuine DAO" DApp and ensure that the integration between the front-end and back-end functions correctly.

5.5.1 Testing Requirement 1 (Distributed Decision-Making) and Requirement 2 (Elimination of Single Points of Failure)

The following steps explain how the "Genuine DAO" DApp transfers the ownership to a new contract, and how the GenuineD contract takes the control from developer/owner and change the transaction fee via the voting process.

The Genuine DAO DApp boots up by using the Command "npm start" in Command Prompt. Then the development server is launched, and the home page of Genuine DAO DApp appears. This decentralised application includes four pages.

1. Home page that shows current fees, the address of current owner, the address of DAO owner, total proposal numbers, token numbers for a specific participant, block number which will get it from Ethereum mainnet and Polygon testnets.



Current Fee
3%

Current Owner
0x9a76e9...9c8CDd22

DAO Address
0x9a76e9...9c8CDd22

Total Proposals
10

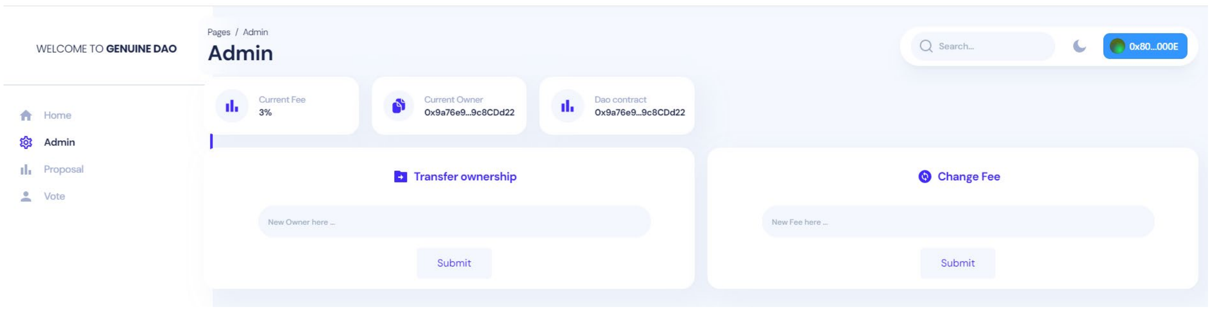
Your Vote Token Balance
1

Block Number
47825715

$37.5K
Total Spent
+2.45%

Figure 28 - Genuine DAO - Home Page.

2. Admin page to change the transaction rate fee and transfer ownership of smart contracts.

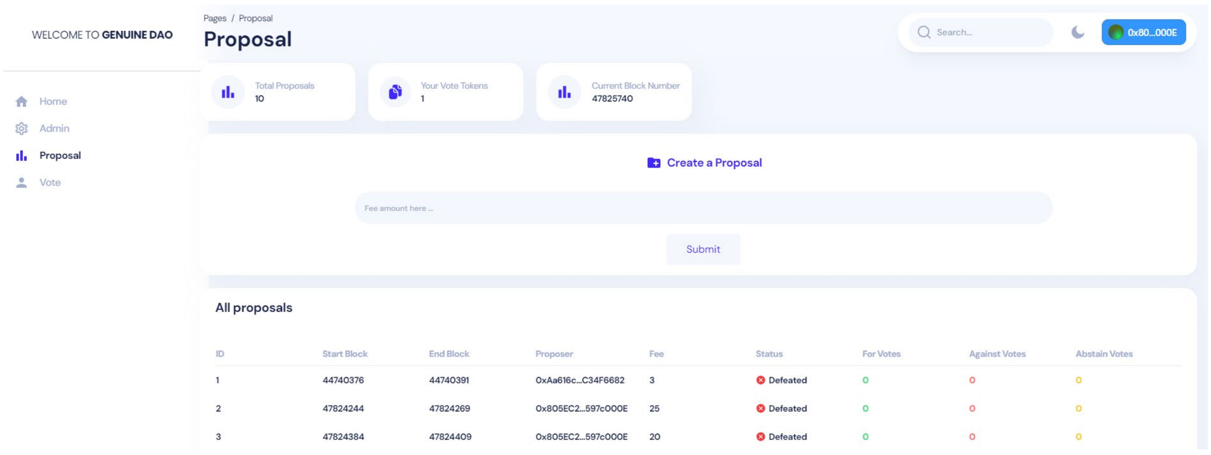


Admin

Current Fee	3%
Current Owner	0x9a76e9...9c8CDd22
Dao contract	0x9a76e9...9c8CDd22

Figure 29 - Genuine DAO - Admin Page.

3. Proposal page is where users can create a proposal with suggested transaction rate fee and submit it on Blockchain network.



Proposal

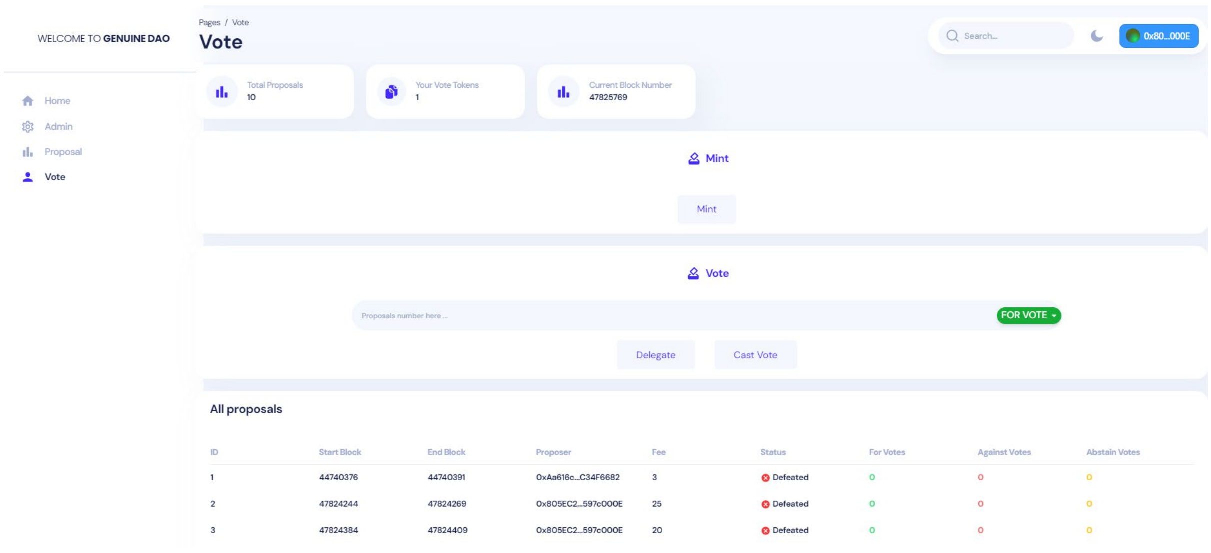
Total Proposals	10
Your Vote Tokens	1
Current Block Number	47825740

All proposals

ID	Start Block	End Block	Proposer	Fee	Status	For Votes	Against Votes	Abstain Votes
1	44740376	44740391	0xAa616c...C34F6682	3	Defeated	0	0	0
2	47824244	47824269	0x805EC2...597c000E	25	Defeated	0	0	0
3	47824384	47824409	0x805EC2...597c000E	20	Defeated	0	0	0

Figure 30 - Genuine DAO - Proposal Page.

4. Vote page is where voters would be able to mint a token to be able to delegate and vote for a proposal.



Vote

Total Proposals	10
Your Vote Tokens	1
Current Block Number	47825769

All proposals

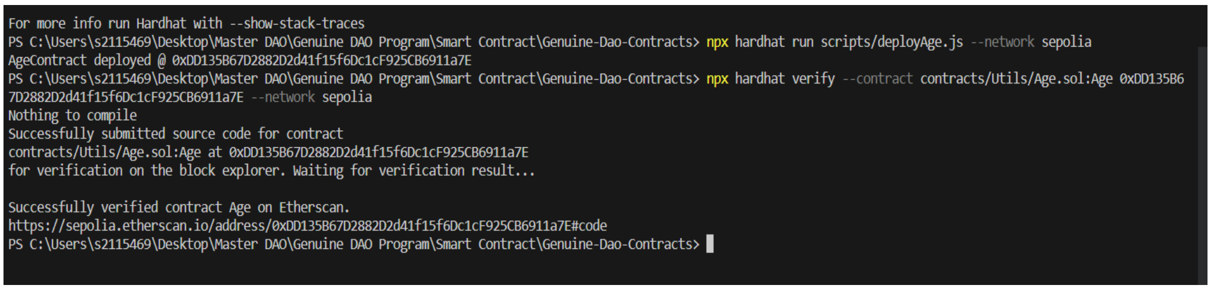
ID	Start Block	End Block	Proposer	Fee	Status	For Votes	Against Votes	Abstain Votes
1	44740376	44740391	0xAa616c...C34F6682	3	Defeated	0	0	0
2	47824244	47824269	0x805EC2...597c000E	25	Defeated	0	0	0
3	47824384	47824409	0x805EC2...597c000E	20	Defeated	0	0	0

Figure 31 - Genuine DAO - Vote Page.

The first step is to deploy the smart contract on Ethereum testnet, connect MetaMask wallet and interact between front-end and back-end. The most important part is the person who deployed the smart contract(s) and interacts with front-end. Usually, a developer who deploys smart contract(s) are referred to as the owner. The decentralised application operates and governs by the predefined rules and logic on the smart contract's code. Once the smart contract is deployed on the Blockchain, it will become immutable, and the developer cannot change/modify the functions and logic directly. However, it can be possible to update data state over time or the developer/owner can implement certain mechanisms or have a specific access to the functions to affect the transaction process indirectly and may cause centralisation risks. Here, we demonstrate with our Genuine DAO, how we take the control from developer/owner and transfer it to Genuine DAO smart contract with the aim to avoid one owner control.

In deploying and verifying smart contracts on different testnets the first step is to get an API key from Etherscan and Polygonscan then add both to Hardhat config. The next step is to add the Sepolia testnet and Polygon Mumbai and Amoy testnets into Hardhat config. Use JSON_RPC URL Alchemy to connect to the networks. The following step is to get some Sepolia Faucet/Mumbai and Amoy Faucets from Alchemy.

- Using Hardhat deploy plugin to deploy smart contract (deploy.Age) by using the following command in terminal: "npx hardhat run scripts/deployAge.js –network Sepolia / Mumbai". Deployed contract address: 0xDD135B67D2882D2d41f15f6Dc1cF925CB6911a7E
- Verify the deployed smart contract by using the contract address on testnet and type the following command on terminal: "npx hardhat verify –contract contracts/Utils/Age.sol:Age 0xDD135B67D2882D2d41f15f6Dc1cF925CB6911a7E"



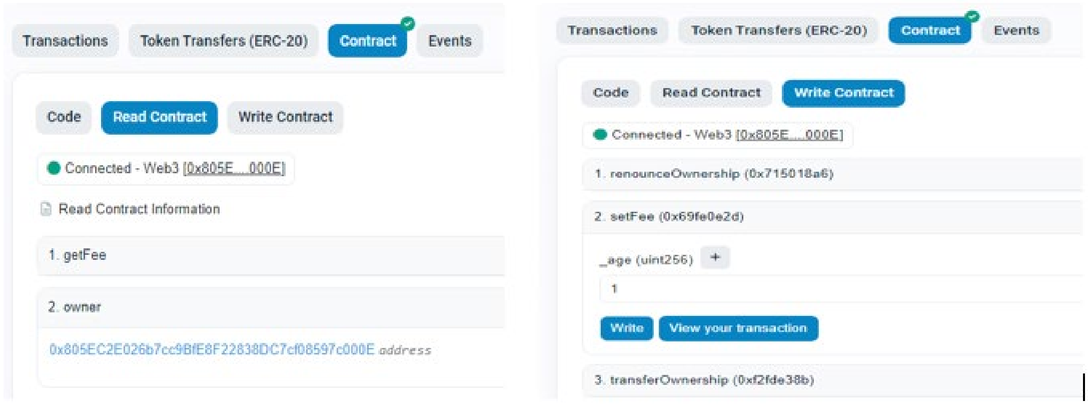
```
For more info run Hardhat with --show-stack-traces
PS C:\Users\s2115469\Desktop\Master DAO\Genuine DAO Program\Smart Contract\Genuine-Dao-Contracts> npx hardhat run scripts/deployAge.js --network sepolia
AgeContract deployed @ 0xDD135B67D2882D2d41f15f6Dc1cF925CB6911a7E
PS C:\Users\s2115469\Desktop\Master DAO\Genuine DAO Program\Smart Contract\Genuine-Dao-Contracts> npx hardhat verify --contract contracts/Utils/Age.sol:Age 0xDD135B6
7D2882D2d41f15f6Dc1cF925CB6911a7E --network sepolia
Nothing to compile
Successfully submitted source code for contract
contracts/Utils/Age.sol:Age at 0xDD135B67D2882D2d41f15f6Dc1cF925CB6911a7E
for verification on the block explorer. Waiting for verification result...

Successfully verified contract Age on Etherscan.
https://sepolia.etherscan.io/address/0xDD135B67D2882D2d41f15f6Dc1cF925CB6911a7E#code
PS C:\Users\s2115469\Desktop\Master DAO\Genuine DAO Program\Smart Contract\Genuine-Dao-Contracts> █
```

Figure 32 - Deploy and Verify Age Contract

- Check the verification of the deployed contract on Sepolia Etherscan website or Polygonscan, paste the contract address on the search bar and check if the contract is verified. By clicking on “read contract” There are two functions (*getFee* and *owner*). In “write contract” section, *renounceOwnership, setFee* and *transferOwnership* are available. Currently, the developer is the owner because developer deployed the contract on Ethereum network for the first time.

After connecting the contract to Web3 through MetaMask wallet, the owner of contract would be able to control functions such as “set the transaction fee” or “transfer the ownership to any addresses”. This gives the owner control over critical functions, which poses a centralisation threat leading to security issues.

Figure 33 displays the verified Age contract on Etherscan with available functions.



Connected - Web3 [0x805E...000E]

Read Contract Information

1. getFee

2. owner

0x805EC2E026b7cc9BfE8F22838DC7cf08597c000E *address*

Connected - Web3 [0x805E...000E]

1. renounceOwnership (0x715018a6)

2. setFee (0x69fe0e2d)

_age (uint256)

1

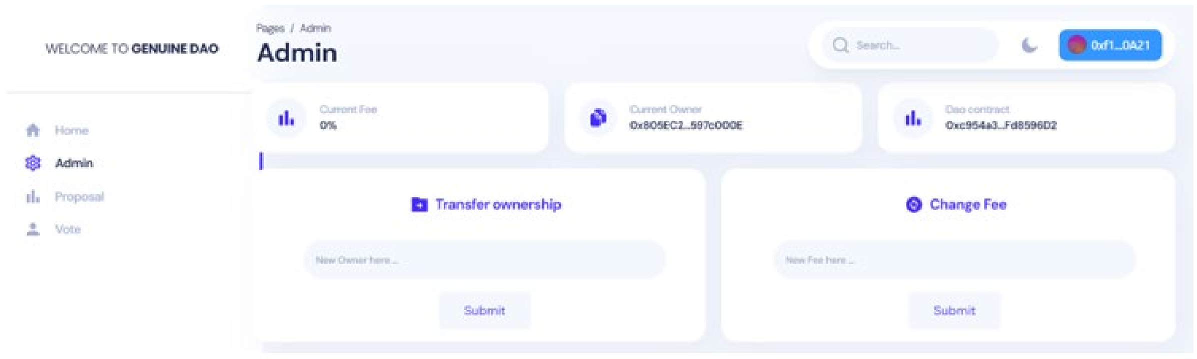
3. transferOwnership (0xf2fde38b)

Figure 33 Deployed contract with available functions on Etherscan.

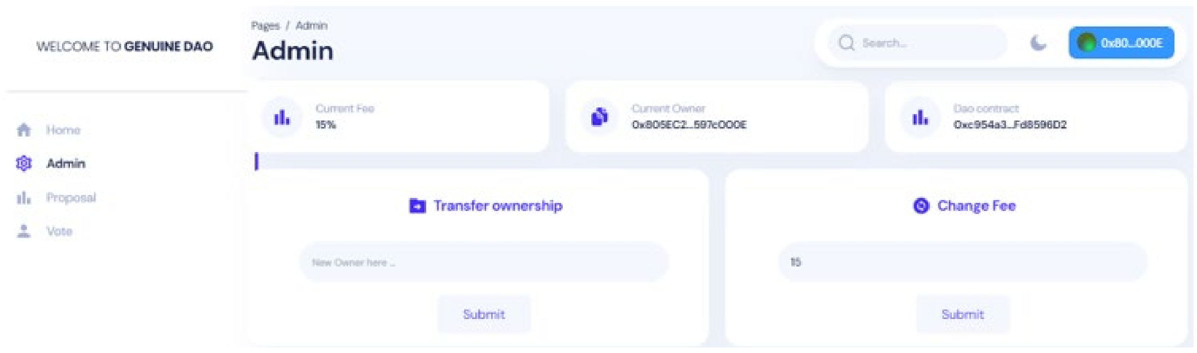
As explained above, Abi.js includes contract’s addresses. There are few steps to follow in order to replace the address of deployed contract (contract creator) and view the address of the new owner (Genuine DAO).

1- Open abi.js on Visual Studio Code, paste the address of deployed contract (deployAge) in front of “ageContractAddress”.

The current owner is the one who deployed the smart contract for the first time (typically the developer). The current owner has permission to change the transaction rate fee and transferring ownership to any addresses which cause major security risks. Figure 34 shows that the address of current owner is different from the address of the Genuine DAO contract. The current address is the address of the developer who deployed the contract with having control over critical functions. The current owner modified the transaction rate fee from %0 to %15.



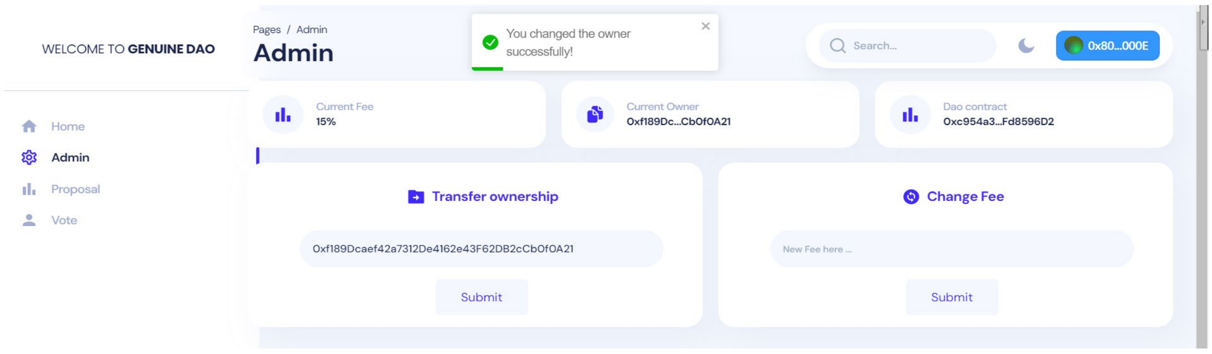
0xf1...0A21
Current Fee 0%
Current Owner 0x805EC2...597c000E
Dao contract 0xc954a3...Fd8596D2



0x80...000E
Current Fee 15%
Current Owner 0x805EC2...597c000E
Dao contract 0xc954a3...Fd8596D2
15

Figure 34 - Deployed Contract with One Owner Control to Change Transaction Rate Fee.

Figure 35 shows that the current owner has permission to transfer ownership with any addresses with centralised control.

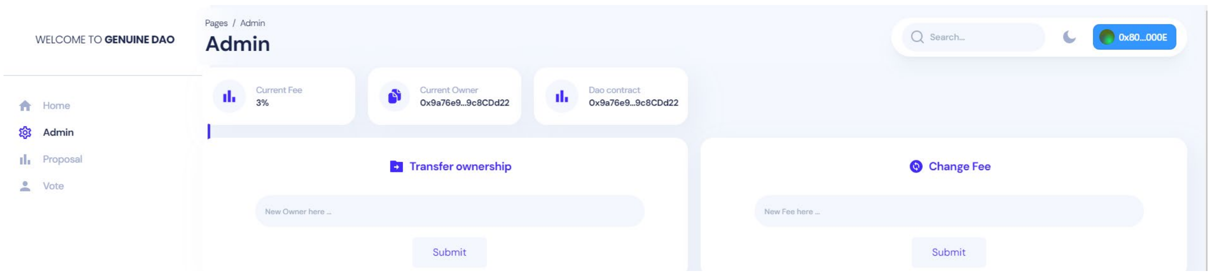


You changed the owner successfully!
0x80...000E
Current Fee 15%
Current Owner 0xf189Dc...Cb0f0A21
Dao contract 0xc954a3...Fd8596D2
0xf189Dcaef42a7312De4162e43F62DB2cCb0f0A21

Figure 35 - Transfer Ownership by Owner/Developer to new Address with Centralised Control.

2- To prevent the centralisation risk associated with smart contracts being controlled only by the owner/developer. The contract address of GenuineDContractAddress is available in abi.js. it should be copied and pasted in Admin page in front of transfer ownership. Then the ownership will be transferred to GenuineD contract. This helps to minimise the control of the system from one owner/developer and use a DAO structure (democratic governance model) and voting process for changing transaction rate fee and making decisions in a decentralised form.

Figures 36 and 37 show that the ownership of the smart contract has been transferred to GenuineD contract with specific contract addresses (GenuineDContractAddress) that are highlighted in figure 27. Figures 36 (Sepolia network) and 37 (Polygon Network) signify the transfer of control and authority from the one owner to a new owner which is a smart contract that is developed with DAO rules. It is evident that the address of the current owner and the GenuineD contract are the same, indicating that both belong to Genuine DAO whereas in figure 34, addresses are different.

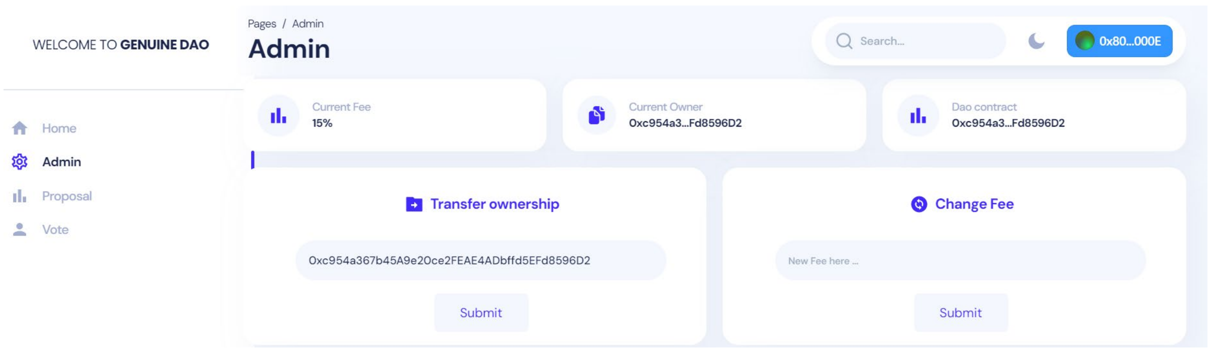


Current Fee
3%

Current Owner
0x9a76e9...9c8CDd22

Dao contract
0x9a76e9...9c8CDd22

Figure 36 - Genuine DAO Ownership to Control the Decentralised Application (Polygon Network).



Current Fee
15%

Current Owner
0xc954a3...Fd8596D2

Dao contract
0xc954a3...Fd8596D2

0xc954a367b45A9e20ce2FEAE4ADbffd5EFd8596D2

Figure 37 - Genuine DAO Ownership to Control the Decentralised Application (Sepolia Network).

This transaction highlights Genuine DAO's role as the owner of the smart contract to mitigate against one owner control and centralisation risks. The Genuine DAO distributes decision-making power among nodes, to ensure a more decentralised and resilient system. By implementing the appropriate logic and access controls within the GenuineD contract, a transparent application is developed that maintains the decentralised nature of the application, allowing it to operate autonomously and independently with reducing control of one owner/developer. This preventive method minimises the centralisation risks and promotes the security of the system. This speaks directly to Requirement 1 (Distributed Decision-Making) and Requirement 2 (Elimination of Single Points of Failure).

The Genuine DAO application with decentralised control is ready for participants to create their proposals and suggest transaction rate fees. Submitted proposals will continue through the voting process. Voters should submit their vote for each proposal. The transaction rate fee from a successful proposal with enough votes would be recorded on the blockchain network. Participants/users would then be able to make secure transactions on a decentralised distributed ledger. This provides transparency and increases users' trust in a system that allows them to contribute to decision-making based on DAO rules, rather than depending on the chosen decisions of particular individuals or groups.

5.5.2 Testing of Requirement 3: Secure Smart Contract Execution

This research focuses on preventing frontrunning attacks to enhance the security of smart contract execution. There are several testing frameworks such as Hardhat, Foundry, Brownie and Truffle for smart contract development and auditing. For the purpose of this research, the Hardhat has been used which is not only a development framework but also includes a built-in testing framework.

Hardhat offers the opportunity to write tests for smart contracts using JavaScript testing libraries, which completely suit the purpose of this research. It is a development framework for Ethereum network. Hardhat offers wide range of tools, plugins, and features to build decentralised applications on Ethereum network. developers can use Hardhat to development process and testing (Hardhat, 2023). The researcher installed Hardhat using

Node.js package manager (npm) and use commands “npm install” and “npx hardhat” to test front running process on the Genuine DAO. It automatically checks codes for bugs and mistakes, runs a test of frontrunning attack to detect and prevent frontrunners from attempting to prioritise their transactions over other users’ transactions and gaining advantage by doing so.

All tools and libraries that were used to test the frontrunning process have been outlined in section 5.3. The frontrunning test code is located in the Genuine DAO, *Contract/TestFrontRunning.js path*. Libraries such as Hardhat and Chai are imported to assist with the testing process. Hardhat contains Ethers that are used to work with contracts. The e*xpect* function is imported from Chai library to define assertions in the test case. Variables are defined to store users, smart contracts and blockchain information such as transactions, block number, proposal state. The time-based access control is implemented. The *beforeEach()* function is used to get addresses from ethers, deploy and initialise smart contracts including main contract, AgeContract and other contracts NFTContract, GenuineDAOContract and GenuineDStorageContract.

- Deploy AgeContract to change the transaction rate fee by the owner.
- Deploying NFTContract to create and mint tokens for users to contribute in voting process.
- Deploying GenuineDAOContract to create proposals and voting process to change the transaction rate fee.
- Deploying GenuineDStorageContract that plays the role of data storage for the main GenuineDAOContract.

Once the ownership from AgeContract to GenuineDAOContract is enforced, as shown in figure 42, it is not possible for the contract creator/owner to control critical functions or change the transaction rate fee. GenuineDAOContract takes control of the system and the transaction rate fee can only be decided through a voting process.

Samples of code used for testing is shown in the next chapter whereby an evaluation of the effectiveness of this implementation is discussed in relation to the objectives of this research.

5.6 Summary

As underlined in Chapter 3, smart contracts with centralised ownership pose major security issues and act as a single point of failure, which contradicts the very decentralised nature of Blockchain. To mitigate against the risks associated with centralised control, a decentralised application with an enhanced DAO structure is proposed and developed. This application enforces automated rules that are encoded in smart contracts and enforces a decentralised decision-making process. The power of decision-making will be distributed and therefore preventing smart contract developers from manipulating the network through one owner control. Furthermore, the developed decentralised application, "Genuine DAO ", added an additional security level to prevent one-owner control by developing a contract called GenuineD as an owner to control critical functions. Genuine DAO is written in a way that not only minimises the risks of single point of failure and one owner control, but also prevents Frontrunning attacks, which will be discussed in the next chapter alongside the evaluation of the developed decentralised application.

Chapter 6: Discussion and Evaluation

This chapter delves into the effectiveness of the chosen methodology, analyses the developed decentralised application and assesses the overall contribution of this study.

6.1 Overall Evaluation Approach

A comprehensive evaluation of the developed "Genuine DAO" application is carried out to assess the effectiveness of the application against the three key requirements namely:

- Distributed decision-making
- Elimination of single points of failure
- Secure smart contract execution

The evaluation is conducted through a multi-faceted approach:

1. **Peer Review**: The research findings, particularly the conceptual taxonomy of vulnerabilities, attacks, and consequences within a seven-layer blockchain system architecture, underwent a rigorous peer review process. The feedback from experts in Blockchain technology, obtained through journal publication, was instrumental in refining the design and enhancing the overall work.
2. **Smart Contract Graphing Code Flow Analysis**: The Solidity Visual Developer tool was utilised to audit the smart contract code, focusing on the interaction between contracts and identifying potential vulnerabilities. This analysis provided insights into the Genuine DAO application' behaviour, ensuring that access control and communication between contracts are secure and efficient.
3. **Security Analysis**: Various mechanisms were implemented to perform a security analysis of the Genuine DAO application, evaluating its effectiveness in preventing centralisation risks, by enforcing distributed decision-making, and enhancing security by eliminating single points of failure and reducing the risks of frontrunning attacks.
4. **Performance Analysis**: The performance of the Genuine DAO application was evaluated in terms of scalability, throughput, and transaction cost. This involved comparing the application's deployment on both Ethereum and Polygon networks,

highlighting the benefits of using Layer-2 scaling solutions to optimise performance and reduce costs.

5. **Expert Review**: Finally, a qualitative analysis was conducted through feedback from Blockchain security experts. Their insights validated the effectiveness of the proposed security controls and the overall design of the Genuine DAO application in addressing centralisation risks and improving performance.

6.2 Peer Review

As mentioned earlier, one of the contributions of this work is the design of a conceptual taxonomy involving vulnerabilities, attacks and their consequences within a seven-layer blockchain system architecture. A model application for best practice towards a more secure smart contract is designed and preventive tools and techniques are suggested for each vulnerability/attack within the seven layers. The research findings, related to this taxonomy, were reviewed and critically evaluated through a peer review process by several experts in the field of Blockchain technology as part of a journal paper submission. Following a robust and thorough review process the paper was accepted for publication (Mollajafari and Bechkoum, 2023). The peer reviewers' feedback, was used by the researcher to further enhance this work.

6.3 Smart Contract Graphing Code Flow Analysis

Auditing is important in smart contract development for improving the quality of smart contract code and for identifying and addressing potential vulnerabilities before deployment. One of the powerful tools to audit smart contracts is *Solidity Visual Developer*, which integrates into the Visual Studio Code (VSCode) extension. This auditing tool provides great features such as visualising the dependencies and interactions between contracts within a codebase. Graphing code flow assisted the researcher to gain insights into Genuine DAO program behaviour, to identify potential vulnerabilities by using an inheritance graphical representation such as access control issues and how contracts communicate and share data with each other.

For the purpose of auditing the code and interaction between the smart contracts, Solidity Visual Developer is installed in the Extension view in VS Code. The graphical code flow includes cycles and lines which represent functions and the interaction between functions respectively. Figure 38 shows the functionality and the interaction within the GenuineD contract. The Internal calls are shown with a green line, External calls with a white line

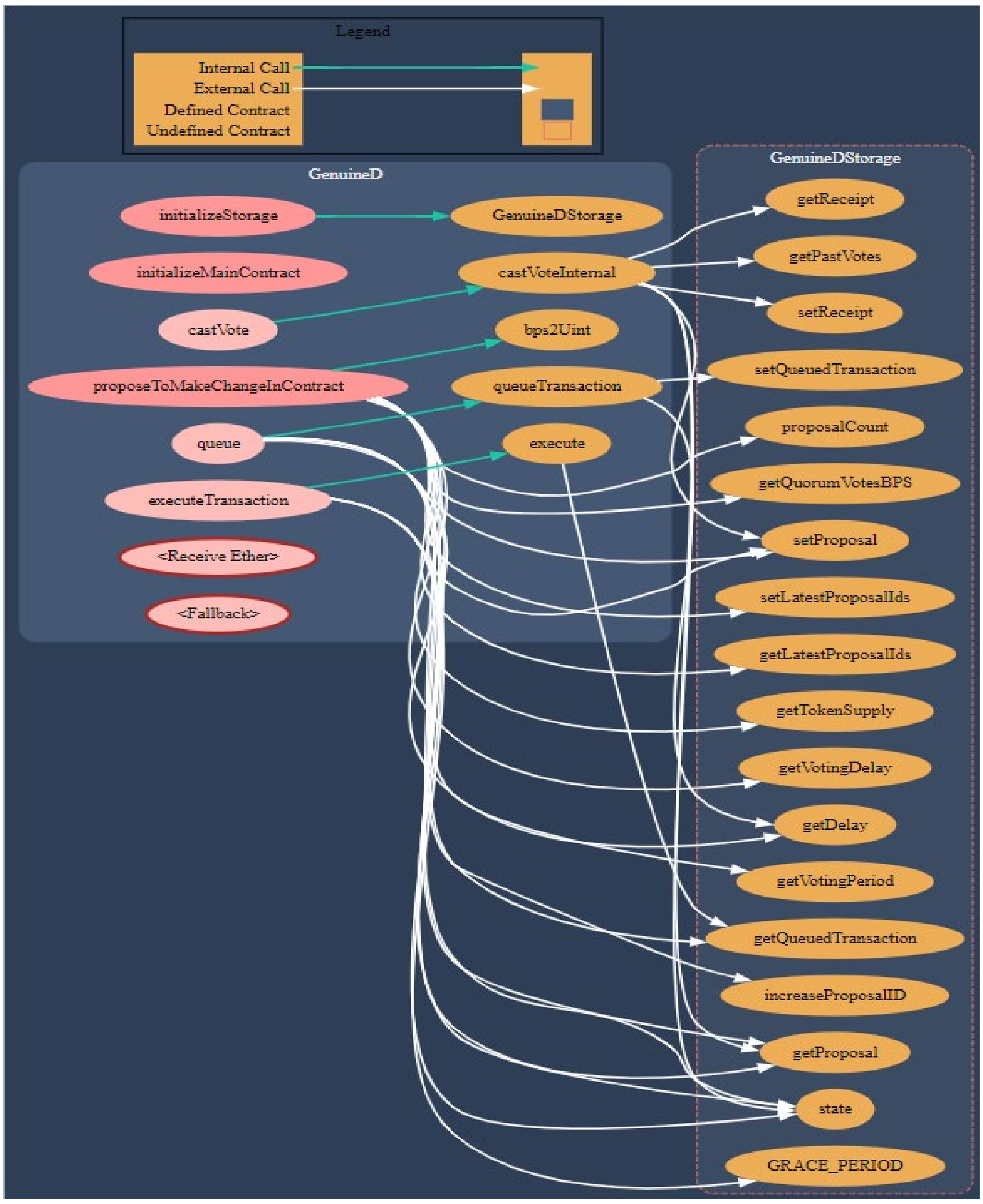


Figure 38 - The Graphical Code Flow of a GenuineD Contract.

6.4 Security Analysis of the Developed Decentralised Application (Genuine DAO)

The evaluation of the developed applications is based on the fact that current implementations and existing projects, as found by the systematic literature review, do have serious centralisation and security concerns. This evaluation, therefore, attempts to show the effectiveness of Genuine DAO and the extent to which the hypothesis is confirmed. This is done by running specific tests, on the Genuine DAO application platform and Polygonscan Amoy testnet, to ascertain the fulfilment of the requirements defined at the design stage.

6.4.1 Enhancing Security by Minimising Centralisation Risks

This section presents the evaluation of Requirements 1 and 2, pertaining to the claim that Genuine DAO ensures that decision-making is distributed, and single individuals or entities are prevented from exerting undue control. Security analysis for the proposed decentralised tendering application is outlined showing how the system enhances security by addressing these centralisation risks. The researcher conducted a review of Ethereum vulnerabilities, various attacks and techniques and tools to detect and mitigate them. By minimising this centralised control trust increases for participant as they will be involved in any decision-making process through a decentralised voting system.

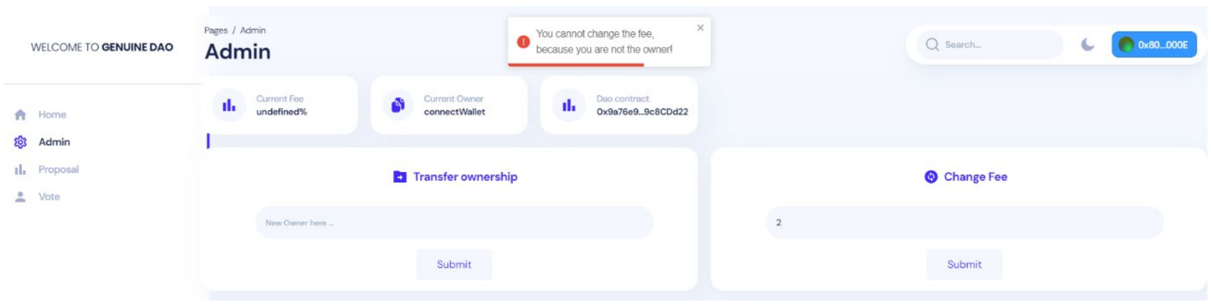
Authentication and Access control to prevent One Owner Control

Authentication includes verifying the identity or authorising users and entities interacting with a system. Digital signatures are used in Blockchain to verify the authenticity and integrity of transactions. A private key and a public key are used to ensure that only authorised users can interact with smart contracts and execute transactions.

As discussed in section 3.5.13, access control mechanisms can be implemented on smart contracts to specify which addresses have permission to perform certain actions. It includes function modifiers that are used to restrict the access and execution of specific functions within the contract. This process can enforce authentication requirements and increase the security of smart contracts against unauthorised access. However, it can cause a centralisation risk as mentioned earlier**.** The owner of a smart contract can set up administrative privileges and defines access control rules. Smart contract ownership can be transferred to different addresses through the use of smart contract functions. with the help of function modifier, the contract developer/owner can restrict access to certain functions. Therefore, this makes the smart contract such that only the contract creator/owner can execute critical functions and prevent non-owners from accessing specific functions.

With identifying the one owner control risk, the proposed blockchain-based DAO system, Genuine DAO, is developed in a way to minimise this risk**.** The developed Genuine DAO implements additional security levels to prevent one-owner control and its access to critical functions, perform sensitive operations such as moderating smart contract, transaction rate fee, minting tokens, transferring ownership, setting any address as validator, and voting on proposals. The "Genuine DAO" does this by enforcing automated rules that are encoded in smart contracts to follow a decentralised decision-making process and transfer the ownership of system to a contract address called GenuineD contract to control critical functions. Figure 39 below, shows how the new owner, which is a GenuineD contract, would not be able to change a critical function such as the transaction rate fee. The application enforces participants to create a proposal and follow the decentralised voting process. The successful proposal with recommended transaction rate fee will be applied to the system without any control from individuals or entities.

The transfer of ownership to the GenuineD contract means that this contract would take control of all critical functions within the developed application. Developers of new contracts are not allowed to change any critical functions. This is achieved by transferring the ownership to the GenuineD contract, which is decentralisation-enforcing and uses a DAO structure and a voting process for making any decisions for all submitted proposals. Without GenuineD, any developer would simply be able to change a critical function and therefore gain unfair advantage.



You cannot change the fee, because you are not the owner!

Current Fee undefined%

Current Owner connectWallet

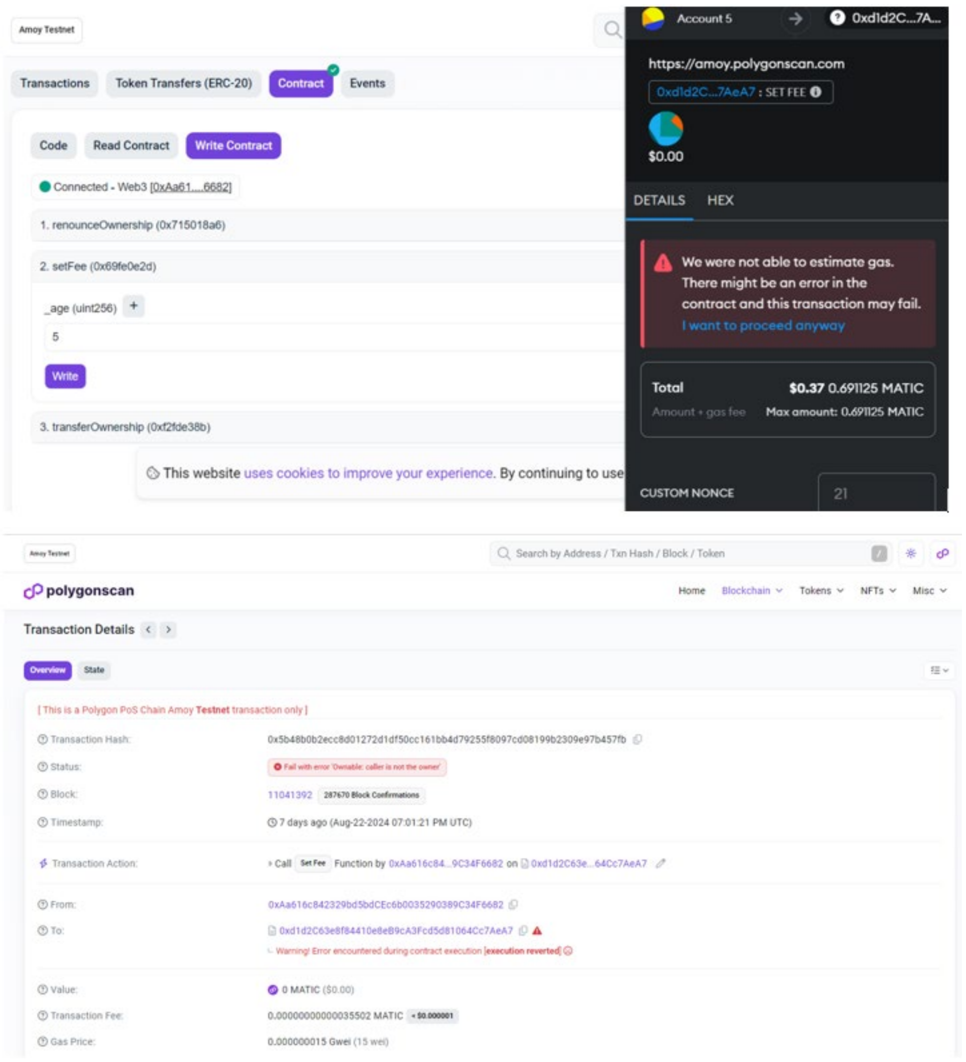
Dao contract 0x9a76e9...9c8CDd22

New Owner here ...

2

Figure 39 - A Front-End Example Showing How Genuine DAO Prevents One Owner Control.

To evaluate this decentralisation approach further, the Genuine DAO ownership has been tested through Polygonscan (Amoy testnet). The figure 40 shows how the system prevents changing the transaction rate fee after transferring the ownership to the GenuineD contract. This confirms that the Genuine DAO is successful in preventing any developer from changing the transaction rate fee without going through a voting process.



Connected - Web3 [0xAa61....6682]

1. renounceOwnership (0x715018a6)

2. setFee (0x69fe0e2d)

_age (uint256)

5

3. transferOwnership (0xf2fde38b)

This website uses cookies to improve your experience. By continuing to use

Account 5

0xd1d2C...7A...

https://amoy.polygonscan.com

0xd1d2C...7AeA7 : SET FEE

$0.00

We were not able to estimate gas. There might be an error in the contract and this transaction may fail.

I want to proceed anyway

Total $0.37 0.691125 MATIC

Amount + gas fee Max amount: 0.691125 MATIC

CUSTOM NONCE 21

Transaction Details

[This is a Polygon PoS Chain Amoy **Testnet** transaction only]

Transaction Hash:	0x5b48b0b2ecc8d01272d1df50cc161bb4d79255f8097cd08199b2309e97b457fb
Status:	Fail with error 'Ownable: caller is not the owner'
Block:	11041392 287670 Block Confirmations
Timestamp:	7 days ago (Aug-22-2024 07:01:21 PM UTC)
Transaction Action:	Call Set Fee Function by 0xAa616c84...9C34F6682 on 0xd1d2C63e...64Cc7AeA7
From:	0xAa616c842329bd5bdCEc6b0035290389C34F6682
To:	0xd1d2C63e8f84410e8eB9cA3Fcd5d81064Cc7AeA7 Warning! Error encountered during contract execution [**execution reverted**]
Value:	0 MATIC ($0.00)
Transaction Fee:	0.00000000000035502 MATIC < $0.000001
Gas Price:	0.000000015 Gwei (15 wei)

Figure 40 - An Example, as captured in Polygonscan Amoy Testnet, Showing How Genuine DAO Prevents One Owner Control.

The above shows that Genuine DAO makes a significant contribution to enforcing decentralisation and therefore preventing related security risks associated with lack of distributed decision-making (Requirement 1) and single point of failure (Requirement 2).

6.4.2 Enhancing Security by Preventing Frontrunning Attacks

One of the security issues, and a notorious heist within the contract layer on an Ethereum platform, is the frontrunning attack which is discussed in section 3.5.3. In brief, Frontrunning is the act of placing a transaction in a queue with the knowledge of a future transaction (Varun, Palanisamy and Sural, 2022). Frontrunning has become prevalent in finance markets, where brokers with privileged access to insider knowledge regarding their clients' trading decisions may prioritise their own trading actions over those of their clients. This unethical practice allows them to potentially gain additional profits. The development of a decentralised application with blockchain and DAO prevents the central authorities from regulating frontrunning.

In this research a solution is proposed during the development of Genuine DAO to reduce the risk of frontrunning. This is achieved by making the transaction non-profitable to the frontrunners. Furthermore, duplicated proposals are avoided by defining an incremental counter inside the smart contract. Therefore, each proposal would have its unique ID that is produced by smart contracts. The proposalId is generated by hashing the proposal data (targets, values, calldatas, signatures). To prevent duplicated proposals, the system checks if the proposalId exists. Furthermore, it checks the last proposalId, that is provided by *msg.sender,* and its state.

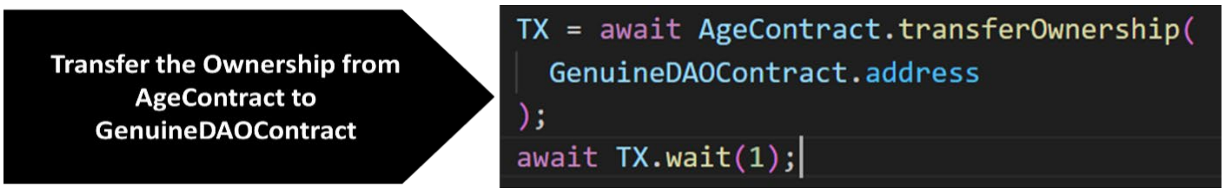
If a frontrunner attempts to replicate a proposal with the same data, it would result in a distinct proposalId. Consequently, a frontrunner would not be able to take any benefit from this frontrunning action. The effectiveness of the proposed solution is analysed through experiments which are discussed in detail below.

All tools and libraries that were used to test the frontrunning process have been outlined in section 5.4. The frontrunning test code is located in the Genuine DAO, *Contract/TestFrontRunning.js path*. Libraries such as Hardhat and Chai are imported to assist with the testing process. Hardhat contains Ethers that are used to work with contracts. The e*xpect* function is imported from Chai library to define assertions in the test case. Variables are defined to store users, smart contracts and blockchain information such as transactions, block number, proposal state. The time-based access control is implemented. The *beforeEach()* function is used to get addresses from ethers, deploy and initialise smart contracts including main contract, AgeContract and other contracts NFTContract, GenuineDAOContract and GenuineDStorageContract.

- Deploy AgeContract to change the transaction rate fee by the owner.
- Deploying NFTContract to create and mint tokens for users to contribute in voting process.

- Deploying GenuineDAOContract to create proposals and voting process to change the transaction rate fee.
- Deploying GenuineDStorageContract that plays the role of data storage for the main GenuineDAOContract.

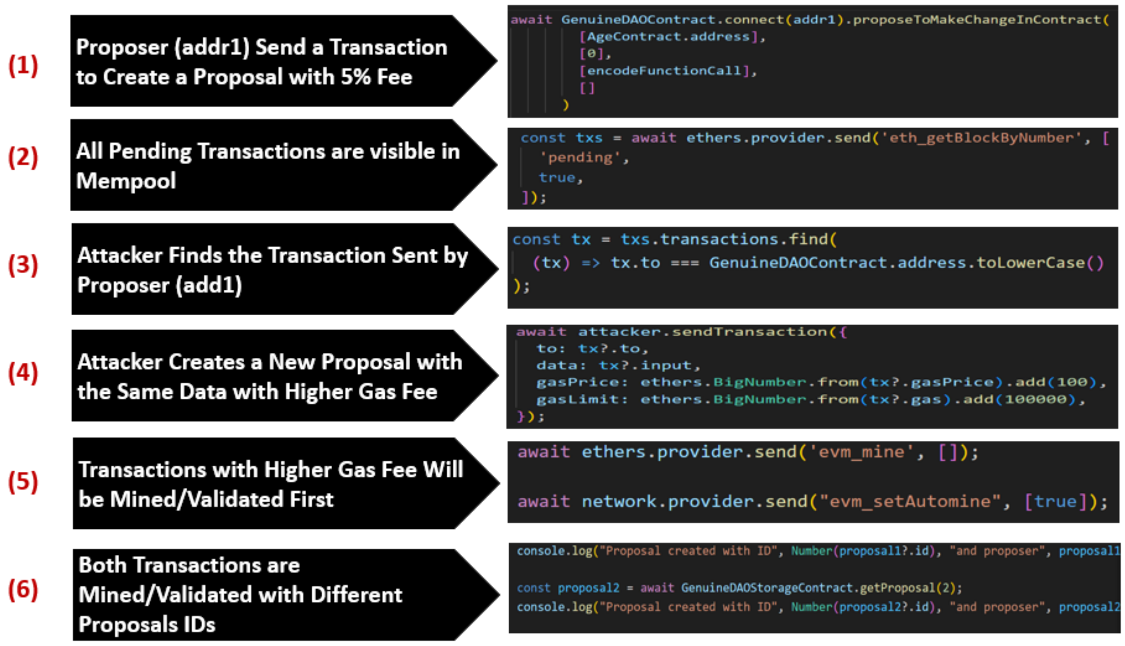
The important action of transferring the ownership from AgeContract to GenuineDAOContract is shown in figure 41. As a result, the contract creator/owner would not be able to control critical functions or change the transaction rate fee. GenuineDAOContract will take control of the system and would decide about transaction rate fee through a voting process.



```
TX = await AgeContract.transferOwnership(
  GenuineDAOContract.address
);
await TX.wait(1);
```

Figure 41 - Transfer the Ownership from AgeContract to GenuineDAOContract.

The process of frontrunning attacks is illustrated in figure 42:


(1)
Proposer (addr1) Send a Transaction to Create a Proposal with 5% Fee
(2)
All Pending Transactions are visible in Mempool
(3)
Attacker Finds the Transaction Sent by Proposer (add1)
(4)
Attacker Creates a New Proposal with the Same Data with Higher Gas Fee
(5)
Transactions with Higher Gas Fee Will be Mined/Validated First
(6)
Both Transactions are Mined/Validated with Different Proposals IDs

```
await GenuineDAOContract.connect(addr1).proposeToMakeChangeInContract(
        [AgeContract.address],
        [0],
        [encodeFunctionCall],
        []
      )
```

```
const txs = await ethers.provider.send('eth_getBlockByNumber', [
  'pending',
  true,
]);
```

```
const tx = txs.transactions.find(
  (tx) => tx.to === GenuineDAOContract.address.toLowerCase()
);
```

```
await attacker.sendTransaction({
  to: tx?.to,
  data: tx?.input,
  gasPrice: ethers.BigNumber.from(tx?.gasPrice).add(100),
  gasLimit: ethers.BigNumber.from(tx?.gas).add(100000),
});
```

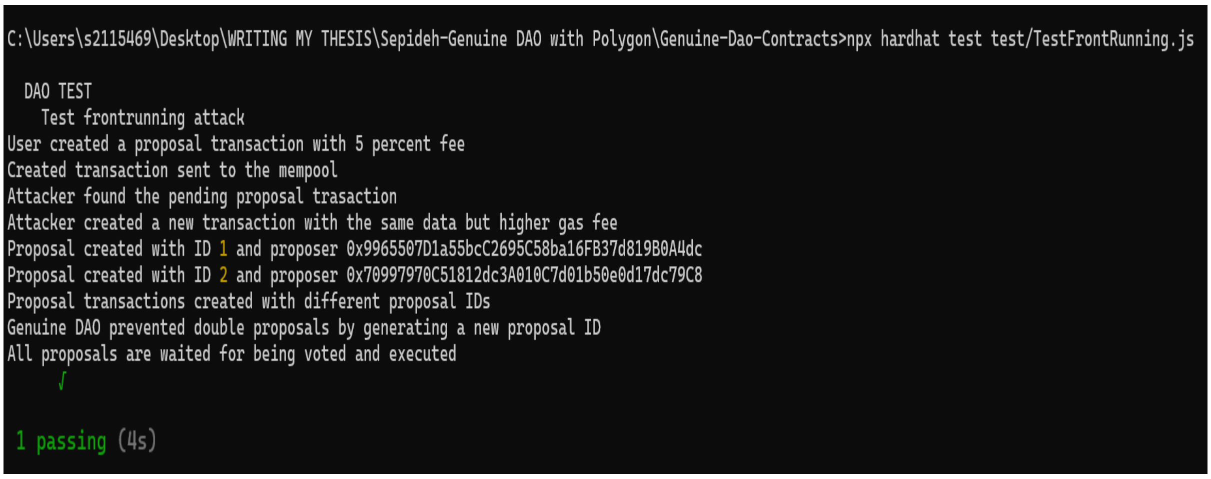
```
await ethers.provider.send('evm_mine', []);

await network.provider.send("evm_setAutomine", [true]);
```

```
console.log("Proposal created with ID", Number(proposal1?.id), "and proposer", proposal1

const proposal2 = await GenuineDAOStorageContract.getProposal(2);
console.log("Proposal created with ID", Number(proposal2?.id), "and proposer", proposal2
```

Figure 42 - The Process of Frontrunning Attacks.

Submitted proposals with different proposalIds with any suggested transaction rate fee will go through the voting process. Voters need to submit their votes for each proposal. The successful proposal with maximum votes, moves into the queue and execution steps. Time-based access control (Timelock) is implemented from the creation of a proposal until it gets executed. The proposed solution not only transfers ownership from the contract creator/owner to the GenuineDAOContract but also prevents double proposals by generating unique proposalIds. As a result, this front-running attack becomes an unprofitable action for the attacker as any proposal should go through a voting process for acceptance or rejection by participants.

Figure 43 shows the result of frontrunning test runs by Hardhat.



```
C:\Users\s2115469\Desktop\WRITING MY THESIS\Sepideh-Genuine DAO with Polygon\Genuine-Dao-Contracts>npx hardhat test test/TestFrontRunning.js

  DAO TEST
    Test frontrunning attack
User created a proposal transaction with 5 percent fee
Created transaction sent to the mempool
Attacker found the pending proposal trasaction
Attacker created a new transaction with the same data but higher gas fee
Proposal created with ID 1 and proposer 0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc
Proposal created with ID 2 and proposer 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
Proposal transactions created with different proposal IDs
Genuine DAO prevented double proposals by generating a new proposal ID
All proposals are waited for being voted and executed
      √

  1 passing (4s)
```

Figure 43 - Genuine DAO - Frontrunning Test Result.

The above is further illustrated in Figure 44, 45 and 46, which show the same results of frontrunning test runs by Genuine DAO through Polygonscan (Amoy testnet).

(1) The proposer sends the transaction and submits “the original proposal” with a unique ProposalId submitted with suggested transaction rate fee

(2) All pending transactions are visible in the Mempool

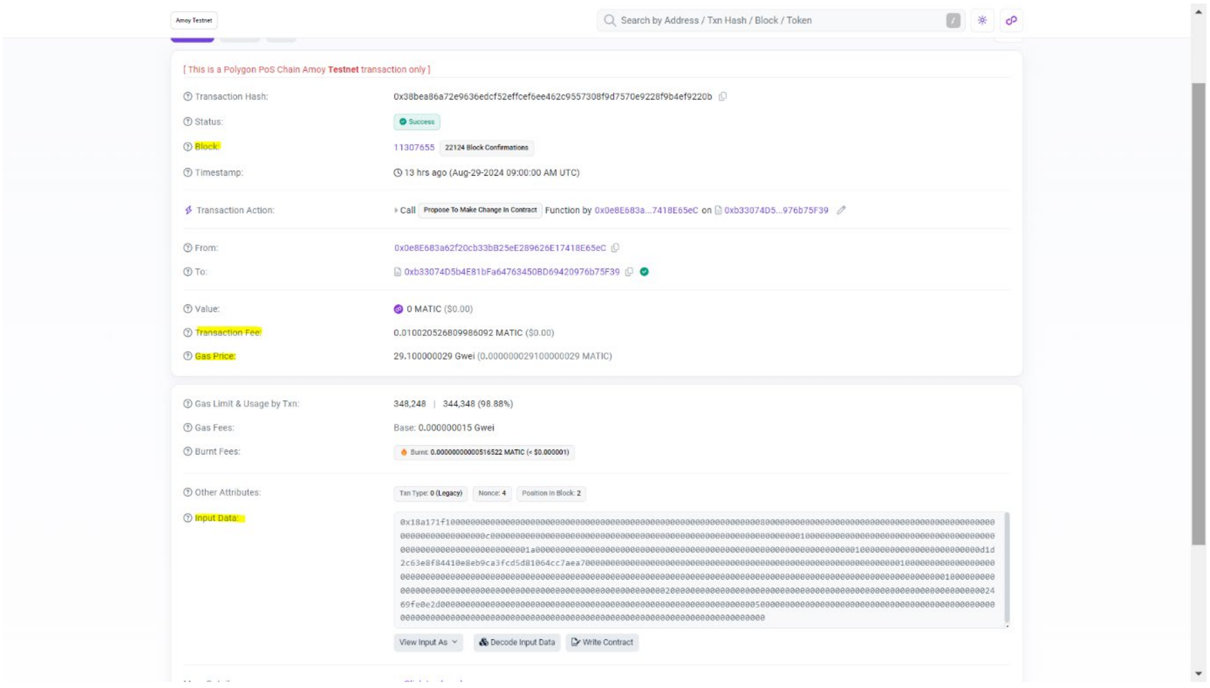
(3) Frontrunners monitor all transactions, identify the transaction, and its associated “original proposal”

(4) The frontrunner creates a new proposal, using the same data, and prioritises the transaction by offering a higher gas fee.

(5) Both proposals will be validated with different ProposalIds and will go through the voting process.

(6) The transaction with higher gas fee will be validated first.

(7) Both proposals may receive enough votes to be successful. However, since the frontrunner’s proposal uses the same data as the original, both proposals suggest the same transaction rate fee (%5 as shown in figure 46). As a result, if both proposals pass the voting process, the final transaction fee remains unchanged. This frontrunning activity does not financially benefit the frontrunner. This is how Genuine DAO makes it non-profitable for frontrunners.


Amoy Testnet

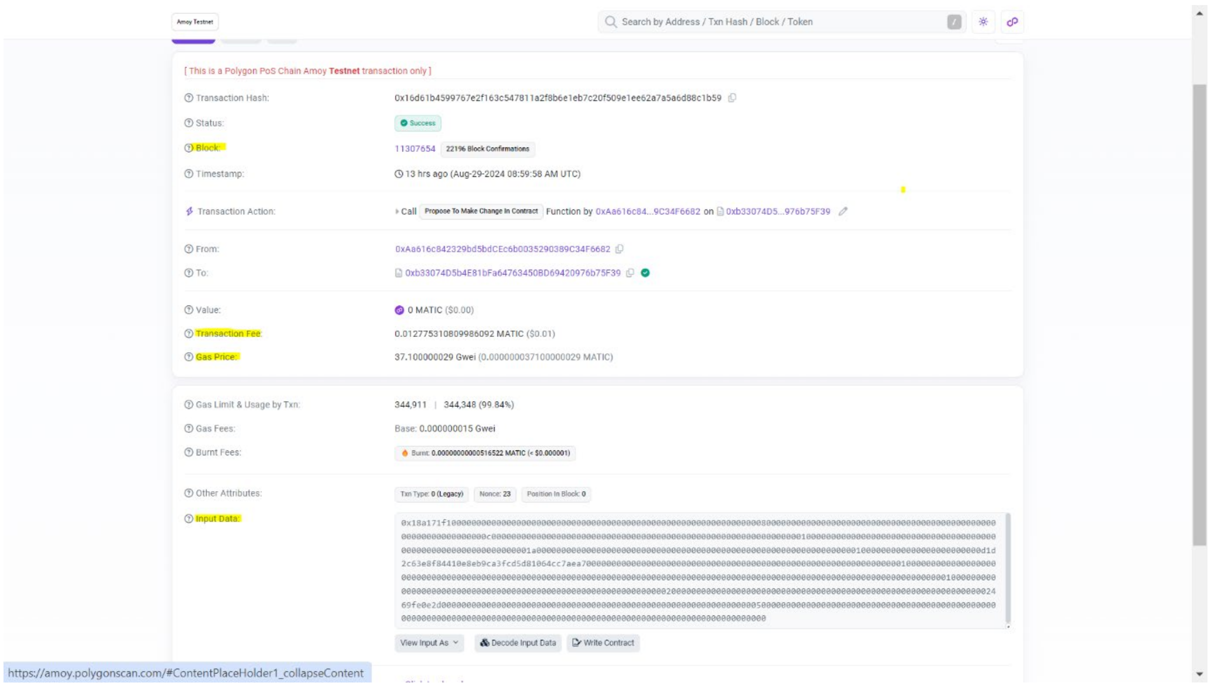
Search by Address / Txn Hash / Block / Token

[This is a Polygon PoS Chain Amoy **Testnet** transaction only]

Transaction Hash:	0x38bea86a72e9636edcf52effcef6ee462c9557308f9d7570e9228f9b4ef9220b
Status:	Success
Block:	11307655 22124 Block Confirmations
Timestamp:	13 hrs ago (Aug-29-2024 09:00:00 AM UTC)
Transaction Action:	Call Propose To Make Change In Contract Function by 0x0e8E683a...7418E65eC on 0xb33074D5...976b75F39
From:	0x0e8E683a62f20cb33bB25eE289626E17418E65eC
To:	0xb33074D5b4E81bFa64763450BD69420976b75F39
Value:	0 MATIC ($0.00)
Transaction Fee:	0.010020526809986092 MATIC ($0.00)
Gas Price:	29.100000029 Gwei (0.000000029100000029 MATIC)
Gas Limit & Usage by Txn:	348,248 | 344,348 (98.88%)
Gas Fees:	Base: 0.000000015 Gwei
Burnt Fees:	Burnt: 0.0000000000516522 MATIC (< $0.000001)
Other Attributes:	Txn Type: 0 (Legacy) Nonce: 4 Position In Block: 2
Input Data:	

```
0x18a171f1000000000000000000000000000000000000000000000000000000000000008000000000000000000000000000000000000000000000
00000000000000000c00000000000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000
00000000000000000000000001a00000000000000000000000000000000000000000000000000000000000000010000000000000000000000000d1d
2c63e8f84410e8eb9ca3fcd5d81064cc7aea70000000000000000000000000000000000000000000000000000000000000001000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001000000000
0000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000024
69fe0e2d000000000000000000000000000000000000000000000000000000000000000500000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure 44 Transaction receipt of the submitted original proposal, as displayed in Polygonscan Amoy Testnet.



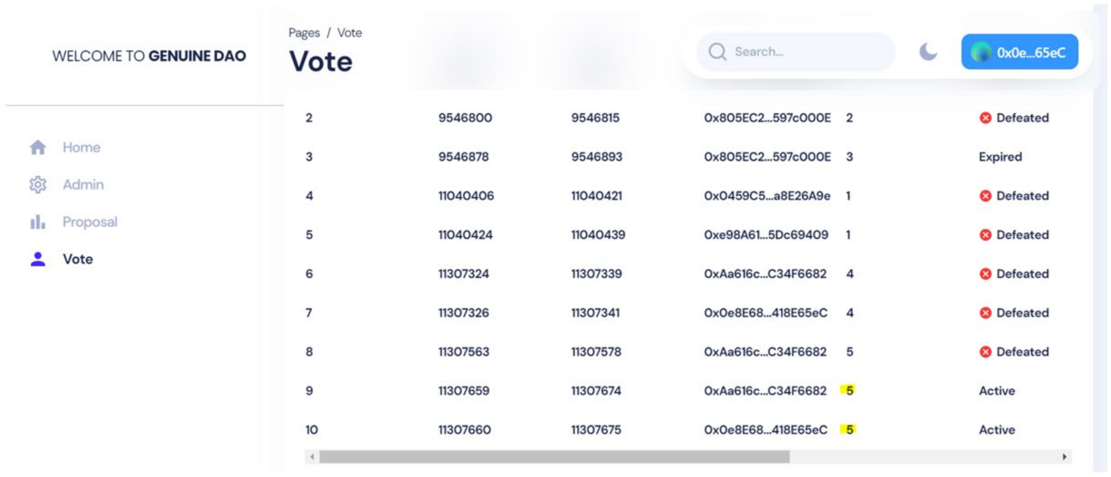
[This is a Polygon PoS Chain Amoy **Testnet** transaction only]

Transaction Hash:	0x16d61b4599767e2f163c547811a2f8b6e1eb7c20f509e1ee62a7a5a6d88c1b59
Status:	Success
Block:	11307654 22196 Block Confirmations
Timestamp:	13 hrs ago (Aug-29-2024 08:59:58 AM UTC)
Transaction Action:	Call Propose To Make Change In Contract Function by 0xAa616c84...9C34F6682 on 0xb33074D5...976b75F39
From:	0xAa616c842329bd5bdCEc6b0035290389C34F6682
To:	0xb33074D5b4E81bFa64763450BD69420976b75F39
Value:	0 MATIC ($0.00)
Transaction Fee:	0.012775310809986092 MATIC ($0.01)
Gas Price:	37.100000029 Gwei (0.000000037100000029 MATIC)
Gas Limit & Usage by Txn:	344,911 | 344,348 (99.84%)
Gas Fees:	Base: 0.000000015 Gwei
Burnt Fees:	Burnt: 0.00000000000516522 MATIC (< $0.000001)
Other Attributes:	Txn Type: 0 (Legacy) Nonce: 23 Position In Block: 0
Input Data:	

```
0x18a171f1000000000000000000000000000000000000000000000000000000000000008000000000000000000000000000000000000000000000
00000000000000000c00000000000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000000
0000000000000000000000000001a000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000d1d
2c63e8f84410e8eb9ca3fcd5d81064cc7aea7000000000000000000000000000000000000000000000000000000000000001000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001000000000
00000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000000024
69fe0e2d00000000000000000000000000000000000000000000000000000000000000050000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
```

https://amoy.polygonscan.com/#ContentPlaceHolder1_collapseContent

Figure 45 Transaction receipt of the submitted the frontrunner proposal, as displayed in Polygonscan Amoy Testnet



2	9546800	9546815	0x805EC2...597c000E	2	Defeated
3	9546878	9546893	0x805EC2...597c000E	3	Expired
4	11040406	11040421	0x0459C5...a8E26A9e	1	Defeated
5	11040424	11040439	0xe98A61...5Dc69409	1	Defeated
6	11307324	11307339	0xAa616c...C34F6682	4	Defeated
7	11307326	11307341	0x0e8E68...418E65eC	4	Defeated
8	11307563	11307578	0xAa616c...C34F6682	5	Defeated
9	11307659	11307674	0xAa616c...C34F6682	5	Active
10	11307660	11307675	0x0e8E68...418E65eC	5	Active

Figure 46 Active state of the submitted frontrunner proposal (9) and original proposal (10)

This section has shown that the developed application, Genuine DAO, makes a significant contribution towards enforcing decentralisation through a distributed decision-making process and a mechanism that prevents one-owner control. The evaluation phase has also confirmed that the application contributes to a more secure execution of smarts contracts, through reducing the risks of frontrunning attacks that, otherwise, would give frontrunners an unfair advantage, which poses a significant threat to the Blockchain system.

6.5 Performance Analysis of the Developed Decentralised Application (Genuine DAO)

As mentioned earlier, this study uses the Ethereum network, which stands as the most prevalent and largest Turing-complete blockchain by market capitalisation. Therefore, the Ethereum network can get congested if many users seek to interact with the network concurrently due to the maximum throughput limit. Moreover, given the elevated price of Ether, the resulting total cost could exceed initial expectations.

Initial development of the Genuine DAO decentralised application utilised the Sepolia testnet. Sepolia uses the PoS consensus mechanism and provides an opportunity for developers to deploy their contracts on Ethereum's mainnet environment. However, in an attempt to enhance scalability and reduce transaction cost, Genuine DAO was migrated to the Polygon network, Matic/Amoy testnets. The following sections discuss the enhancement of the system performance when integrating a Polygon network within Ethereum.

6.5.1 Enhanced Performance by Using Polygon

Ethereum operates as a layer 1 Blockchain, where transactions are processed directly on the mainnet. It has faced challenges with throughput, latency, storage and cost (Hafid *et al.,* 2020). Ethereum can be costly and slow during period of high network traffic and running complex operations. To address this, a Layer-2 scaling solution such as Polygon was introduced and integrated within an Ethereum network. This solution has already been identified as having the potential to empower decentralised applications to access the Ethereum network while significantly enhancing scalability and transaction throughput by using Rollups technology (Neiheiser *et al.,* 2023). Polygon uses a Zero- Knowledge rollups (ZK rollups) with the aim to bolster throughput and enhance scalability of Ethereum without compromising decentralisation. ZK rollups are designed to process transactions of-chain, alleviating the computation on the base layer (Alchemy and Werkheiser, 2022).

- **Storage:** Amount of data stored on Ethereum increases rapidly due to having more transactions, deploying more DApps and updating the state of every smart contract. This growth creates a great demand on storage and can lead to longer transaction processing times. Polygon reduces the storage challenges by processing transactions through layer 2 solutions (Hafid *et al.,* 2020).
- **Throughput:** Due to throughput limitation, not all pending transactions will be promptly confirmed and be part of a block. Therefore, the higher the gas fee users are willing to pay for their transactions, the more likely that their transaction will be included in a block. This combination of throughput constraints and high demand for network transactions compels users to pay elevated transaction fees (Neiheiser *et al.,* 2023).

  According to Arthur (2024) Ethereum can only process around 20-30 transactions per second. On the other hand, Polygon claims to have faster transaction speed and achieve up to 7,000 transactions a second. At the moment of writing, Polygon has an average block time of 2.1 and a gas limit per block of 30 millions (Polygon, 2024). The following calculations can be used to find out information about the Polygon, based on data available on Polygon network (Sguanci *et al.,* 2021).

$$Average\ Transactions\ per\ block\ = \frac{Total\ transactions}{Total\ blocks}$$

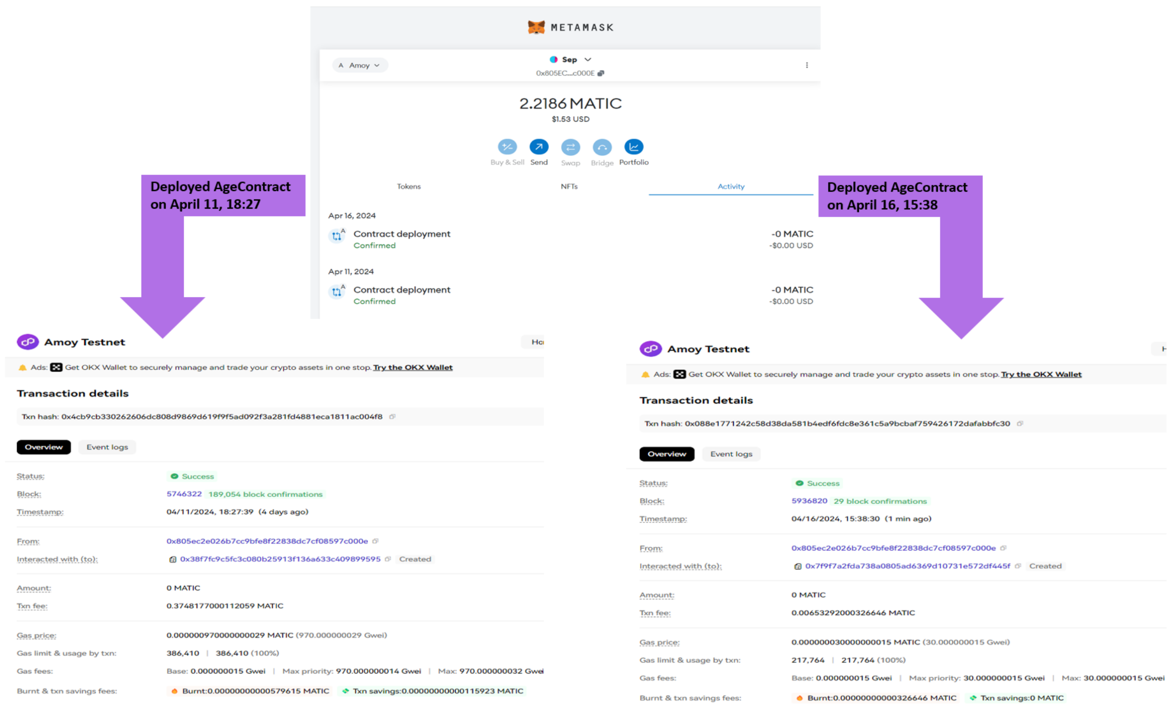
$$Average\ gas\ per\ transaction\ = \frac{Block\ gas\ limit}{Average\ transaction\ per\ block}$$

$$Throughput\ = \frac{Average\ transaction\ per\ block}{Average\ block\ time}$$

$$Transaction\ Cost\ = gasUsed\ * \frac{gasPrice}{1000000000} * Ether/Matic\ Price$$

Cost: An essential consideration in any network is network congestion, which can significantly affect transaction fees. The level of congestion would vary throughout the day for both Ethereum and Polygon. The fluctuation and detailed graphs of gas prices can be found on Etherscan and Polygon website (Etherscan, 2024; Polygon, 2024), resulting in the same transaction costs and less during certain times. Notably, contract deployment, interacting with different smart contracts, executing complex functions, storage operations have massive impact on the gas fee.

- For example, AgeContract has been deployed on Polygon network, Amoy testnet in a different day/time, resulting in different gas usage, gas fee, and transaction fee.


Deployed AgeContract on April 11, 18:27
Deployed AgeContract on April 16, 15:38

METAMASK

Amoy | Sep | 0x805EC...c000E

2.2186 MATIC
$1.53 USD

Buy & Sell | Send | Swap | Bridge | Portfolio

Tokens | NFTs | Activity

Apr 16, 2024
Contract deployment — Confirmed — -0 MATIC -$0.00 USD

Apr 11, 2024
Contract deployment — Confirmed — -0 MATIC -$0.00 USD

Amoy Testnet

Ads: Get OKX Wallet to securely manage and trade your crypto assets in one stop. Try the OKX Wallet

Transaction details

Txn hash: 0x4cb9cb330262606dc808d9869d619f9f5ad092f3a281fd4881eca1811ac004f8

Overview | Event logs

Status:	Success
Block:	5746322 189,054 block confirmations
Timestamp:	04/11/2024, 18:27:39 (4 days ago)
From:	0x805ec2e026b7cc9bfe8f22838dc7cf08597c000e
Interacted with (to):	0x38f7fc9c5fc3c080b25913f136a633c409899595 Created
Amount:	0 MATIC
Txn fee:	0.3748177000112059 MATIC
Gas price:	0.000000970000000029 MATIC (970.000000029 Gwei)
Gas limit & usage by txn:	386,410 | 386,410 (100%)
Gas fees:	Base: 0.000000015 Gwei | Max priority: 970.000000014 Gwei | Max: 970.000000032 Gwei
Burnt & txn savings fees:	Burnt:0.00000000000579615 MATIC Txn savings:0.00000000000115923 MATIC

Amoy Testnet

Ads: Get OKX Wallet to securely manage and trade your crypto assets in one stop. Try the OKX Wallet

Transaction details

Txn hash: 0x088e1771242c58d38da581b4edf6fdc8e361c5a9bcbaf759426172dafabbfc30

Overview | Event logs

Status:	Success
Block:	5936820 29 block confirmations
Timestamp:	04/16/2024, 15:38:30 (1 min ago)
From:	0x805ec2e026b7cc9bfe8f22838dc7cf08597c000e
Interacted with (to):	0x7f9f7a2fda738a0805ad6369d10731e572df445f Created
Amount:	0 MATIC
Txn fee:	0.00653292000326646 MATIC
Gas price:	0.000000030000000015 MATIC (30.000000015 Gwei)
Gas limit & usage by txn:	217,764 | 217,764 (100%)
Gas fees:	Base: 0.000000015 Gwei | Max priority: 30.000000015 Gwei | Max: 30.000000015 Gwei
Burnt & txn savings fees:	Burnt:0.00000000000326646 MATIC Txn savings:0 MATIC

Figure 47- Deploy AgeContract on Polygon in a Different Day/Time

According to the current information available on Coinbase, price of Matic (Polygon network) is £ 0.58 whereas Ether price is £ 2,567.74 (Coinbase, 2024). This makes Polygon an attractive option for users and developers to access faster and cheaper transactions compared to the Ethereum Mainnet.

Table 9 illustrates the transaction cost of deploying an AgeContract on both networks, Ethereum mainnet and Polygon. The gas used is very similar to the gasUsed in Gas optimisation on figure 48. It is slightly different due to network congestion and time of deployment. The transaction costs are calculated according to the formula explained in section 2.4.

Platform	Smart Contract	Gas Used	Gas Fee (Gwei)	Total Gas Fee (Eth/Matic)	Exchange Rate (GBP)	Total Cost (GBP)
Ethereum	AgeContract	217,764	2.281722781	0.000496877079681684 ETH	2567.74	1.2758

Polygon	AgeContract	217,764	30.000000015	0.00653292000326646 MATIC	0.58	0.0038

Table 9 - Comparing cost of AgeContract Deployment on Polygon vs. Ethereum.

6.5.2 Enhanced Performance Using Gas Optimisation

As mentioned in Section 2.4, researchers like (Marchesi *et al.,* 2020; Li, 2021; Bashir, 2020) claimed that gas fees have been a significant challenge and have had obvious implications on the Ethereum network. The transition from Ethereum 1.0 to Ethereum 2.0 not only enhances the security, scalability and speed, it also reduces the transaction cost. However, the Ethereum gas fee remains quite high even after the transition to Ethereum 2.0 due to different factors, such as unfamiliarity of developers with smart contracts and EVM (Kong *et al.,* 2022). Same researchers suggested optimisation tools and patterns at the source code level to minimise the cost of gas which are explained in Section 2.4.

Writing efficient and optimised code plays a crucial role in reducing gas consumption and transaction cost. The model application presented in Section 3.13 exemplifies the best practice to develop a secure smart contract. This model can aid to reduce the gas consumption.

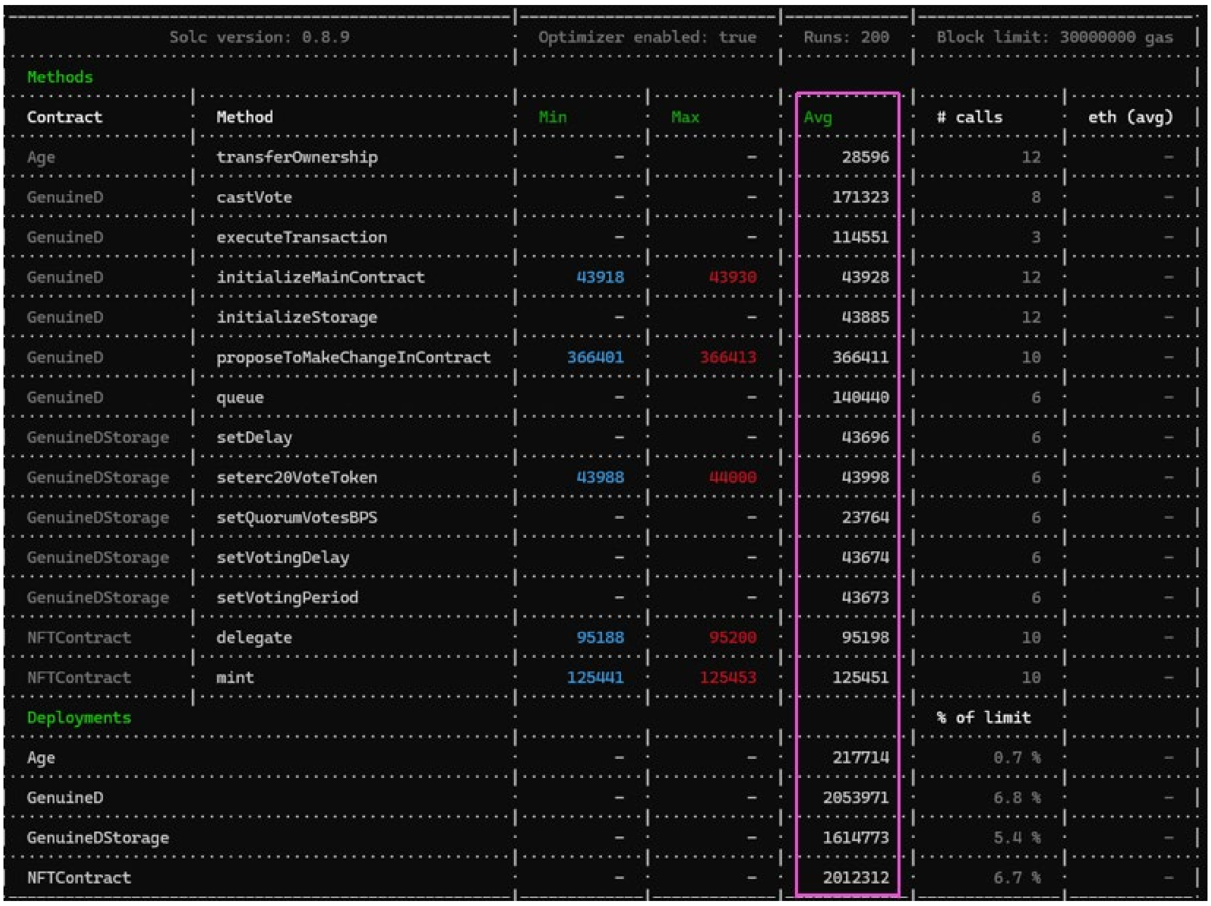
This work attempts to use gas saving best practice and patterns that are recommended by researchers (Marchesi *et al.,* 2020; Baldauf, Sonnleitner and Kurz, 2023) to develop Genuine DAO to decrease interaction costs. The techniques that have been used in development of smart contracts in Genuine DAO are depicted in Table 10.

Best Practices to Reduce Gas Consumption	
Name	**Description**
Simplicity over complex contracts	Create clear and simple smart contracts to improve security, save gas and easier to detect bugs.
Code quality	Use graphing code Flow analysis and reviewing the code by security analyser and experienced programmer.
Code reuse	Import code from trusted libraries such as OpenZeppelin.
Robust testing	Use of Hardhat to test smart contract and test over multiple stages on Ethereum and Polygon testnets.

Solidity optimiser	Use of optimiser with the default settings of 200 runs.
Limit storage	store state variable as a local variable in memory to save gas for multiple calls.
Packing variables	Use of struct to pack variables together to use less storage
Use of *external* function	Use of external function when it is intended to be called externally. The external function is *Read Only* and is stored in Calldata memory, which is cheaper than the public function that is stored in memory.
Mapping	Use of mapping for efficient data access to optimise storage operations

Table 10 - Genuine DAO - Best Practices to Reduce Gas Consumption.

Gas optimisation is crucial for reducing deployment and execution costs for the end users. In this work, Solidity optimiser has been turned on and the created contracts are optimised for 200 runs by applying various optimisation techniques. The number of runs roughly indicates the frequency of execution for each opcode in the deployed code throughout the contract's lifetime. Figure 48 illustrates the experimental result of gas consumption of numerous operations within Genuine DAO as indicated in “Avg” column. The figure is generated using Hardhat Gas Reporter. The report indicates that contract deployment and all executions have been optimised, with the average gas consumption below the maximum allowable limit. There is no doubt that implementing gas saving best practices and utilising Solidity optimiser, can lead to the development of optimised smart contracts that reduce gas usage and overall transaction costs.



```
-----------------------------------------------|--------------------------|-------------|-----------------------------
            Solc version: 0.8.9                ·  Optimizer enabled: true  ·  Runs: 200  ·  Block limit: 30000000 gas  │
···············································|···························|·············|·····························
  Methods                                                                                                              │
·················|·····························|·············|·············|·············|·············|···············
  Contract       ·  Method                     ·  Min        ·  Max        ·  Avg        ·  # calls    ·  eth (avg)    │
·················|·····························|·············|·············|·············|·············|···············
  Age            ·  transferOwnership          ·          -  ·          -  ·      28596  ·         12  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineD       ·  castVote                   ·          -  ·          -  ·     171323  ·          8  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineD       ·  executeTransaction         ·          -  ·          -  ·     114551  ·          3  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineD       ·  initializeMainContract     ·      43918  ·      43930  ·      43928  ·         12  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineD       ·  initializeStorage          ·          -  ·          -  ·      43885  ·         12  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineD       ·  proposeToMakeChangeInContract  ·  366401  ·     366413  ·     366411  ·         10  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineD       ·  queue                      ·          -  ·          -  ·     140440  ·          6  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineDStorage  ·  setDelay                 ·          -  ·          -  ·      43696  ·          6  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineDStorage  ·  seterc20VoteToken        ·      43988  ·      44000  ·      43998  ·          6  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineDStorage  ·  setQuorumVotesBPS        ·          -  ·          -  ·      23764  ·          6  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineDStorage  ·  setVotingDelay           ·          -  ·          -  ·      43674  ·          6  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  GenuineDStorage  ·  setVotingPeriod          ·          -  ·          -  ·      43673  ·          6  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  NFTContract    ·  delegate                   ·      95188  ·      95200  ·      95198  ·         10  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  NFTContract    ·  mint                       ·     125441  ·     125453  ·     125451  ·         10  ·          -    │
·················|·····························|·············|·············|·············|·············|···············
  Deployments                                  ·                                         ·  % of limit ·               │
···············································|·············|·············|·············|·············|···············
  Age                                          ·          -  ·          -  ·     217714  ·      0.7 %  ·          -    │
···············································|·············|·············|·············|·············|···············
  GenuineD                                     ·          -  ·          -  ·    2053971  ·      6.8 %  ·          -    │
···············································|·············|·············|·············|·············|···············
  GenuineDStorage                              ·          -  ·          -  ·    1614773  ·      5.4 %  ·          -    │
···············································|·············|·············|·············|·············|···············
  NFTContract                                  ·          -  ·          -  ·    2012312  ·      6.7 %  ·          -    │
-----------------------------------------------|-------------|-------------|-------------|-------------|---------------
```

Figure 48 - Output of Gas Optimisation of Genuine DAO - Hardhat Gas Reporter.

6.6 Issues and Challenges During the Development of Genuine DAO

There have been a number of technical challenges faced when developing, and deploying, the Genuine DAO application and when interacting with it on different testnets. Some of these challenges are summarised below.

- Writing smart contracts requires a full understanding of Solidity programming language, Ethereum virtual machine, tools and libraries. Not only there is a requirement to write a secure, user-friendly decentralised application but also the gas optimisation strategy added complexity to the whole development. The researcher had to complete a few training courses and get feedback from experienced developers who have been working on Blockchain and are familiar with its environment.

- Deploying smart contracts on different Testnets required setting up a development environment and a network configuration. Receiving Testnet Ether or Matic (Sepolia,

> Mumbai/Amoy) tokens can be challenging. Although these tokens are provided for free by faucet services, such as Alchemy, to allow developers to deploy and interact with smart contracts on testnets, testnets can become inaccessible or leave the developer with insufficient amount of token for interacting with decentralised applicationThis is due to different reasons such as network congestion, network upgrade or closing a faucet. This work started by working on an Ethereum Testnet called Goerli. After closing this faucet, the interaction was taking place on the Sepolia network. At the same time, for migration to a Polygon network, Mumbai faucet was used. Later, with closing this faucet, Amoy Testnet had been replaced.

All these challenges required collaboration with other researchers and developers in this field, experimentation, and continuous learning in the rapidly evolving blockchain ecosystem.

6.7 Expert Review

Once the implemented Genuine DAO was tested and validated by the Researcher, a qualitative analysis was carried out to ascertain the effectiveness of the proposed solution. As stated in Section 4.7.3, the analysis involved gathering experts' reviews through a questionnaire. Three experts were judiciously chosen from industry, based on their knowledge and experience working in the field of blockchain security. The full questionnaire and responses can be found in Appendix A. Below is a description of the analysis of feedback received from the participants.

6.7.1 "Genuine DAO" as a Solution to Address Centralisation Risks

Blockchain Security Analysts and Developers were asked to confirm whether the proposed Genuine DAO application does assist with reducing the security concerns of one owner control and centralisation risks. Participant 1 explained how the proposed "Genuine DAO" addresses potential centralisation risks, such as concentration of power, control, or decision-making authority:

> "In this DAO model, the ownership of the smart contracts is not owned by a wallet address and the ownership is transferred to the DAO smart contract so any change in the smart contract main functions and rules can only be executed by DAO members with the power of voting. This point resolves the centralised management issues of smart contracts". [Participant 1]

The ‘centralisation risks’ were emphasised by the other two participants in the context of DAO structure, decision making with voting process, and transferring the ownership to a contract address would have great impact to minimise centralisation risks.

> “Implementing a DAO structure that involves community participation in decision-making rather than relying on a single entity or an owner with elevated privilege would enhance security. Taking control off the developer and transferring the ownership to the DAO contract for critical functions is beneficial to reduce centralisation risks” [Participant 2].
>
> The other expert focused on highlighting the risks of centralisation in blockchain-based applications, particularly emphasising their significance in DeFi projects. This participant stated:
>
> “Centralisation is a significant risk factor in Blockchain-based projects especially in DeFi. It can expose protocols to single points of failure, making them susceptible to attacks or manipulation. Emphasising decentralisation and community governance helps distribute decision-making power and ensures no single entity holds excessive control. Active participation by a diverse group of stakeholders in protocol governance fosters transparency, reduces conflicts of interest, and promotes the adoption of risk-mitigating measures. Genuine DAO with DAO structure reduces the centralisation risks by transferring the ownership address to the contract address and other security measures in place” [Participant 3].

The responses highlighted that minimising the control over critical functions from contract creator/owner, transferring ownership to a contract without centralised control, and promoting decentralised structure (DAO) will have a significant impact in terms of mitigating centralisation risks.

6.7.2 Implementation of security controls and countermeasures

One Participant articulated the importance of implementing security controls and countermeasures to mitigate against one owner control risks. They stated that the implemented security features such as access control to critical functions and timelock would assist in avoiding risk of the ownership of smart contracts and enhance the security of the decentralised application.

> "We see some features in the DAO smart contract that actually have significant effect on the ownership improvements, such as the proposed changes by creating proposals, and voting to the proposals in the specific period of time. We can consider these changes as main improvements to avoid risks of the ownerships of smart contracts" [Participant 1].
>
> "Limiting the access control to critical function by implementing features such as contract ownership would enhance the security. In addition, having timelock at different stages of the proposal creation and voting process would improve the security" [Participant 2].
>
> Participant 3 provided information on the effectiveness of the implemented security measures in Genuine DAO and highlighted the importance of mitigating centralisation risks by mentioning an example where a protocol was exploited due to such risks. The Participant stated:
> "Security measures that have been taken in Genuine DAO would definitely enhance the security and reduce risk of ownership. MGold rug pull is an example of centralisation where founders used the private keys to drain the contracts of all funds. The founders decided to take the money and run, which is only possible due to the centralisation privilege of them holding the private keys. Having security measures like what have been implemented in Genuine DAO, such as timelock and DAO structure alongside with a multi signature wallet, would avoid this risk" [Participant 3].

The responses from, and reactions of, the experts that took part in this feedback evaluation process indicate that implementing security controls as proposed in Genuine DAO would minimise centralisation risks. By implementing security measures, Genuine DAO can reduce the likelihood of centralisation through contract creator/owner or an entity. These measures guard against potential threats and attacks and enhance users' trust in the system.

6.7.3 Leveraging Layer 2 Scaling Solutions

To evaluate the effectiveness of migrating Genuine DAO from Ethereum mainnet (Sepolia) testnet to Polygon (Mumbai/Amoy) testnets, the participants were asked to answer the following question:

> *How effective is the proposed "Genuine DAO" in leveraging layer 2 scaling solutions to enhance scalability and throughput and to minimise cost on Polygon network?*

The participants confirmed the importance of scalable network for blockchain transactions which can be achieved by using layer 2 scaling solutions. They stated:

> "What layer 2 solutions offer is more about taking the responsibly of the execution tasks from layer 1 and storing data in the layer 1. So, layer 2 solutions increase the scalability and decrease the fee at the same time of being secure and rely on layer 1. Therefore, Polygon as a layer 2 Blockchain offers lower fee because of the execution optimisations and better scalability". [Participant 1]

> "Layer 2 solutions provide a more scalable network as by processing transactions off-chain. This technology enables faster transaction confirmation times and decreases transaction cost by reducing the computational and storage costs. Many developers and users prefer to use layer 2 solutions over Ethereum mainnet" [Participant 2].

> "Scalability, throughput and cost are key factors in any network. Introducing layer 2 scaling would help with these challenges that Ethereum has faced" [Participant 3].

The Feedback received from these Security Analysts and Developers points to a clear advocacy to using Polygon, instead of Ethereum mainnet, with layer 2 scaling solution built on Ethereum. They seem to agree that layer 2 scaling solutions, such as Polygon, employ techniques that provide a more scalable network, as well as enhance throughput and reduce transaction cost.

Although this feedback is by no means a scientific evidence of the effectiveness of the proposed Genuine DAO, it does provide added confidence in the validity of the results of the work carried out in this thesis.

6.8 Summary

This chapter focused on the evaluation of the developed Genuine DAO and the effectiveness of the proposed security controls and countermeasures to minimise centralisation risks. The evaluation approach used is described along three main axes.

The first axis described the evaluation of the proposed seven-layer blockchain architecture, the designed conceptual taxonomy involving vulnerabilities, attacks and their consequences and the model application for best practice towards developing a more secure smart contract. These foundational components of this research were critically evaluated through a peer review process by several experts in the field of Blockchain technology as part of an academic paper publication.

Along the second axis lies the main contribution of this work, the proposed Genuine DAO application. The security controls and performance-enhancing practices of the developed application have been validated and tested as part of the evaluation process. This evaluation provides a thorough assessment of the Genuine DAO application, confirming its potential as a more secure, decentralised solution for managing smart contracts on the Blockchain.

Lastly, the third axis involved feedback evaluation from judiciously chosen experts. These independent experts were selected based on their knowledge and expertise in the Blockchain field.

Whilst the initial peer review gave confidence in the validity of the foundational pillars of this work, leading to implementing the Genuine DAO application, the experts' responses reinforced this Researcher's view that the proposed decentralised application, Genuine DAO, can contribute to enhancing security, scalability, throughput and cost. Security is enhanced by reducing centralised control, enforcing distributed decision-making, and preventing potential attacks such as frontrunning attacks, while scalability, throughput and cost are enhanced through a migration to a Polygon network.

Chapter 7: Conclusion and Recommendations

7.1 The Main Achievements

This research attempts to develop a better understanding of the vulnerabilities that exist within each of the Blockchain layers. This work adopted a seven-layer Blockchain architecture to understand the nature of the security vulnerabilities/threats within each layer. After carrying out a systematic investigation into existing vulnerabilities and attacks, this research suggested, for each of the seven layers of the Blockchain, a table detailing the location, the nature of vulnerabilities/attacks, the authors of key related works, and the detection tools or preventive techniques. The outcome of this investigation is summarised in a taxonomy for a seven-layer Blockchain architecture, describing the inter-relationships between vulnerabilities, attacks and the related consequences. This detailed investigation has exposed the contract layer as, arguably, the most vulnerable layer in a Blockchain architecture.

To enhance the security of blockchain transactions, a novel automated decentralised application, “Genuine DAO”, is proposed. This decentralised application has the potential to reduce security risks and improve the performance of blockchain networks. “Genuine DAO” achieves the reduction in security risks by minimising the threats inherent to centralisation, which can be caused by smart contracts’ owners, or developers, and by guarding against Frontrunning attacks. A robust procedure has been implemented to test and validate the developed “Genuine DAO” with the Hardhat framework and Polygonscan Amoy Testnet.

In addition, the “Genuine DAO” has been developed with the aim to improve the performance of the Blockchain network and reduce the cost. This is achieved through gas optimisation using a Polygon layer 2 scaling solution built on the Ethereum network.

Once testing and validation of the developed application were completed successfully, views from independent experts were sought to gain an external expert view on the effectiveness of the proposed solution. The feedback received from these experts provides an additional validation of the results, confirming that “Genuine DAO”, and the proposed security controls and countermeasures to minimise centralisation risks and prevent frontrunning attacks, provides an alternative solution for enhancing Blockchain security and performance within an Ethereum platform.

7.2 Research Limitations

Despite the above achievements this work is not without limitations. In addition to the challenges outlined in Chapter 6, most of the limitations of this research fall within the implementation phase of the work. In particular:

- Run on Testnets Rather Than Real-World Mainnets

  While testnets provide a valuable environment for the development and experimentation of decentralised application, developers are facing a number of challenges in using testnets effectively. Developers and researchers often encounter challenges such as network congestion, transaction execution delays and shortages of test tokens during deployment which can slow than the testing process. Furthermore, it has been a challenge testing how developed decentralised application interacts with various type of user inputs. Therefore, inadequately tested smart contracts can compromise the performance and security of blockchain. In addition, testnets can pose security risks that may not be apparent until deployment of smart contracts on a real-world mainnets (Habic, 2024).

- Solidity Programming Language

  Solidity is part of a rapidly evolving environment and the most popular language to write smart contracts on the Ethereum blockchain. There are a range of tools, libraries and frameworks available for development, debugging and test. However, it is not as mature a language as the more established programming languages. Discovered vulnerabilities and threats within the smart contracts, highlights developers need to implement robust testing practices and enhance their knowledge of Solidity's features to alleviate potential security risks. Therefore, writing a secure decentralised application which can withstand scrutiny from security analysers is no simple task.

- Involvement of independent experts in the evaluation phase

  Although using the three experts, who participated in the evaluation of the proposed solution, was of an immense value to this work, and is fit for the purposes for which it was used, involving a greater number of experts and users will definitely add value to identifying further areas for improvements. This is particularly important for future research.

7.3 Future Work

Although the proposed “Genuine DAO” fulfils the research aims and provides an enhanced security and performance for Ethereum Blockchains, more work is needed in this area.

One area that may need particular attention is to make implementing Blockchain applications more accessible to developers than it currently is. As far as the focus of this research is concerned, and to build on what has been achieved and advance the current research further, a couple of areas can be considered for further development:

- Use zero knowledge proof to enhance the user ‘privacy and secure authentication,
- Apply a multi signature wallet to minimise even further the potential for a single point of failure yielding additional enhancement to the security of the Blockchain network.
- Use the taxonomy developed as part of this work to implement security enhancement solutions for each of the seven layers of the Blockchain. Can the taxonomy be used for a wholistic enhancement solution?
- With the advent of Large Language Models there is an opportunity to focus on the integration of artificial intelligence techniques and tools and the “Genuine DAO” to analyse smart contract’s behaviour and data transactions.

References

1. Academy, S. (2023) *Solidity Data Storage and Management Strategies*. Available at: https://medium.com/coinmonks/solidity-data-storage-and-management-strategies-811700abda8a. (Accessed: 10 December 2023).
2. Achour, I., Ayed, S. and Idoudi, H. (2021) 'On the Implementation of Access Control in Ethereum Blockchain', in *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*. IEEE, pp. 483–487. doi: 10.1109/3ICT53449.2021.9581591.
3. Adi, S. H. (2022) *How to Secure a Smart Contract through Solidity Visibility Modifiers?* Available at: https://www.bluelabellabs.com/blog/secure-smart-contract-solidity-visibility-modifiers/ (Accessed: 5 January 2023).
4. Ahmadisheykhsarmast, S., Senji, S. G. and Sonmez, R. (2023) 'Decentralized tendering of construction projects using Blockchain-based smart contracts and storage systems', Automation in Construction, 151, p. 104900. doi: 10.1016/j.autcon.2023.104900.
5. Ahmed, K. B. and Kumar, D. (2019) 'Blockchain use Cases in Financial Services for Improving Security', in *2019 Third International Conference on Inventive Systems and Control (ICISC)*. IEEE, pp. 220–224. doi: 10.1109/ICISC44355.2019.9036406.
6. Aini, Q. *et al.* (2022) 'Security Level Significance in DApps Blockchain-Based Document Authentication', *Aptisi Transactions on Technopreneurship (ATT)*, 4(3), pp. 292–305. doi: 10.34306/att.v4i3.277.
7. Ajienka, N., Vangorp, P. and Capiluppi, A. (2020) 'An empirical analysis of source code metrics and smart contract resource consumption', *Journal of Software: Evolution and Process*, 32(10). doi: 10.1002/smr.2267.
8. Akbar, N. A. *et al.* (2021) 'Distributed Hybrid Double-Spending Attack Prevention Mechanism for Proof-of-Work and Proof-of-Stake Blockchain Consensuses', *Future Internet*, 13(11), p. 285. doi: 10.3390/fi13110285.
9. Alchemy (2022) 'Learn Solidity: What are events?' Available at: https://www.alchemy.com/overviews/solidity-events (Accessed: 3 January 2023).

10. Alchemy and Werkheiser, B. (2022) *Polygon ZK Rollups: Everything You Need to Know*. Available at: https://www.alchemy.com/overviews/polygon-zk-rollups. (Accessed: 10 February 2023).

11. Alkhalifah, A. *et al.* (2021) 'A Mechanism to Detect and Prevent Ethereum Blockchain Smart Contract Reentrancy Attacks', *Frontiers in Computer Science*, 3. doi: 10.3389/fcomp.2021.598780.

12. Alsunaidi, S. J. and Alhaidari, F. A. (2019) 'A Survey of Consensus Algorithms for Blockchain Technology', in *2019 International Conference on Computer and Information Sciences (ICCIS)*. IEEE, pp. 1–6. doi: 10.1109/ICCISci.2019.8716424.

13. Amiet, N. (2021) 'Blockchain Vulnerabilities in Practice', *Digital Threats: Research and Practice*, 2(2), pp. 1–7. doi: 10.1145/3407230.

14. Annessi, R. and Fast, E. (2021) 'Improving Security for Users of Decentralized Exchanges Through Multiparty Computation', in *2021 IEEE International Conference on Blockchain (Blockchain)*. IEEE, pp. 229–236. doi: 10.1109/Blockchain53845.2021.00038.

15. Antonopoulos, A. and Wood, G. (2018) *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media.

16. Apostolaki, M. *et al.* (2019) 'SABRE: Protecting Bitcoin against Routing Attacks', in *Proceedings 2019 Network and Distributed System Security Symposium*. Reston, VA: Internet Society. doi: 10.14722/ndss.2019.23252.

17. Apriani, M. and Sari, R. F. (2021) 'Performance Comparison of Spongent and Photon Hashing Algorithms in Ethereum-based Blockchain System', in *2021 7th International Conference on Electrical, Electronics and Information Engineering (ICEEIE)*. IEEE, pp. 564–569. doi: 10.1109/ICEEIE52663.2021.9616831.

18. Arthur, V. (2024) 'Polygon vs Ethereum: Which Offers Better Scalability?' Available at:

19. https://coinwire.com/polygon-vs-ethereum/#:~:text=Polygon offers faster transaction speeds, faster than the Ethereum blockchain (Accessed: 10 April 2024).

20. Baldauf, M., Sonnleitner, E. and Kurz, M. (2023) 'Exemplary Ethereum Development Strategies Regarding Security and Gas-Saving', *Electronics*, 13(1), p. 117. doi: 10.3390/electronics13010117.

21. Bartoletti, M.; Carta, S.; Cimoli, T.; Saia, R. Dissecting Ponzi Schemes on Ethereum: Identification, Analysis, and Impact. *Futur. Gener. Comput. Syst.* **2020**, *102*, 259–277. https://doi.org/10.1016/j.future.2019.08.014.

22. Bashir, I. (2020) *Mastering Blockchain*. 3rd edn. Packt Publishing.

23. Begum, A. *et al.* (2020) ‘Blockchain Attacks, Analysis and a Model to Solve Double Spending Attack’, *International Journal of Machine Learning and Computing*, 10, pp. 352–357.

24. Beikverdi, A. and JooSeok Song (2015) ‘Trend of centralization in Bitcoin’s distributed network’, in *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, pp. 1–6. doi: 10.1109/SNPD.2015.7176229.

25. Bhutta, M. N. M. *et al.* (2021) ‘A Survey on Blockchain Technology: Evolution, Architecture and Security’, *IEEE Access*. doi: 10.1109/ACCESS.2021.3072849.

26. Bonneau, J. (2016) ‘Why buy when you can rent? Bribery attacks on Bitcoin-style consensus’, in Security, I. C. on F. C. and D. (ed.). International Conference on Financial Cryptography and Data Security, pp. 19–26. Available at: https://jbonneau.com/doc/B16a-BITCOIN-why_buy_when_you_can_rent.pdf (Accessed: 3 April 202).

27. Bouichou, A., Mezroui, S. and Oualkadi, A. El (2020) ‘An overview of Ethereum and Solidity vulnerabilities’, in *2020 International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*. IEEE, pp. 1–7. doi: 10.1109/ISAECT50560.2020.9523638.

28. Breidenbach, L. *et al.* (2017) *An In-Depth Look at the Parity Multisig Bug*. Available at: https://hackingdistributed.com/2017/07/22/deep-dive-parity-bug/ (Accessed: 11 March 2022).

29. Carlin, D. *et al.* (2018) ‘Detecting Cryptomining Using Dynamic Analysis’, in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, pp. 1–6. doi: 10.1109/PST.2018.8514167.

30. CertiK (2021) *Inari Token*. Available at: https://www.certik.com/projects/inaritoken (Accessed: 10 May 2023).

31. Certik (2021) *The State of DeFi Security 2021*. Available at: https://certik-2.hubspotpagebuilder.com/the-state-of-defi-security-2021 (Accessed: 8 May 2023).

32. Certik (2021) *Vectorspace AI*. Available at: https://skynet.certik.com/projects/vectorspace-ai (Accessed: 10 April 2023).

33. CertiK (2022) 'What is Centralization Risk?' Available at: https://certik.medium.com/what-is-centralization-risk-41cf848f5a74. (Accessed: 10 March 2023).

34. CertiK (2023) *Better security for Blockchains and smart contracts*. Available at: https://www.certik.com/products/formal-verification (Accessed: 12 January 2024).

35. CertiK. 2021. *What is a Timelock?* Available at: https://www.certik.com/resources/blog/Timelock (Accessed: 10 October 2023).

36. Chai Assertation library (2023) 'Chai assertion styles'. Available at: https://www.chaijs.com/guide/styles/ (Accessed: 10 December 2023).

37. Chatterjee, K., Goharshady, A. K. and Pourdamghani, A. (2019) 'Probabilistic Smart Contracts: Secure Randomness on the Blockchain', in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, pp. 403–412. doi: 10.1109/BLOC.2019.8751326.

38. Chaudhry, N. and Yousaf, M. M. (2018) 'Consensus Algorithms in Blockchain: Comparative Analysis, Challenges and Opportunities', in *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*. IEEE, pp. 54–63. doi: 10.1109/ICOSST.2018.8632190.

39. Chen, H. *et al.* (2020) 'A Survey on Ethereum Systems Security', *ACM Computing Surveys*, 53(3), pp. 1–43. doi: 10.1145/3391195.

40. Chi, P.-W., Lu, Y.-H. and Guan, A. (2023) 'A Privacy-Preserving Zero-Knowledge Proof for Blockchain', *IEEE Access*, 11, pp. 85108–85117. doi: 10.1109/ACCESS.2023.3302691.

41. Chittoda, J. (2019) 'Mastering Blockchain Programming with Solidity: Write production-ready smart contracts for Ethereum Blockchain with Solidity', in. Packt Publishing.

42. Choo, K.-K. R., Dehghantanha, A. and Parizi, R. M. (2020) *Blockchain Cybersecurity, Trust and Privacy*. Cham: Springer International Publishing (Advances in Information Security). doi: 10.1007/978-3-030-38181-3.

43. Code4rena (2022) *Badger Citadel contest Findings & Analysis Report*. Available at: https://code4rena.com/reports/2022-02-badger-citadel/#severity-criteria (Accessed: 10 January 2024).

44. Code4rena (2022) *Frax Ether Liquid Staking contest Findings & Analysis Report - Centra*. Available at: https://code4rena.com/reports/2022-09-frax/#m-01-centralization-risk-admin-have-privileges-admin-can-set-address-to-mint-any-amount-of-frxeth-can-set-

any-address-as-validator-and-change-important-state-in-frxethminter-and-withdraw-fund-from-frcethminter- (Accessed: 11 January 2024).

45. Code4rena (2023) *Lybra Finance Findings and Analysis Report*. Available at: https://code4rena.com/reports/2023-06-lybra#overview (Accessed: 11 January 2024).

46. Coinbase (2024) 'Ethereum price'. Available at: https://www.coinbase.com/en-gb/price/ethereum (Accessed: 20 April 2024).

47. Coinbase (2024) Polygon Price. Available at: https://www.coinbase.com/en-gb/price/polygon (Accessed: 20 April 2024).

48. Collins, H. (2019) Creative research: the theory and practice of research for the creative industries. Second edn. London, UK: Bloomsbury Visual Arts, Bloomsbury Publishing Plc.

49. ConsenSys (no date) *Ethereum Smart Contract Best Practices- Known Attacks.* Available at: https://consensys.github.io/smart-contract-best-practices/ (Accessed: 20 June 2023).

50. Conti, M. *et al.* (2018) 'A Survey on Security and Privacy Issues of Bitcoin', *IEEE Communications Surveys & Tutorials*, 20(4), pp. 3416–3452. doi: 10.1109/COMST.2018.2842460.

51. Core, T. *Moving from Admin Key to DAO — Tellor's Parachute Smart Contract*. 2021. https://medium.com/tellor/moving-from-admin-key-to-dao-tellors-parachute-smart-contract-aba4cc9d71fb (Accessed: 21 December 2023).

52. Creswell, J. W. (2007) Qualitative Inquiry& Research Design. Thousand Oaks. CA: Sage.

53. Dai, W. *et al.* (2019) 'Blockchain-Based Smart Contract Access Control System', in *2019 25th Asia-Pacific Conference on Communications (APCC)*. IEEE, pp. 19–23. doi: 10.1109/APCC47188.2019.9026509.

54. Daian, P. *et al.* (2019) 'Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges'. Available at: http://arxiv.org/abs/1904.05234 (Accessed: 19 April 2023).

55. Deng, W., Huang, T. and Wang, H. (2022) 'A Review of the Key Technology in a Blockchain Building Decentralized Trust Platform', *Mathematics*, 11(1), p. 101. doi: 10.3390/math11010101.

56. Destefanis, G. *et al.* (2018) 'Smart contracts vulnerabilities: a call for Blockchain software engineering?', in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, pp. 19–25. doi: 10.1109/IWBOSE.2018.8327567.

57. Destefanis, G. *et al.* (2018) 'Smart contracts vulnerabilities: a call for Blockchain software engineering?', in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, pp. 19–25. doi: 10.1109/IWBOSE.2018.8327567.

58. DevCon, G. (2018) *What are Blockchain Protocols and How Do they Work?* Available at: https://medium.com/@genesishack/draft-what-are-Blockchain-protocols-and-how-do-they-work-94815be5efa7 (Accessed: 12 September 2022).

59. Di Angelo, M. and Slazer, G. (2020) 'Wallet Contracts on Ethereum', in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, pp. 1–2. doi: 10.1109/ICBC48266.2020.9169467.

60. EatTheBlocks (2019) 'Access Control with Solidity & OpenZeppelin | Authorization'. Available at: https://www.google.com/search?q=access+control+issue+by+using+solidity&rlz=1C1GCEA_enGB971GB974&ei=jioHZJ30KYTGgQbu2wM&oq=access+control+issue+by+using&gs_lcp=Cgxnd3Mtd2l6LXNlcnAQAxgAMgUIIRCgATIFCCEQoAEyCAghEBYQHhAdMggIIRAWEB4QHTIICCEQFhAeEB0yCAghEBYQHhAd (Accessed: 20 May 2023).

61. Edgcombe., J. (2016) *So, you want to connect your IoT device to the Blockchain?* Available at: https://www.cambridgeconsultants.com/insights/so-you-want-to-connect-your-iot-device-to-the-b)lockchain (Accessed: 2 July 2022).

62. Elliott, S. (2022) *Has Proof of Stake Made Ethereum More Centralized?* Available at: https://decrypt.co/111485/has-proof-of-stake-made-ethereum-more-centralized (Accessed: 20 April 2023).

63. Eskandari, S., Moosavi, S. and Clark, J. (2019) 'SoK: Transparent Dishonesty: front-running attacks on Blockchain'. Available at: http://arxiv.org/abs/1902.05164 (Accessed: 2 June 2022).

64. Essaid, M. *et al.* (2018) 'Network Usage of Bitcoin Full Node', in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, pp. 1286–1291. doi: 10.1109/ICTC.2018.8539723.

65. Ethereum (2023) 'Proof-of-stake (PoS)'. Available at: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/ (Accessed: 20 December 2023).

66. Ethereum.org Team (2023) *Visibility and Getters*. Available at: https://docs.soliditylang.org/en/latest/contracts.html (Accessed: 1 December 2023).

67. Ethers (2023) ‘What is Ethers?’ Available at: https://docs.ethers.org/v5/ (Accessed: 1 December 2023).

68. Etherscan (2021) *Inari Token Smart Contract*. Available at: https://etherscan.io/address/0xca75c43f8c9afd356c585ce7aa4490b48a95c466#code (Accessed: 15 December 2023).

69. Etherscan (2024) ‘Etherscan gas tracker’. Available at: https://etherscan.io/gastracker (Accessed: 15 April 2024).

70. Fang, Y. *et al.* (2021) ‘Jyane: Detecting Reentrancy vulnerabilities based on path profiling method’, in *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, pp. 274–282. doi: 10.1109/ICPADS53394.2021.00040.

71. Feng, Y., Torlak, E. and Bodik, R. (2019) ‘Precise Attack Synthesis for Smart Contracts’. Available at: http://arxiv.org/abs/1902.06067 (Accessed: 18 May 2022).

72. Future Learn (2021) *4 of the top DeFi cybersecurity risks*. Available at: https://www.futurelearn.com/info/courses/defi-exploring-decentralised-finance-with-Blockchain-technologies/0/steps/256219 (Accessed: 2 March 2023).

73. Gao, J. *et al.* (2019) ‘EASYFLOW: Keep Ethereum Away from Overflow’, in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, pp. 23–26. doi: 10.1109/ICSE-Companion.2019.00029.

74. Gartner (2023) *Blockchain platforms reviews and ratings*. Available at: https://www.gartner.com/reviews/market/blockchain-platforms (Accessed: 6 January 2024).

75. Ghaffari, F. *et al.* (2020) ‘Authentication and Access Control based on Distributed Ledger Technology: A survey’, in *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, pp. 79–86. doi: 10.1109/BRAINS49436.2020.9223297.

76. Ghaffari, F. *et al.* (2021) ‘A Novel Access Control Method Via Smart Contracts for Internet-Based Service Provisioning’, *IEEE Access*, 9, pp. 81253–81273. doi: 10.1109/ACCESS.2021.3085831.

77. Ghaleb, A., Rubin, J. and Pattabiraman, K. (2022) ‘eTainter: detecting gas-related vulnerabilities in smart contracts’, in *Proceedings of the 31st ACM SIGSOFT International*

Symposium on Software Testing and Analysis. New York, NY, USA: ACM, pp. 728–739. doi: 10.1145/3533767.3534378.

78. Ghaleb, A., Rubin, J. and Pattabiraman, K. (2023) 'AChecker: Statically Detecting Smart Contract Access Control Vulnerabilities', in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, pp. 945–956. doi: 10.1109/ICSE48619.2023.00087.

79. Github (2022) 'Centralization risk: admin have privileges'. Available at: https://github.com/code-423n4/2022-09-frax-findings/issues/107 (Accessed: 20 March 2023).

80. Github (no date) *Ethereum/devp2p: Ethereum peer-to-peer networking*. Available at: https://github.com/ethereum/devp2p (Accessed: 22 March 2023).

81. Goldberg, O. (2018) *How to Not Destroy Millions in Smart Contracts*. Available at: https://hackernoon.com/how-to-not-destroy-millions-in-smart-contracts-pt-2-85c4d8edd0cf (Accessed: 18 May 2022).

82. Grech, N. *et al.* (2018) 'MadMax: surviving out-of-gas conditions in Ethereum smart contracts', *Proceedings of the ACM on Programming Languages*, 2(OOPSLA), pp. 1–27. doi: 10.1145/3276486.

83. Habic, B. (2024) Public testnets are a threat to the development of dapps. Available at: https://blockworks.co/news/public-testnets-threat-development-dapps (Accessed: 15 March 2024).

84. Hafid, A., Hafid, A. S. and Samih, M. (2020) 'Scaling Blockchains: A Comprehensive Survey', *IEEE Access*, 8, pp. 125244–125262. doi: 10.1109/ACCESS.2020.3007251.

85. Han, J. *et al.* (2021) 'An efficient multi-signature wallet in Blockchain using bloom filter', in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, pp. 273–281. doi: 10.1145/3412841.3441910.

86. Han, R. *et al.* (2023) 'How Can Incentive Mechanisms and Blockchain Benefit with Each Other? A Survey', *ACM Computing Surveys*, 55(7), pp. 1–38. doi: 10.1145/3539604.

87. Hardhat (2023) 'Hardhat, Ethereum development environment for professionals'. Available at: https://hardhat.org/tutorial/setting-up-the-environment (Accessed: 11 January 2024).

88. He, D. *et al.* (2020) 'Smart Contract Vulnerability Analysis and Security Audit', *IEEE Network*, 34(5), pp. 276–282. doi: 10.1109/MNET.001.1900656.

89. Hoang Minh. (2022) *Building a Front-end Decentralized Application with ReactJS*. Available at: https://techfi.tech/building-a-front-end-decentralized-application-with-reactjs/ (Accessed: 20 March 2023).

90. Homoliak, I. *et al.* (2021) 'The Security Reference Architecture for Blockchains: Toward a Standardized Model for Studying Vulnerabilities, Threats, and Defenses', *IEEE Communications Surveys and Tutorials*. doi: 10.1109/COMST.2020.3033665.

91. Hooper, D., Solorio, K. and Kanna, R. (2019) *Hands-On Smart Contract Development with Solidity and Ethereum: From Fundamentals to Deployment Paperback*. O'Reilly Media.

92. Hou, C. *et al.* (2019) 'SquirRL: Automating Attack Analysis on Blockchain Incentive Mechanisms with Deep Reinforcement Learning'. Available at: http://arxiv.org/abs/1912.01798 (Accessed: 20 March 2022).

93. Hsieh, Y.-Y.; (JP) Vergne, J.-P.; Wang, S. The Internal and External Governance of Blockchain-Based Organizations. In *Bitcoin and Beyond*; Routledge: London, 2017; pp 48–68. https://doi.org/10.4324/9781315211909-3.

94. Salomon, H. (2023) *Ethereum Block (PoS)*. Available at: https://inevitableeth.com/home/ethereum/blockchain/block (Accessed: 18 December 2023).

95. Hu, B. *et al.* (2019) 'A Collaborative Intrusion Detection Approach Using Blockchain for Multimicrogrid Systems', *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(8), pp. 1720–1730. doi: 10.1109/TSMC.2019.2911548.

96. Hu, Y. *et al.* (2021) 'Security Threats from Bitcoin Wallet Smartphone Applications', in *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*. New York, NY, USA: ACM, pp. 89–100. doi: 10.1145/3422337.3447832.

97. Huang, J. *et al.* (2019) 'Survey on Blockchain Incentive Mechanism', in *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*. IEEE, pp. 386–395. doi: 10.1007/978-981-15-0118-0_30.

98. Huang, J. *et al.* (2021) 'Blockchain Network Propagation Mechanism Based on P4P Architecture', *Security and Communication Networks*, 2021. doi: 10.1155/2021/8363131.

99. Huang, Y. *et al.* (2019) 'Smart Contract Security: A Software Lifecycle Perspective', *IEEE Access*, 7, pp. 150184–150202. doi: 10.1109/ACCESS.2019.2946988.

100. Huang, Y. *et al.* (2022) ‘Deep Smart Contract Intent Detection’. Available at: http://arxiv.org/abs/2211.10724.

101. Inari (2021) *Inari Contract Source Code*. Available at: https://etherscan.io/address/0xca75c43f8c9afd356c585ce7aa4490b48a95c466#code (Accessed: 18 January 2024).

102. Infura (2023) ‘Ethereum JavaScript Libraries: web3.js vs. ethers.js’. Available at: https://blog.infura.io/post/ethereum-javascript-libraries-web3-js-vs-ethers-js-part-i (Accessed: 15 February 2024).

103. Ivanov, N. and Yan, Q. (2022) ‘Decentralization Paradox: A Study of Hegemonic and Risky ERC-20 Tokens’. Available at: http://arxiv.org/abs/2209.08370.

104. Javaid, M. *et al.* (2021) ‘Blockchain technology applications for Industry 4.0: A literature-based review’, *Blockchain: Research and Applications*, p. 100027. doi: 10.1016/j.bcra.2021.100027.

105. Jha, P. (2022) *Ethereum at the center of centralization debate as SEC lays claim*. Available at: https://cointelegraph.com/news/ethereum-at-the-center-of-centralization-debate-as-sec-lays-claim (Accessed: 20 January 2024).

106. Jiang, B., Liu, Y. and Chan, W. K. (2018) ‘ContractFuzzer: fuzzing smart contracts for vulnerability detection’, in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. New York, NY, USA: ACM, pp. 259–269. doi: 10.1145/3238147.3238177.

107. Jin, L. *et al.* (2021) ‘DNSonChain: Delegating Privacy-Preserved DNS Resolution to Blockchain’, in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, pp. 1–11. doi: 10.1109/ICNP52444.2021.9651951.

108. Kearney, J. J. and Perez-Delgado, C. A. (2021) ‘Vulnerability of Blockchain technologies to quantum attacks’, *Array*. doi: 10.1016/j.array.2021.100065.

109. Kenneth, H. (2019) *Web3.js Ethereum Javascript API*. Available at: https://medium.com/coinmonks/web3-js-ethereum-javascript-api-72f7b22e2f0a (Accessed: 18 December 2023).

110. Khalifa, A. M., Bahaa-Eldin, A. M. and Sobh, M. A. (2019) ‘Quantum attacks and defenses for proof-of-stake’, in *Proceedings - ICCES 2019: 2019 14th International Conference on Computer Engineering and Systems*. doi: 10.1109/ICCES48960.2019.9068181.

111. Khan, K. M., Arshad, J. and Khan, M. M. (2020) ‘Simulation of transaction malleability attack for Blockchain-based e-Voting’, *Computers & Electrical Engineering*, 83, p. 106583. doi: 10.1016/j.compeleceng.2020.106583.

112. Khan, Z. A. and Siami Namin, A. (2020) *Ethereum Smart Contracts: Vulnerabilities and their Classifications*, *2020 IEEE International Conference on Big Data (Big Data)*. IEEE. doi: 10.1109/BigData50022.2020.9439088.

113. Kim, S. K.; Mason, J.; Ma, Z.; Miller, A.; Murali, S.; Bailey, M. Measuring Ethereum Network Peers. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*; Association for Computing Machinery, 2018; pp 91–104. https://doi.org/10.1145/3278532.3278542.

114. Kitakami, M. and Matsuoka, K. (2018) ‘An Attack-Tolerant Agreement Algorithm for Block Chain’, in *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, pp. 227–228. doi: 10.1109/PRDC.2018.00041.

115. Kong, Q.-P. *et al.* (2022) ‘Characterizing and Detecting Gas-Inefficient Patterns in Smart Contracts’, *Journal of Computer Science and Technology*, 37(1), pp. 67–82. doi: 10.1007/s11390-021-1674-4.

116. Kumar, V. (2021) ‘Finally a better react.js folder structure’. Available at: https://medium.com/@kumarvinoth/finally-a-better-react-js-folder-structure-821a2210835 (Accessed: 22 December 2023).

117. Kuryłowicz, P. 2023. *The Role of Access Control in Solidity Smart Contracts*. Available at: https://composable-security.com/blog/the-role-of-access-control-in-solidity-smart-contracts/ (Accessed: 25 January 2024).

118. Kushwaha, S. S. *et al.* (2022) ‘Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract’, *IEEE Access*, 10, pp. 6605–6621. doi: 10.1109/ACCESS.2021.3140091.

119. Kuznetsov, O. *et al.* (2024) ‘Merkle Trees in Blockchain: A Study of Collision Probability and Security Implications’. doi: https://doi.org/10.48550/arXiv.2402.04367.

120. Larson, S. (2022) *Creating an Ownable Smart Contract in Solidity for Ethereum*. Available at: https://grizzlypeaksoftware.com/articles?id=6NvaOcWdWhwGEKbILwVUKr (Accessed: 12 January 2024).

121. Le, D.-P., Yang, G. and Ghorbani, A. (2019) ‘A New Multisignature Scheme with Public Key Aggregation for Blockchain’, in *2019 17th International Conference on Privacy, Security and Trust (PST)*. IEEE, pp. 1–7. doi: 10.1109/PST47121.2019.8949046.

122. Leonardos, N., Leonardos, S. and Piliouras, G. (2019) ‘Oceanic Games: Centralization Risks and Incentives in Blockchain Mining’. doi: 10.1007/978-3-030-37110-4_13.

123. Li, C. (2021) ‘Gas Estimation and Optimization for Smart Contracts on Ethereum’, in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, pp. 1082–1086. doi: 10.1109/ASE51524.2021.9678932.

124. Li, X., Ma, Z. and Luo, S. (2022) ‘Blockchain-Oriented Privacy Protection with Online and Offline Verification in Cross-Chain System’, in *2022 International Conference on Blockchain Technology and Information Security (ICBCTIS)*. IEEE, pp. 177–181. doi: 10.1109/ICBCTIS55569.2022.00048.

125. Li, Z. *et al.* (2021) ‘B-DNS: A Secure and Efficient DNS Based on the Blockchain Technology’, *IEEE Transactions on Network Science and Engineering*, 8(2), pp. 1674–1686. doi: 10.1109/TNSE.2021.3068788.

126. Liang, Y.-C. (2020) ‘Blockchain for Dynamic Spectrum Management’, in, pp. 121–146. doi: 10.1007/978-981-15-0776-2_5.

127. Liao, K. and Katz, J. (2017) ‘Incentivizing double-spend collusion in bitcoin’. Available at: https://www.semanticscholar.org/paper/Incentivizing-Double-Spend-Collusion-in-Bitcoin-Liao-Katz/81eeb6553790e81c183c7798672bd82df957ff68 (Accessed: 10 June 2022).

128. Lin, I.-C. and Liao, T.-C. (2017) ‘A Survey of Blockchain Security Issues and Challenges’, *International Journal of Network Security*, 19, pp. 653–659. doi: 10.6633/IJNS.201709.19(5).01.

129. Liu, Y. *et al.* (2019) ‘A Comparative Study of Blockchain-Based DNS Design’, in *Proceedings of the 2019 2nd International Conference on Blockchain Technology and Applications*. New York, NY, USA: ACM, pp. 86–92. doi: 10.1145/3376044.3376057.

130. Lone, A. H. and Naaz, R. (2020) ‘Demystifying Cryptography behind Blockchains and a Vision for Post-Quantum Blockchains’, in *2020 IEEE International Conference for Innovation in Technology (INOCON)*. IEEE, pp. 1–6. doi: 10.1109/INOCON50539.2020.9298215.

131. Ma, R. *et al.* (2019) *Fundamentals of Smart Contract Security Kindle Edition*. Kindle Edition.

132. Marchesi, L. *et al.* (2020) 'Design Patterns for Gas Optimization in Ethereum', in *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, pp. 9–15. doi: 10.1109/IWBOSE50093.2020.9050163.

133. Maruf O Abubakar (2022) *Rug-Pull: How to Spot the Fraud Easily via Smart Contract Codes*. Available at: https://coinsbench.com/rug-pull-how-to-spot-the-fraud-easily-via-smart-contract-codes-39e69160d009 (Accessed: 18 June 2023).

134. Mazorra, B., Adan, V. and Daza, V. (2022) 'Do Not Rug on Me: Leveraging Machine Learning Techniques for Automated Scam Detection', *Mathematics*, 10(6), p. 949. doi: 10.3390/math10060949.

135. Mazorra, B., Adan, V. and Daza, V. (2022) 'Do not rug on me: Zero-dimensional Scam Detection'. doi: doi.org/10.48550/arXiv.2201.07220.

136. Mendes, C. (2023) JavaScript and Web3: Building decentralized applications with the most popular language. Available at: https://blog.bepro.network/javascript-and-web3-build-dapps (Accessed: 10 January 2024).

137. Mense, A. and Flatscher, M. (2018) 'Security Vulnerabilities in Ethereum Smart Contracts', in *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services*. New York, NY, USA: ACM, pp. 375–380. doi: 10.1145/3282373.3282419.

138. MetaOpenSource (2023) *React*. Available at: https://react.dev/learn/describing-the-ui (Accessed: 18 January 2024).

139. Min, T. (2023) *Ethereum Structure*. Available at: https://medium.com/@feelwjd/ethereum-structure-836215125df8 (Accessed: 14 December 2024).

140. Minima (2022) *Is Bitcoin incentivizing its own centralization?* Available at: https://www.minima.global/post/is-bitcoin-incentivizing-its-own-centralization (Accessed: 18 November 2023).

141. Mirkin, M. *et al.* (2020) 'BDoS: Blockchain Denial-of-Service', in *Proceedings of the ACM Conference on Computer and Communications Security*. doi: 10.1145/3372297.3417247.

142. Modiri, N. The ISO Reference Model Entities. *IEEE Netw.* 1991, *5* (4), 24–33. https://doi.org/10.1109/65.93182.

143. Mollajafari, S. (2022) *Warning on Blockchain security risk*. Available at: https://www.bcs.org/articles-opinion-and-research/warning-on-Blockchain-security-risk/ Accessed: 10 October 2023).

144. Mollajafari, S. and Bechkoum, K. (2023) 'Blockchain Technology and Related Security Risks: Towards a Seven-Layer Perspective and Taxonomy', *Sustainability*, 15(18), p. 13401. doi: 10.3390/su151813401.

145. Monteiro, A. P. C. A Study of Static Analysis Tools for Ethereum Smart Contracts. **2019**.

146. Mosakheil, J. H. (2018) *Security Threats Classification in Blockchains*. Available at: https://www.semanticscholar.org/paper/Security-Threats-Classification-in-Blockchains-Mosakheil/91bbbb31101cbc2e803726d7210b4100f7b09ac5. (Accessed: 3 January 2022).

147. Mou, T., Coblenz, M. and Aldrich, J. (2021) 'An Empirical Study of Protocols in Smart Contracts'. doi: https://doi.org/10.48550/arXiv.2110.08983.

148. Najafi, S. (2020) *Front-Running Attacks on Blockchain*. Available at: https://medium.com/codechain/front-running-attacks-on-Blockchain-1f5ba28cd42b (Accessed: 1 July 2022).

149. Nakamura, Y. *et al.* (2019) 'Capability-Based Access Control for the Internet of Things: An Ethereum Blockchain-Based Scheme', in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, pp. 1–6. doi: 10.1109/GLOBECOM38437.2019.9013321.

150. Nathan Sexer (2018) *Decentralized Exchanges vs. Centralized Exchanges: Overview*. Available at: https://consensys.net/blog/news/decentralized-exchanges-overview-benefits-and-advantages-over-centralized-exchanges/ (Accessed: 10 January 2022).

151. Neiheiser, R. *et al.* (2023) 'Practical Limitations of Ethereum's Layer-2', *IEEE Access*, 11, pp. 8651–8662. doi: 10.1109/ACCESS.2023.3237897.

152. Nguyen, G.-T. and Kim, K. (2018) 'A Survey about Consensus Algorithms Used in Blockchain', *Journal of Information Processing Systems*, 14. doi: 10.3745/JIPS.01.0024.

153. Nguyen, T. (2023) *Top 10 Popular Node JS Blockchain Frameworks for Building dApps*. Available at: https://www.front-endmag.com/insights/node-js-Blockchain-frameworks/ (Accessed: 23 January 2024).

154. Nico (2024) *Gas and fees*. Available at: https://ethereum.org/en/developers/docs/gas/ (Accessed: 15 April 2024).

155. Oates, B. J. (2006) Researching information systems and computing. London: SAGE Publications.

156. Onyeije, P. (2023) 'React's Layout Component's Concept'. Available at: https://blog.openreplay.com/reacts-layout-components-concept/ (Accessed: 18 February 2024).

157. OpenZeppelin (2022) *Access Control*. Available at: https://docs.openzeppelin.com/contracts/4.x/access-control (Accessed: 18 October 2023).

158. OpenZeppelin (2023) *Access Control Contracts*. Available at: https://github.com/OpenZeppelin/openzeppelincontracts/tree/master/contracts/access (Accessed: 18 January 2024).

159. OpenZeppelin (2023) *The standard for secure Blockchain applications*. Available at: https://www.openzeppelin.com/ (Accessed: 18 January 2024).

160. Pagan, M. (2021) 'Getting Started with React and TypeScript'. Available at: https://www.thisdot.co/blog/getting-started-with-react-and-typescript/ (Accessed: 10 December 2023).

161. PARTZ, H. (2021) *Bilaxy exchange suspends website after ERC-20 hot wallet hack*. Available at: https://cointelegraph.com/news/bilaxy-exchange-suspends-website-after-erc-20-hot-wallet-hack (Accessed: 18 May 2022).

162. Patel, S. (2015) The research paradigm – methodology, epistemology and ontology – explained in simple language. Available at: https://salmapatel.co.uk/academia/the-research-paradigm-methodology-epistemology-and-ontology-explained-in-simple-language/ (Accessed: 14 January 2022).

163. Pierro, G. A. and Tonelli, R. (2021) 'Analysis of Source Code Duplication in Ethereum Smart Contracts', in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, pp. 701–707. doi: 10.1109/SANER50967.2021.00089.

164. Polygon (2024) 'Polygon gas tracker'. Available at: https://polygonscan.com/gastracker#gasguzzler (Accessed: 18 April 2024).

165. Polygon (2024) 'Polygon PoS Chain Average Block Time Chart'. Available at: https://polygonscan.com/chart/blocktime (Accessed: 18 April 2024).

166. Polygon (2024) ‘Polygon PoS Chain Average Gas Limit Chart’. Available at: https://polygonscan.com/chart/gaslimit (Accessed: 18 April 2024).

167. Praitheeshan, P. *et al.* (2019) ‘Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey’, pp. 1–21. Available at: http://arxiv.org/abs/1908.08605.

168. Qin, K. *et al.* (2020) ‘Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit’. Available at: http://arxiv.org/abs/2003.03810.

169. QuickNode (2023) *How to Build a React Front-end with wagmi*. Available at: https://www.quicknode.com/guides/ethereum-development/dapps/building-dapps-with-wagmi#:~:text=wagmi is an open-source,using React%2C Viem and Bootstrap.

170. Raikwar, M., Gligoroski, D. and Kralevska, K. (2019) ‘SoK of Used Cryptography in Blockchain’, *IEEE Access*, 7, pp. 148550–148575. doi: 10.1109/ACCESS.2019.2946983.

171. Ramanan, P., Li, D. and Gebraeel, N. (2022) ‘Blockchain-Based Decentralized Replay Attack Detection for Large-Scale Power Systems’, *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(8), pp. 4727–4739. doi: 10.1109/TSMC.2021.3104087.

172. React (2023) ‘Rules of Hooks’. Available at: https://legacy.reactjs.org/docs/hooks-rules.html (Accessed: 10 February 2024).

173. Ren, S. *et al.* (2019) ‘BlockDNS: Enhancing Domain Name Ownership and Data Authenticity with Blockchain’, in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, pp. 1–6. doi: 10.1109/GLOBECOM38437.2019.9013817.

174. Rezaeighaleh, H. and Zou, C. C. (2019) ‘New Secure Approach to Backup Cryptocurrency Wallets’, in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, pp. 1–6. doi: 10.1109/GLOBECOM38437.2019.9014007.

175. Rouhani, S. and Deters, R. (2019) ‘Blockchain based access control systems: State of the art and challenges’, in *IEEE/WIC/ACM International Conference on Web Intelligence*. New York, NY, USA: ACM, pp. 423–428. doi: 10.1145/3350546.3352561.

176. Saad, M. *et al.* (2019) ‘Countering Selfish Mining in Blockchains’, in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, pp. 360–364. doi: 10.1109/ICCNC.2019.8685577.

177. Saad, M. *et al.* (2022) ‘RouteChain: Towards Blockchain-based secure and efficient BGP routing’, *Computer Networks*, 217, p. 109362. doi: 10.1016/j.comnet.2022.109362.

178. Saad, M., Khormali, A. and Mohaisen, A. (2019) ‘Dine and Dash: Static, Dynamic, and Economic Analysis of In-Browser Cryptojacking’, in *2019 APWG Symposium on*

Electronic Crime Research (eCrime). IEEE, pp. 1–12. doi: 10.1109/eCrime47957.2019.9037576.https://doi.org/10.1109/eCrime47957.2019.9037576.

179. Saad, M., Thai, M. T. and Mohaisen, A. (2018) 'POSTER', in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. New York, NY, USA: ACM, pp. 809–811. doi: 10.1145/3196494.3201584.

180. Sai, A. R. *et al.* (2021) 'Taxonomy of centralization in public Blockchain systems: A systematic literature review', *Information Processing & Management*, 58(4), p. 102584. doi: 10.1016/j.ipm.2021.102584.

181. Saini, K., Sharma, S. and Sarkar, U. (2022) 'Blockchain and Cryptography', in *2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*. IEEE, pp. 1863–1868. doi: 10.1109/ICAC3N56670.2022.10074345.

182. Salah, K.; Rehman, M. H. U.; Nizamuddin, N.; Al-Fuqaha, A. Blockchain for AI: Review and Open Research Challenges. *IEEE Access* **2019**, *7*, 10127–10149. https://doi.org/10.1109/ACCESS.2018.2890507.

183. Saleh, L. (2022) *Cold Wallets, Hot Wallets: The Basics of Storing Your Crypto Securely*. Available at: https://www.mcafee.com/blogs/internet-security/cold-wallets-hot-wallets-the-basics-of-storing-your-crypto-securely/ (Accessed: 11 July 2023).

184. Salomon, H. (2023) *Ethereum Block (PoS)*. Available at: https://inevitableeth.com/home/ethereum/blockchain/block (Accessed: 24 December 2023).

185. Samreen, N. F. and Alalfi, M. H. (2021) 'SmartScan: An approach to detect Denial of Service Vulnerability in Ethereum Smart Contracts', in *2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, pp. 17–26. doi: 10.1109/WETSEB52558.2021.00010.

186. Santana, C. and Albareda, L. (2022) 'Blockchain and the emergence of Decentralized Autonomous Organizations (DAOs): An integrative model and research agenda', *Technological Forecasting and Social Change*, 182, p. 121806. doi: 10.1016/j.techfore.2022.121806.

187. Santana, C.; Albareda, L. Blockchain and the Emergence of Decentralized Autonomous Organizations (DAOs): An Integrative Model and Research Agenda. *Technol. Forecast. Soc. Change* **2022**, *182*, 121806. https://doi.org/10.1016/j.techfore.2022.121806.

188. Saunders, M. N. K., Lewis, P. and Thornhill, A. (2020) 'Research Methods for Business Students, 8th edition', in. Pearson.

189. Sayeed, S., Marco-Gisbert, H. and Caira, T. (2020) 'Smart Contract: Attacks and Protections', *IEEE Access*, 8, pp. 24416–24427. doi: 10.1109/ACCESS.2020.2970495.

190. Sguanci, C., Spatafora, R. and Vergani, A. M. (2021) 'Layer 2 Blockchain Scaling: a Survey'. Available at: http://arxiv.org/abs/2107.10881 (Accessed: 10 March 2024).

191. Shahda, W. (2019) *Protect Your Solidity Smart Contracts from Re-entrancy Attacks*. Available at: https://medium.com/coinmonks/protect-your-solidity-smart-contracts-from-reentrancy-attacks-9972c3af7c21 (Accessed: 11 July 2022).

192. Shanzson (2022) *Smart Contract Auditor Tools and Techniques*. Available at: https://github.com/shanzson/Smart-Contract-Auditor-Tools-and-Techniques (Accessed: 12 July 2023).

193. Shevko, M., Yanovich, Y. and Zhukova, D. (2023) 'Demo: Decentralized Autonomous Organization with Centralized Crisis Resolution', in *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, pp. 1005–1008. doi: 10.1109/ICDCS57875.2023.00112.

194. Siddiqi, A. ud din and Ali, Z. (2022) 'The Sybil Attack Prevention Algorithm', *International Journal of Advance Sciences and Computing*, 1. Available at: http://ijasc.com/index.php/ijasc/index (Accessed: 8 March 2023).

195. Sigurdsson, G., Giaretta, A. and Dragoni, N. (2020) 'Vulnerabilities and Security Breaches in Cryptocurrencies', in, pp. 288–299. doi: 10.1007/978-3-030-14687-0_26.

196. Singh, M. and Kim, S. (2019) 'Blockchain technology for decentralized autonomous organizations', in, pp. 115–140. doi: 10.1016/bs.adcom.2019.06.001.

197. Sm4rty (2022) *Smart Contract Audit Methodology & Tips*. Available at: https://sm4rty.medium.com/smart-contract-audit-methodology-tips-6e529a3f3435 (Accessed: 11 May 2023).

198. Solidgroup (2021) 'New Security Vulnerability: How owners can mint tokens AFTER renouncing ownership'. Available at: https://solidgroup-68170.medium.com/new-

security-vulnerability-how-owners-can-mint-tokens-after-renouncing-ownership-f3f2af13b5f6 (Accessed: 25 July 2023).

199. Solidity Team (2023) *Contract ABI Specification*. Available at: https://docs.soliditylang.org/en/v0.8.11/abi-spec.html (Accessed: 15 December 2023).

200. Sun, H., Ruan, N. and Su, C. (2020) 'How to Model the Bribery Attack: A Practical Quantification Method in Blockchain', in, pp. 569–589. doi: 10.1007/978-3-030-59013-0_28.

201. Sung, S. (2021) 'A new key protocol design for cryptocurrency wallet', *ICT Express*, 7(3), pp. 316–321. doi: 10.1016/j.icte.2021.08.002.

202. Suresh, A. *et al.* (2020) 'A Hybrid Proof based Consensus Algorithm for Permission less Blockchain', in *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE, pp. 707–713. doi: 10.1109/ICIRCA48905.2020.9183109.

203. Sward, A., Vecna, I. and Stonedahl, F. (2018) 'Data Insertion in Bitcoin's Blockchain', *Ledger*, 3. doi: 10.5195/ledger.2018.101.

204. Swathi, P., Modi, C. and Patel, D. (2019) 'Preventing Sybil Attack in Blockchain using Distributed Behavior Monitoring of Miners', in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, pp. 1–6. doi: 10.1109/ICCCNT45670.2019.8944507.

205. Swayne, M. (2024) *Vitalik Buterin Proposes Hard-Fork Strategy to Protect Ethereum From Quantum Attacks*. Available at: https://thequantuminsider.com/2024/03/11/vitalik-buterin-proposes-hard-fork-strategy-to-protect-ethereum-from-quantum-attacks/ (Accessed: 12 March 2024).

206. SWC (2020) *Smart Contract Weakness Classification and Test Cases*. Available at: https://swcregistry.io/ (Accessed: 11 June 2022).

207. Swcregistry (2020) *Hash Collisions With Multiple Variable Length Arguments*. Available at: https://swcregistry.io/docs/SWC-133.

208. Swcregistry (2020) *Weak Sources of Randomness from Chain Attributes*. Available at: https://swcregistry.io/docs/SWC-120 (Accessed: 11 June 2023).

209. Tailwindcss (2023) *Get started with Tailwind CSS*. Available at: https://tailwindcss.com/docs/installation (Accessed: 8 January 2024).

210. Tanana, D. (2020) ‘Behavior-Based Detection of Cryptojacking Malware’, in *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*. IEEE, pp. 0543–0545. doi: 10.1109/USBEREIT48449.2020.9117732.

211. Tapsell, J.; Naeem Akram, R.; Markantonakis, K. An Evaluation of the Security of the Bitcoin Peer-To-Peer Network. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*; IEEE, 2018; pp 1057–1062. https://doi.org/10.1109/Cybermatics_2018.2018.00195.

212. Taylor, P. J.; Dargahi, T.; Dehghantanha, A.; Parizi, R. M.; Choo, K.-K. R. A Systematic Literature Review of Blockchain Cyber Security. *Digit. Commun. Networks* **2020**, *6* (2), 147–156. https://doi.org/10.1016/j.dcan.2019.01.005.

213. Tenkhoff, C. (2023) *That’ll be the DAO: an overview of the structure and status of decentralised autonomous organisations under English law*. Available at: https://www.taylorwessing.com/en/insights-and-events/insights/2023/04/that-will-be-the-dao (Accessed: 18 February 2024).

214. Thomas, D. (2021) *AscendEX Hacked, $77.7M Lost From Hot Wallets*. Available at: https://beincrypto.com/ascendex-hacked-77-7m-lost-from-hot-wallets/.

215. Thurman, A. (2021) *Cream Finance Exploited in Flash Loan Attack Netting Over $100M*. Available at: https://www.coindesk.com/business/2021/10/27/cream-finance-exploited-in-flash-loan-attack-worth-over-100m (Accessed: 15 July 2022).

216. Tikhomirov, S. *et al.* (2018) ‘SmartCheck:Static analysis of ethereum smart contracts’, in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. New York, NY, USA: ACM, pp. 9–16. doi: 10.1145/3194113.3194115.

217. Traore, P. (2023) *Timelock: Achieving Security for Decentralized Projects*. Available at: https://blog.hashex.org/timelock-achieving-security-for-decentralized-projects-62216b1f6f7d (Accessed: 10 February 2024).

218. Ubaidullah Rajput, Fizza Abbas, Heekuck. O. (2018) ‘A Solution towards Eliminating Transaction Malleability in Bitcoin’, *Journal of information processing system*, 14. doi: doi.org/10.3745/JIPS.03.0101.

219. Ukey, U. (2022) *Why you should use TailwindCSS in your ReactJS project*. Available at: https://dev.to/kunalukey/why-you-should-use-tailwindcss-in-your-reactjs-project-51kf#:~:text=In conclusion%2C TailwindCSS is a,reduce your CSS file size.

220. Ul Hassan, M., Rehmani, M. H. and Chen, J. (2020) 'Differential privacy in Blockchain technology: A futuristic approach', *Journal of Parallel and Distributed Computing*, 145, pp. 50–74. doi: 10.1016/j.jpdc.2020.06.003.

221. Usman, T. A.; Selcuk, A. A.; Ozarslan, S. An Analysis of Ethereum Smart Contract Vulnerabilities. In *2021 International Conference on Information Security and Cryptology (ISCTURKEY)*; IEEE, 2021; pp 99–104. https://doi.org/10.1109/ISCTURKEY53027.2021.9654305.

222. Varun, M., Palanisamy, B. and Sural, S. (2022) 'Mitigating Frontrunning Attacks in Ethereum', in *Proceedings of the Fourth ACM International Symposium on Blockchain and Secure Critical Infrastructure*. New York, NY, USA: ACM, pp. 115–124. doi: 10.1145/3494106.3528682.

223. Vivar, A. L. *et al.* (2020) 'An analysis of smart contracts security threats alongside existing solutions', *Entropy*. doi: 10.3390/e22020203.

224. Wagmi (2022) *Wagmi React Hooks for Ethereum*. Available at: https://wagmi.sh/ (Accessed: 19 December 2023).

225. Wang, A. *et al.* (2020) 'Artemis: An Improved Smart Contract Verification Tool for Vulnerability Detection', in *2020 7th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, pp. 173–181. doi: 10.1109/DSA51864.2020.00031.

226. Wang, B. *et al.* (2021) 'BLOCKEYE: Hunting for DeFi Attacks on Blockchain', in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, pp. 17–20. doi: 10.1109/ICSE-Companion52605.2021.00025.

227. Wang, G.; Li, J.; Wang, X.; Li, J.; Yuan, Y.; Wang, F.-Y. Blockchain-Based Crypto Management for Reliable Real-Time Decision-Making. *IEEE Trans. Comput. Soc. Syst.* **2022**, 1–10. https://doi.org/10.1109/TCSS.2022.3211331.

228. Wang, J. *et al.* (2018) 'A Blockchain Based Privacy-Preserving Incentive Mechanism in Crowdsensing Applications', *IEEE Access*, 6, pp. 17545–17556. doi: 10.1109/ACCESS.2018.2805837.

229. Wen, Y. *et al.* (2021) 'Attacks and countermeasures on Blockchains: A survey from layering perspective', *Computer Networks*, 191, p. 107978. doi: 10.1016/j.comnet.2021.107978.

230. Werapun, W. *et al.* (2022) 'The Flash Loan Attack Analysis (FAA) Framework—A Case Study of the Warp Finance Exploitation', *Informatics*, 10(1), p. 3. doi: 10.3390/informatics10010003.

231. Xiao, Y. *et al.* (2020) 'A Survey of Distributed Consensus Protocols for Blockchain Networks', *IEEE Communications Surveys and Tutorials*, 22(2), pp. 1432–1465. doi: 10.1109/COMST.2020.2969706.

232. Xing, Q., Wang, B. and Wang, X. (2018) 'BGPcoin: Blockchain-Based Internet Number Resource Authority and BGP Security Solution', *Symmetry*, 10(9), p. 408. doi: 10.3390/sym10090408.

233. Xing, Z. and Chen, Z. (2021) 'A Protecting Mechanism Against Double Spending Attack in Blockchain Systems', in *2021 IEEE World AI IoT Congress (AIIoT)*. IEEE, pp. 0391–0396. doi: 10.1109/AIIoT52608.2021.9454224.

234. Xu, G. *et al.* (2020) 'Am I eclipsed? A smart detector of eclipse attacks for Ethereum', *Computers & Security*, 88, p. 101604. doi: 10.1016/j.cose.2019.101604.

235. Xu, Y. (2018) 'Section-Blockchain: A Storage Reduced Blockchain Protocol, the Foundation of an Autotrophic Decentralized Storage Architecture', in *2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, pp. 115–125. doi: 10.1109/ICECCS2018.2018.00020.

236. Yang, J. *et al.* (2020) 'A Survey on Blockchain: Architecture, Applications, Challenges, and Future Trends', in *Proceedings - IEEE Congress on Cybermatics: 2020 IEEE International Conferences on Internet of Things, iThings 2020, IEEE Green Computing and Communications, GreenCom 2020, IEEE Cyber, Physical and Social Computing, CPSCom 2020 and IEEE Smart Data, SmartD*. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics50389.2020.00129.

237. Yap, K. Y., Chin, H. H. and Klemeš, J. J. (2023) 'Blockchain technology for distributed generation: A review of current development, challenges and future prospect', *Renewable and Sustainable Energy Reviews*, 175, p. 113170. doi: 10.1016/j.rser.2023.113170.

238. Yin, R. K. (2014). Case study research: design and methods, 5th edn. Thousand Oaks, CA: Sage Publications (Applied social research methods series, 5).

239. Zamani, E., He, Y. and Phillips, M. (2020) ‘On the Security Risks of the Blockchain’, *Journal of Computer Information Systems*, 60(6), pp. 495–506. doi: 10.1080/08874417.2018.1538709.

240. Zefeiti, S. M. B. Al and Mohamad, N. A. (2015) ‘Methodological Considerations in Studying Transformational Leadership and its Outcomes’, International Journal of Engineering Business Management, 7, p. 10. doi: 10.5772/60429.

241. Zheng, Z. *et al.* (2020) ‘An overview on smart contracts: Challenges, advances and platforms’, *Future Generation Computer Systems*, 105(2), pp. 475–491. doi: 10.1016/j.future.2019.12.019.

242. Zhong, Y. *et al.* (2021) ‘Distributed Blockchain-Based Authentication and Authorization Protocol for Smart Grid’, *Wireless Communications and Mobile Computing*. Edited by S. H. Islam, 2021, pp. 1–15. doi: 10.1155/2021/5560621.

243. Zhu, X. *et al.* (2020) ‘An Improved Proof-of-Trust Consensus Algorithm for Credible Crowdsourcing Blockchain Services’, *IEEE Access*, 8, pp. 102177–102187. doi: 10.1109/ACCESS.2020.2998803.

244. Zipfel, K. (2020) *New Smart Contract Weakness: Hash Collisions With Multiple Variable Length Arguments*. Available at: https://medium.com/swlh/new-smart-contract-weakness-hash-collisions-with-multiple-variable-length-arguments-dc7b9c84e493 (Accessed: 17 October 2022).

APPENDICES

Appendix A: Questionnaire for Experts' Feedback

Appendix A.1 - Participant 1

Questionnaire

UNIVERSITY OF
GLOUCESTERSHIRE

Inquiries: PhD Research on Blockchain Security

Email: [redacted]

Dear respondents,

You are kindly requested to participate in the research questions conducted by Sepideh Mollajafari, a Cyber and Technical Computing Lecturer at the University of Gloucestershire. This questionnaire is supported by the School of Business, Computing and Social Sciences, University of Gloucestershire. Your participation is voluntary, and your identity will be kept anonymous. Be assured that your responses to the questions are for research purposes only and will not be used outside of this study. Finally, we appreciate your participation in this study. If you would like to receive the results of this research, we will be happy to send these to you in due course.

Overview of Blockchain Technology and the Problem at Hand

This research focuses on smart contracts' centralised ownership, which pose major security issues and act as a single point of failure, thus contradicting the very decentralised nature of blockchain. To mitigate against the risks associated with centralised control, a decentralised application with DAO structure is developed. the developed "Genuine DAO ", implemented a smart contract to control critical functions and prevent one-owner control by developer/owner. In addition, this program is developed in a way to prevent Frontrunning attack. The code of Genuine DAO, both the backend and frontend, is available on the GitHub repository https://github.com/Sepideh-M/Genuine-DAO--Contracts.

Your kind cooperation is highly appreciated in this regard.

Sincerely,

Sepideh Mollajafari

SECTION A: Please complete all following questions by inserting a tick mark (**√**) in the boxes or by writing in the spaces provided.

---

1. Your gender: ☐ [✓ Male] ☐ Female

2. How many years of experience do you have as a Solidity Developer or Security Analyser?
   ☐ <3 years ☐ [✓ 3-5 years] ☐ More than 5 years

SECTION B: Please answer the following questions.

`

1. In your view, how does the proposed "Genuine DAO" address potential centralisation risks, such as concentration of power, control, or decision-making authority?

   ANSWER: In this DAO model, the ownership of the smart contracts is not owned by a wallet address and the ownership is transferred to the DAO smart contract so any change in the smart contract main functions and rules only can be executed by DAO members with the power of voting. This point solves the centralised management issues of the smart contracts.

2. How effective are the implemented security controls and countermeasures to mitigate against one owner control risks?

   ANSWER: We see some features in the DAO smart contract that actually have significant effect on the ownership improvements, such propose changes by creating proposals, and voting to the proposals in the specific period of time. So, we can consider these solutions as main improvements to avoid risks of the ownerships of the smart contracts.

3. How effective is the proposed "Genuine DAO" in leveraging layer 2 scaling solutions to enhance scalability and throughput and to minimise cost on Polygon network?

   ANSWER: What layer2 solutions offer is more about the taking the responsibly of the execution tasks from layer one and storing data in the layer1. So, layer 2 solutions increase the scalability and decrease the fee at the same time of being secure and relied one layer1. Therefore, polygon as a layer2 blockchain offers lower fee because of the execution optimisations and better scalability.

4. In your opinion, what are the most critical centralisation risks or vulnerabilities that should be addressed by future research and development?

   ANSWER: One of the most import points about blockchain technology that makes that interesting is about trust and privacy. We still have some problems about privacy that can be solved by blockchain, like proof and verification related actions in the economic actions. One of the new technologies that can make some solutions about this challenge is zero knowledge proof that I suggest to involve that in your research.

5. What specific part, if any, of the proposed "Genuine DAO" needs particular attention for improvement?

   ANSWER: You can decrease functions to have shorter process for proposals to be finalised and users can run this process in the faster time.

Thank You Very Much for Your Precious Time ☺

Appendix A.2 - Participant 2

Questionnaire

UNIVERSITY OF GLOUCESTERSHIRE

Inquiries: PhD Research on Blockchain Security

Email: [redacted]

Dear respondents,

You are kindly requested to participate in the research questions conducted by Sepideh Mollajafari, a Cyber and Technical Computing Lecturer at the University of Gloucestershire. This questionnaire is supported by the School of Business, Computing and Social Sciences, University of Gloucestershire. Your participation is voluntary, and your identity will be kept anonymous. Be assured that your responses to the questions are for research purposes only and will not be used outside of this study. Finally, we appreciate your participation in this study. If you would like to receive the results of this research, we will be happy to send these to you in due course.

Overview of Blockchain Technology and the Problem at Hand

This research focuses on smart contracts' centralised ownership, which pose major security issues and act as a single point of failure, thus contradicting the very decentralised nature of blockchain. To mitigate against the risks associated with centralised control, a decentralised application with DAO structure is developed. the developed "Genuine DAO ", implemented a smart contract to control critical functions and prevent one-owner control by developer/owner. In addition, this program is developed in a way to prevent Frontrunning attack. The code of Genuine DAO, both the backend and frontend, is available on the GitHub repository https://github.com/Sepideh-M/Genuine-DAO--Contracts.

Your kind cooperation is highly appreciated in this regard.

Sincerely,

Sepideh Mollajafari

SECTION A: Please complete all following questions by inserting a tick mark (**√**) in the boxes or by writing in the spaces provided.

---

3. Your gender: ☐ [✓ Male] ☐ Female

4. How many years of experience do you have as a Solidity Developer or Security Analyser?
   ☐ <3 years ☐ [✓ 3-5 years] ☐ More than 5 years

SECTION B: Please answer the following questions.

`

6. In your view, how does the proposed "Genuine DAO" address potential centralisation risks, such as concentration of power, control, or decision-making authority?

   ANSWER: implementing DAO structure that involve community participation in decision-making rather than relying on a single entity or an owner with elevated privilege would enhance the security. Taking control of developer and transferring the ownership to DAO contract for critical functions is beneficial to reduce the centralisation risks.

7. How effective are the implemented security controls and countermeasures to mitigate against one owner control risks?

ANSWER: limiting the access control to critical function by implementing features such as contract ownership would enhance the security. In addition, having timelock at different stage of proposal creation and voting process would improve the security.

8. How effective is the proposed "Genuine DAO" in leveraging layer 2 scaling solutions to enhance scalability and throughput and to minimise cost on Polygon network?

   ANSWER: layer 2 solutions provide a more scalable network as by processing transactions off-chain. This technology enables faster transaction confirmation times and decreases transaction cost by reducing the computational and storage costs. Many developers and users prefer to use layer 2 solutions over Ethereum Mainnet.

9. In your opinion, what are the most critical centralisation risks or vulnerabilities that should be addressed by future research and development?

   ANSWER: one area that needs to be checked as part of centralisation risks would be the owner's privilege to control over minting functions.

10. What specific part, if any, of the proposed "Genuine DAO" needs particular attention for improvement?

    ANSWER: Techniques such as zero-knowledge proofs or homomorphic encryption can be applied to enhance confidentiality and privacy and overall security.

Thank You Very Much for Your Precious Time ☺

Appendix A.3 - Participant 3

Questionnaire

UNIVERSITY OF GLOUCESTERSHIRE

Inquiries: PhD Research on Blockchain Security

Email: [redacted]

Dear respondents,

You are kindly requested to participate in the research questions conducted by Sepideh Mollajafari, a Cyber and Technical Computing Lecturer at the University of Gloucestershire. This questionnaire is supported by the School of Business, Computing and Social Sciences, University of Gloucestershire. Your participation is voluntary, and your identity will be kept anonymous. Be assured that your responses to the questions are for research purposes only and will not be used outside of this study. Finally, we appreciate your participation in this study. If you would like to receive the results of this research, we will be happy to send these to you in due course.

Overview of Blockchain Technology and the Problem at Hand

This research focuses on smart contracts' centralised ownership, which pose major security issues and act as a single point of failure, thus contradicting the very decentralised nature of blockchain. To mitigate against the risks associated with centralised control, a decentralised application with DAO structure is developed. the developed "Genuine DAO ", implemented a smart contract to control critical functions and prevent one-owner control by developer/owner. In addition, this program is developed in a way to prevent Frontrunning attack. The code of Genuine DAO, both the backend and frontend, is available on the GitHub repository https://github.com/Sepideh-M/Genuine-DAO--Contracts.

Your kind cooperation is highly appreciated in this regard.

Sincerely,

Sepideh Mollajafari

SECTION A: Please complete all following questions by inserting a tick mark (**√**) in the boxes or by writing in the spaces provided.

---

5. Your gender: ☐ [✓ Male] ☐ Female

6. How many years of experience do you have as a Solidity Developer or Security Analyser?
   ☐ <3 years ☐ [✓ 3-5 years] ☐ More than 5 years

SECTION B: Please answer the following questions.

`

11. In your view, how does the proposed "Genuine DAO" address potential centralisation risks, such as concentration of power, control, or decision-making authority?

ANSWER: Centralisation is a significant risk factor in Blockchain-based projects especially in DeFi. It can expose protocols to single points of failure, making them susceptible to attacks or manipulation. Emphasising decentralisation and community governance helps distribute decision-making power and ensures no single entity holds excessive control. Active participation by a diverse group of stakeholders in protocol governance fosters transparency, reduces conflicts of interest, and promotes the adoption of risk-mitigating measures. Genuine DAO with DAO structure reduces the centralisation risks by transferring the ownership address to the contract address and other security measures in place.

12. How effective are the implemented security controls and countermeasures to mitigate against one owner control risks?

    ANSWER: Security measures that have been taken in Genuine DAO would definitely enhance the security and reduce risk of ownership. MGold rug pull is an example of centralisation where founders used the private keys to drain the contracts of all funds. The founders decided to take the money and run which is only possible due to the centralisation privilege of them holding the private keys. Having security measures like what have been implemented on Genuine DAO such as time-lock and DAO structure alongside with a multi signature wallet would avoid this risk.

13. How effective is the proposed "Genuine DAO" in leveraging layer 2 scaling solutions to enhance scalability and throughput and to minimise cost on Polygon network?

    ANSWER: scalability, throughput and cost are ley factors in any network. Introducing layer 2 scaling would help with these challenges that Ethereum has faced.

14. In your opinion, what are the most critical centralisation risks or vulnerabilities that should be addressed by future research and development?

    ANSWER: There are many examples of scams and exploits that have taken advantage of centralisation risks. bZx protocol was exploited for more than $55 million as a result of private key mismanagement. They did not have a multi-signature wallet on their contract private keys. The attacker gained control of the private keys through a phishing email. This is a type of centralisation risk that allowed the attacker to take full control of all contracts that the keys managed. In the case of bZx, once the attacker was able to take control of the contracts, they removed tokens from both the Polygon and BSC deployments. Having multi signature wallet would add a layer of security in Genuine DAO.

15. What specific part, if any, of the proposed "Genuine DAO" needs particular attention for improvement?

    ANSWER: The Genuine DAO enhances security of blockchain transaction. We recommend to design smart contracts and voting structures in a way to limit the power of large token holders. As a result, avoid a few accounts holding the vast majority of tokens and prevent centralisation and manipulation by insiders.