HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Privacy Guard DAO: Enabling Confidential Investment and Private Voting on the Blockchain

Phạm Văn Nam
nam.pv183599@sis.hust.edu.vn

Major: Information and Communication Technology
Specialization: Computer Science

Supervisor: Dr. Trần Vĩnh Đức _____

Signature

Department: Computer Science

School: Information and Communications Technology

HANOI, 08/2023

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my supervisor Dr. Tran Vinh Duc at the School of Information and Technology, who give me invaluable guidance and support throughout the process of completing this thesis.

I would also like to extend my heartfelt thanks to my family and friends for their unwavering support. Their encouragement and belief in me have been a constant source of motivation.

Special thanks go to friends in the ZKVN team, my research group on zero-knowledge proof technology, led by Dr. Tran Vinh Duc. Their collaborative spirit, insightful ideas, and wealth of knowledge have been pivotal in the development of this thesis. Their feedback, comments, and constructive criticisms have helped refine my work and bring it to fruition. Without the moral support and input from these individuals, this thesis would not have been possible. I am truly grateful for their contributions and presence in my academic journey.

ABSTRACT

The emergence of blockchain technology has revolutionized various industries, offering solutions to complex challenges in finance, healthcare, and agriculture, among others. Blockchain's decentralization, transparency, and security, enabled by cryptographic techniques and consensus algorithms, have underscored its undeniable impact on diverse applications, ushering in a dynamic ecosystem of innovation. Despite these advancements, organizations face challenges in building on the blockchain, necessitating a well-designed governance process to achieve transparency and decentralization. The introduction of Decentralized Autonomous Organizations (DAOs) addresses this need, empowering stakeholders with voting rights for decision-making and automatic execution through smart contracts. However, transparency on the blockchain can expose sensitive information during the DAO's governance process, demanding solutions that strike a balance between confidentiality and transparency. To address this issue, this thesis introduces a novel DAO concept and proposes an innovative solution to achieve privacy on the blockchain. The proposed solution integrates zero-knowledge proof techniques to conceal voting information, preserving anonymity while ensuring transparent governance. The proposed solution provides a robust and secure framework for decentralized decision-making, encompassing investment choices and participation in governance processes on-chain. This thesis will present comprehensive research, system architecture design, and a prototype for demonstration. At the heart of the solution lies a decentralized IT infrastructure with a smart contract system as its core, where the integration of zero-knowledge techniques plays a pivotal role in achieving the desired privacy.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBRIVIATIONS

Abriviation	Full Expression
ABI	Application Binary Interface
API	Application Programming Interface
DAO	Decentralized Autonomous Organization
DeFi	Decentralized Finance
DKG	Distributed Key Generation
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EOA	Externally-Owned Account
EVM	Ethereum Virtual Machine
IPFS	InterPlanetary File System
JSON	JavaScript Object Notation
OOP	Object-Oriented Programming
PPT	Probabilistic Polynomial Time
REST	REpresentational State Transfer
UI	User Interface
UML	Unified Modeling Language
VSS	Verifiable Secret Sharing
ZK-SNARK	Zero-Knowledge Succinct Non-interactive Argument of Knowledge
ZKP	Zero-knowledge Proof

CHAPTER 1. INTRODUCTION

Nowadays, blockchain technology has emerged as a prominent area of research and application in computer science. It is effectively addressing complex challenges across critical industries such as financial services, food and agriculture, healthcare, and more. Notably, cryptocurrencies, with a trillion-dollar market, have been instrumental in driving innovation by introducing groundbreaking financial models that attract a diverse range of investors, from individuals to venture capitalists.

Despite the hype surrounding blockchain and debates about its significance, its substantial impact on various applications remains undeniable. This rapid growth can be attributed to the inherent characteristics of blockchain, including decentralization, transparency, and security, all facilitated by a distributed ledger framework based on cryptographic techniques and consensus algorithms. In this thesis, we will delve into the intricacies of blockchain technology and explore its high-level application, smart contracts.

The advent of blockchain technology has triggered a surge in organizations transitioning to blockchain platforms. Beyond digital currencies, a wide array of traditional financial services such as trading, lending, payment, and insurance have been adapted and innovated into the burgeoning market of DeFi (Decentralized Finance). Moreover, blockchain has found diverse applications in other fields, including crowdfunding for social organizations, secure health information storage in healthcare services, and supply chain tracking and tracing in food industry corporations.

These organizations share a common vision of leveraging blockchain technology's unique characteristics to enhance their business processes. The decentralized and transparent nature of blockchain enables them to foster trust, streamline operations, and create new opportunities in their respective industries. As blockchain continues to evolve, its impact on various sectors will undoubtedly grow, fueling a dynamic ecosystem of innovation and transformation.

1.1 The privacy problem on the blockchain

Despite the numerous advantages, organizations still encounter various challenges when building on top of the blockchain. They require a thorough understanding of the technology and a carefully designed governance process. The ultimate goal is to achieve transparency and decentralization within the organization,

leading to the emergence of a new organizational model known as DAO (Decentralized Autonomous Organization).

In the operation of this model, the primary objective is decentralized governance, where all stakeholders possess voting rights to make decisions for the organization. Following the decision-making process, an automatic execution mechanism ensures that no centralized authority can interfere, guaranteeing autonomy. This seamless execution is made possible by blockchain technology, which supports immutable programs and enables the automatic execution of smart contracts.

However, the transparent nature of blockchain also raises certain limitations for the DAO model, as external parties can observe the organization's activities. One significant limitation is the potential for opponents or external entities to monitor the governance process. This public visibility may expose sensitive information, strategies, or vulnerabilities that could be exploited to disrupt or undermine the DAO's operations.

To overcome this challenge, this thesis proposes the utilization of zero-knowledge proof techniques to conceal voting information in DAOs while maintaining transparency and ensuring the accuracy of voting results. By doing so, participants can enjoy anonymity, keeping their decision-making information private, while still upholding transparency in the organization's governance process. This demand for new DAO models seeks to strike a balance between confidentiality and transparency, addressing the need for enhanced privacy in decentralized organizations.

1.2 Objectives and scope of the graduation thesis

The objective of this thesis is to introduce the concept of "DAO" (Decentralized Autonomous Organization) and present a novel design for a DAO that prioritizes privacy in the decision-making process, known as "Privacy Guard DAO." The proposed design encompasses a well-structured business process that enables users to participate privately in the DAO. Additionally, the governance process, including voting and execution enforcement, is designed to maintain the confidentiality of participants' decision-making.

To achieve this, the thesis outlines the implementation of the proposed design through the development of a web application, which serves as a practical demonstration. The system operates seamlessly in the background, transparent to end users, while effectively ensuring privacy and confidentiality throughout the DAO's functioning.

The scope of this thesis is focused on developing and analyzing a DAO model

with fundamental functionalities tailored for organizations building products and solutions on public blockchains. The Privacy Guard DAO model allows users to establish their own DAO, which will be governed and managed by its community, effectively making the business model a DAO itself. Key aspects of the Privacy Guard DAO model include ensuring complete concealment of sensitive information, such as users' participation in specific DAOs and their voting decisions during the governance process. By implementing zero-knowledge proof techniques, the model aims to safeguard the confidentiality of participants' actions and decisions.

The deployment scope is limited to specific blockchains that share common interfaces and background technologies. The model's functionalities will be developed and assessed within this defined range of blockchains, enabling a focused and comprehensive evaluation of its effectiveness and practicality.

1.3 Methodology

To accomplish the defined objective, the development of this thesis follows a structured methodology. The initial phase involves thorough research and literature review, acquiring an in-depth understanding of foundational concepts such as zero-knowledge proof, private voting mechanisms on blockchains, and the fundamental principles of the DAO model.

Subsequently, the requirements specification, architecture design, and implementation of the Privacy Guard DAO model will be conducted sequentially. Each stage will rely on various software analysis and design techniques to shape the model into a complete and functional system. These stages will build upon the insights gained in the previous steps, and revisions may be necessary to achieve the most optimal and desirable solution.

Throughout the development process, careful consideration will be given to ensuring the model's effectiveness in addressing privacy concerns and enhancing decentralized decision-making processes. The completed Privacy Guard DAO model will be subject to thorough examination and evaluation, assessing its contributions in resolving privacy issues and its potential for future enhancements and advancements.

By adhering to this systematic approach, the thesis aims to deliver a robust and innovative Privacy Guard DAO model that contributes to the advancement of privacy-focused decentralized organizations and establishes a foundation for further improvements in the field.

1.4 Organization of Thesis

The thesis is organized as follows:

Chapter 2: This chapter provides a comprehensive overview of the essential cryptographic concepts, including distributed key generation, additively homomorphic encryption, Merkle tree, and the foundation of zero-knowledge proof technology - Zero-Knowledge Succinct Non-interactive Argument of Knowledge (ZK-SNARK). Additionally, it introduces the background knowledge of decentralized platforms, such as blockchain, transactions, smart contracts, IPFS, and the concepts of DAO.

Chapter 3: This chapter delves into the exploration of the Privacy Guard DAO model by surveying existing solutions and defining its purpose. It identifies the main actors, business processes, and requirements of the system. The survey process provides valuable insights into the pros and cons of existing solutions, offering essential lessons for the subsequent stages. The objectives and requirements defined in this chapter act as a guiding principle for the design and prototyping process.

Chapter 4: This chapter presents the application of cryptographic knowledge from Chapter 2 to design algorithms for achieving privacy in the decision-making process of participants. It also proposes an architecture design for the Privacy Guard DAO model, ensuring coverage of all specified business processes and requirements from the previous stage. These designs serve as a wireframe for the development of the prototype in the next stage.

Chapter 5: In this chapter, a prototype implementation of the proposed design is showcased. It outlines the selection of technologies and resources that align with the design specifications. The demonstrations present the functionality of the model through system documentation and application interfaces.

Chapter 6: This chapter presents the results of the development and examines potential avenues for future work. It offers a summary of the proposed solution's contributions in addressing identified issues. Furthermore, the chapter explores ways to enhance the proposed solution and suggests promising directions for future development.

2.1 Cryptographic primitives

2.1.1 Notation

Let $\mathbb{F}_p$ represent a finite field consisting of p elements, where p is a prime number, and let g denote a generator element. Let $\mathbb{G}$ be the set of points on an elliptic curve, and let G denote the generator point, while $\mathcal{O}$ represents the zero point. The function $f(\cdot)$ is used to denote a polynomial, and z represents a single variable. Bold letters, such as $\mathbf{v}$, $\mathbf{r}$, $\mathbf{o}$, $\mathbf{M}$, and $\mathbf{R}$, denote vectors. The notations $ENC(\cdot)$ and $DEC(\cdot)$ are used to denote the Elgamal encryption and decryption functions, respectively, while $H(\cdot)$ represents a cryptographic hash function.

2.1.2 Distributed Key Generation

The concept of secret sharing was independently introduced by Shamir [1] and Blakley [2] in 1979. Since that time, it has retained significant relevance within the realm of cryptographic research.

Definition 1. *In the context of secret sharing, the dealer is a person has a secret, represented as data $\mathcal{D}$, and want to distribute it among n nodes. The objective is to partition $\mathcal{D}$ into n pieces, denoted as $\mathcal{D}_1, \ldots, \mathcal{D}_n$, in such a way that:*

- *Having knowledge of any subset of t or more $\mathcal{D}_i$ pieces, $\mathcal{D}$ can be easily computed.*
- *Having knowledge of any subset of $t-1$ or fewer $\mathcal{D}_i$ pieces, $\mathcal{D}$ remains completely undetermined, with all possible values being equally likely.*

This scheme is called a (t, n) secret sharing scheme.

In secret sharing, assume that the dealer is dishonest, then there is no way for nodes to verify that indeed $\mathcal{D}_i$ is a "valid" piece of the secret $\mathcal{D}$. These limitations make Shamir's secret sharing scheme hard to use in a fault-tolerant distributed environment. To prevent malicious behavior of the dealer, Chor et al. [3] introduced an alternative method for sharing a secret, which led to the concept of verifiable secret sharing.

Definition 2. *In a (t, n) Verifiable Secret Sharing (VSS) scheme, the overall process can be divided into two main phases: the distribution phase (Dis) and the reconstruction phase (Rec).*

The distribution phase:

- *The dealer initiates the secret sharing process.*
- *The dealer divides the secret into n shares, denoted as $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$ and distributes these shares to the n nodes.*
- *Each node receives their share, which is a piece of the original secret.*

The reconstruction phase:

- *Each node broadcast their share, denoted as $\mathcal{D}'_i$. For honest nodes, $\mathcal{D}'_i$ should match their original share $\mathcal{D}_i$.*
- *The secret $\mathcal{D}$ will be reconstructed from these shares $(\mathcal{D}'_1, \mathcal{D}'_2, \ldots, \mathcal{D}'_n)$ by combining the necessary threshold of shares, typically t or more.*

The VSS scheme ensures that during the distribution phase, the dealer distributes shares in a way that allows nodes to verify the correctness of their shares. This verification capability helps to prevent malicious behavior by the dealer and ensures the integrity of the secret sharing process.

The additional cryptographic techniques and protocols maybe employed in a VSS scheme to enable the verification process, such as digital signatures, commitments or zero-knowledge proofs. These techniques enhance the security and trustworthiness of the VSS scheme.

Pedersen [4] introduced the concept of distributed key generation (DKG) and devised a protocol that eliminates the reliance on a trusted party for the selection and distribution of the secret key. Furthermore, Pedersen demonstrated a method for sharing the secret key that enables each group member to independently verify the correctness of their share. This property holds significance since the absence of a trusted party precludes the assumption of flawless computations during the key generation process.

Definition 3. *In an (t, n) DKG scheme, the overall process can be divided into three main phases: the selection phase (Sel), the distribution phase (Dis) and the reconstruction phase (Rec).*

The selection phase:

1. *Each node $\mathcal{P}_i$ chooses $x_i \in Z_p$ at random (uniform distribution) as a secret. Then a random polynomial $f_i(z) \in Z_p$ of degree at most $t-1$ is chosen such*

that $f_i(0) = x_i$. The polynomial can be expressed as:

$$f_i(z) = f_{i,0} + f_{i,1} \times z + \cdots + f_{i,t-1} \times z^{t-1}$$

where $f_{i,0} = x_i$

2. *$\mathcal{P}i$ computes $F_{i,j} = g^{f_{i,j}}$ for $0 \leq j \leq t-1$.*
3. *$\mathcal{P}i$ broadcasts the Commitment $C_i = (F_{i,j})_{j=0,1,\ldots,t-1}$ to all n nodes.*
4. *When all n nodes have broadcasted their commitments, the public key h is computed as $h = F_{1,0} + F_{2,0} + \cdots + F_{n,0}$.*
5. *Now, all nodes know the public key, but they cannot individually find the secret key s, which is computed as $s = x_1 + x_2 + \cdots + x_n$, unless they collaborate or find discrete logarithms.*

The distribution phase:

1. *$\mathcal{P}_i$ sends $s_{i,j} = f_i(j)$ secretly to $\mathcal{P}_j$ for $j = 1, 2, \ldots, n$ (in particular, $\mathcal{P}_i$ keeps $s_{i,i}$ secret).*
2. *$\mathcal{P}_i$ verifies the received share from $\mathcal{P}_j$ ($s_{j,i}$) by checking if the following equation holds:*

$$g^{s_{j,i}} = \prod_{l=0}^{t-1} F_{j,l}^{i^l}$$

If this verification fails, $\mathcal{P}_i$ broadcasts to all nodes that $\mathcal{P}_j$ is cheating.

3. *$\mathcal{P}i$ computes their share of the secret s as the sum of all received shares, $s_i = \sum_{j=1}^{n} s_{j,i}$ (including the secret share $s_{i,i}$ they kept).*

The reconstruction phase:

1. *Let $(\mathcal{P}_i)_{i \in H}$ be a group of t members who wish to reconstruct the secret key s.*
2. *Let $f^{'}(z) = \varphi_0 + \varphi_1 \times z + \cdots + \varphi_{t-1} \times z^{t-1}$ be the (unique) polynomial of degree at most $t-1$ such that $f^{'}(i) = s_i$ for $i \in H$. Then $g^{\varphi_i} = \prod_{j=1}^{n} F_{j,i}$ for $i = 0, 1, \ldots, t-1$.*
3. *This implies that $\varphi_0 = s$ as*

$$\prod_{j=1}^{n} F_{j,0} = \prod_{j=1}^{n} h_j = h$$

4. *With at least t members gathered together, they can solve the system of equa-*

tions

$$f^{'}(i) = s_i, i \in H, |H| >= t$$

using Lagrange multipliers to achieve φ_0

The keys selected as described in Definition 3 can serve the purpose of both secret communication and authentication. With knowledge of the public key, an individual can encrypt the plaintext message, denoted as m, using asymmetric encryption. Subsequently, any subset of t members can decrypt the ciphertext by employing the decipher protocol outlined in Definition 3.

2.1.3 Additively homomorphic encryption

Homomorphic encryption is a type of encryption that enables calculations to be carried out on encrypted data without the need to decrypt it. The Homomorphic Encryption scheme is characterized by four fundamental functions: key generation, encryption, decryption, and evaluation. Among these, the key generation, encryption, and decryption functions are similar to those in other encryption schemes. However, the evaluation function is a specific operation in homomorphic encryption that takes ciphertexts as input and produces a ciphertext corresponding to the function applied to the plaintext.

Definition 4. *An encryption scheme is called homomorphic over an operation + if it supports the following equation*

$$E(m_1) \oplus E(m_2) = E(m_1 + m_2), \forall m_1, m_2 \in M$$

with notation of encryption $E : (M, +) \rightarrow (C, \oplus)$ *where* M *and* C *are the algebraic structures the encryption function operates on.*

2.1.4 Merkle tree

The Merkle tree [5], also known as a hash tree, is a hash-based data structure widely used in computer science and cryptography. Each leaf node in a Merkle tree represents the hash value of a specific data block, while non-leaf nodes store the hash value of their respective child nodes. The topmost node of the tree, known as the Merkle root, contains a single hash value representing the entire dataset or a subset of it. Merkle trees typically exhibit a binary structure, where each node has a maximum of two children. This branching factor of 2 ensures that the tree branches out in a balanced manner, contributing to the efficient organization and traversal of the data structure.

The primary advantage of using Merkle trees is their ability to efficiently verify

the integrity of a large dataset using a compact Merkle root. By comparing the Merkle root with a known or trusted hash value, it is possible to quickly confirm the validity of the entire dataset without needing to traverse and verify each individual element.

Merkle trees offer several benefits in terms of data integrity and security. Firstly, any modification or tampering with a single data block will lead to a change in the corresponding leaf node, thereby affecting all the nodes along the path to the Merkle root. This property allows for the efficient detection of alterations in the dataset.

Moreover, Merkle trees support the concept of proof of inclusion or exclusion, enabling efficient verification of whether a particular data block is part of the dataset. By providing a path from the leaf node to the Merkle root along with the required sibling nodes, one can construct a compact proof that validates the presence or absence of the data block in the dataset. This property is especially useful in distributed systems, where network bandwidth and computational resources are often limited.

In conclusion, Merkle trees provide an elegant and efficient solution for verifying data integrity and membership proofs in large datasets. Their applications extend beyond blockchain technology and have found relevance in various domains, including distributed file systems, peer-to-peer networks, and cryptographic protocols. By leveraging the properties of Merkle trees, developers can ensure the integrity and security of their data while minimizing computational overhead.

2.1.5 Zero-Knowledge Succinct Non-interactive Argument of Knowledge

A Zero-Knowledge Succinct Non-interactive ARgument of Knowledge [6] is a cryptographic proof that allows a prover to convince a verifier that some statement is true, without revealing any additional information. ZK-SNARKs have three key properties:

- *Completeness:* If the statement is true, an honest prover can always convince the verifier.
- *Knowledge soundness:* If the statement is false, no cheating prover can convince the verifier except with negligible probability.
- *Zero-knowledge:* The proof reveals no information beyond the validity of the statement.

Moreover, the proof provided by the prover is both succinct and fast to verify for the verifier.

More precisely, the prover can prove knowledge of a witness $\mathbf{w}$ satisfying a predicate $\phi(\mathbf{w}; \mathbf{x})$ for some public input $\mathbf{x}$, without revealing any information about $\mathbf{w}$ other than the fact that $\phi(\mathbf{w}; \mathbf{x})$ holds. We call ϕ the proof circuit, w the private input, and $\mathbf{x}$ the public input.

Definition 5. *Let λ be a security parameter and $\mathcal{R}$ be an NP relation. A tuple of algorithm $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a zero-knowledge argument of knowledge for $\mathcal{R}$ if the following holds*

- Completeness. For every pp output by $\mathcal{G}(1^\lambda), (\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ and $\pi \leftarrow \mathcal{P}(\mathbf{x}, \mathbf{w}, pp)$

$$\Pr[\mathcal{V}(\mathbf{x}, \pi, pp = 1)] = 1$$

- Knowledge Soundness. For any PPT prover $\mathcal{P}^*$, there exists a PPT exactor $\mathcal{E}$ such that for any auxiliary string $\mathbf{z}$, $pp \leftarrow \mathcal{G}(1^\lambda), \pi^* \leftarrow \mathcal{P}^*(\mathbf{x}, \mathbf{z}, pp), \mathbf{w} \leftarrow \mathcal{E}^{\mathcal{P}^*(\cdot)}(\mathbf{x}, \mathbf{z}, pp)$ and

$$\Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge \mathcal{V}(\mathbf{x}, \pi^*, pp) = 1] \leq negl(\lambda),$$

  where $\mathcal{E}^{\mathcal{P}^*(\cdot)}$ represents that $\mathcal{E}$ can rewind $\mathcal{P}^*$,

- Zero knowledge. There exists a PPT simulator $\mathcal{S}$ such that for any PPT algorithm $\mathcal{V}^*, (\mathbf{x}, \mathbf{w}) \in \mathcal{R}, pp$ output by $\mathcal{G})(1^\lambda)$, it holds that

$$\text{View}(V^*(pp, \mathbf{x})) \approx \mathcal{S}^{\mathcal{V}^*}(\mathbf{x}),$$

  where $\text{View}(V^*(pp, \mathbf{x}))$ denotes the view that the verifier sees during the execution of the interactive process with $\mathcal{P}, \mathcal{S}^{\mathcal{V}^*}(\mathbf{x})$ denotes the view generated by $\mathcal{S}$ given input $\mathbf{x}$ and transcript of $\mathcal{V}^*$ and $\approx$ denotes two perfectly indistinguishable distributions.

2.2 Blockchain

2.2.1 Introduction

Blockchain is a decentralized, distributed ledger technology that allows multiple parties to maintain a shared and immutable record of transactions or data in a transparent and secure way. The concept of blockchain was first mentioned in 1991 as "a cryptographically secured chain of blocks [7]". However, it wasn't until 2008 with the emergence of Bitcoin [8], a cryptocurrency built on blockchain technol-

ogy, that blockchain gained widespread recognition. Unlike traditional currencies, which are issued and regulated by central banks, Bitcoin is a digital currency that operates independently of any central authority. It relies on cryptographic principles, consensus algorithms and the decentralized nature of blockchain to facilitate secure and transparent transactions.

At its core, the blockchain data structure is an ordered, back-linked list of blocks of transactions or data. These blocks are linked "back" using cryptographic hashes (such as SHA256, ethash, etc.), each referring to the previous block in the chain, creating a chronological and immutable sequence of blocks. The linkage between blocks ensures the integrity of the data within the blockchain. If any change is made to a block, the corresponding hash would be altered, breaking the chain's continuity and indicating tampering. This feature makes it extremely challenging for any participant to modify or manipulate past transactions without detection.

The distributed nature of blockchain involves storing and synchronizing copies of the entire blockchain on multiple computer systems known as validation nodes. This decentralized approach prevents any single entity from having exclusive control over the data, thereby eliminating the risk of a single point of failure. To achieve consensus and transaction ordering, widely recognized consensus mechanisms like Proof of Work (PoW) or Proof of Stake (PoS) are utilized. These mechanisms enable agreement among the validation nodes, ensuring the integrity and security of the blockchain. After a fixed number of blocks, the data appended to the chain can be considered immutable. This design establishes a tamper-evident and tamper-resistant system that embodies the core attributes of decentralization, transparency, and security.

2.2.2 Key and addresses

Public key cryptography was invented in the 1970s and is a mathematical foundation for computer and information security. Since the invention of public key cryptography, various mathematical functions, such as prime number exponentiation and elliptic curve multiplication, have been developed and found suitable for cryptographic operations. These functions possess the property of being computationally easy to perform in one direction, while computationally infeasible to reverse. As a result, cryptography enables the creation of digital secrets and unforgeable digital signatures. By employing these irreversible mathematical functions, cryptographic systems ensure the confidentiality, integrity, and authenticity of digital information. Private keys are generated using these functions, serving as the basis for encryption, decryption, and digital signature generation. Public keys,

derived from the corresponding private keys, enable the verification of digital signatures and encryption of data intended for specific recipients.

Elliptic curve cryptography (ECC) [9] is a popular branch of public key cryptography. ECC utilizes the difficulty of the elliptic curve discrete logarithm problem to ensure the security of the cryptographic operations. The computational complexity of solving this problem makes ECC resistant to attacks by both classical and quantum computers. The primary applications of ECC lie in key exchange, digital signatures, and encryption. Notably, digital signatures based on ECC, such as the Elliptic Curve Digital Signature Algorithm (ECDSA) [10], enable the verification of digital message authenticity and integrity. ECDSA serves as a vital identification mechanism in blockchain technology, offering trust and reliability in transaction verification.

A user, or an account, on the blockchain possesses a pair of public and private keys. The private key is used to sign transactions for interactions with the blockchain. It is a secret key known only to the owner and must be securely stored to prevent unauthorized access. On the other hand, the public key is derived from the private key through cryptographic algorithms. It is mathematically linked to the private key, but can be openly shared with others. The public key serves multiple purposes, including generating a unique address that represents the account's identity on the blockchain. It also plays a vital role in the signature validation process within the blockchain network.

2.2.3 Transaction

Transactions constitute the fundamental building blocks of the blockchain network, serving as the pivotal elements that facilitate state changes within the system. The entire infrastructure of the blockchain is intricately designed to streamline the lifecycle of transactions, encompassing their creation, propagation, validation, and ultimate inclusion into the blockchain network state.

Within the blockchain context, transactions represent user-driven activities aimed at modifying the state of the blockchain. When a user initiates a transaction, it needs to be verified and validated by the nodes on the blockchain network. This verification process involves ensuring that the transaction adheres to predefined rules and that the user has the necessary authorization and funds to perform the intended action. Once verified, the transaction is added to a block and included in the blockchain through the consensus mechanism, such as PoW or PoS. Once a transaction is recorded on the blockchain, it becomes virtually impossible to alter or tamper with, ensuring the integrity and transparency of the transaction history.

Transaction history is public for anyone to query and read.

When making transactions on the blockchain, it is important to note that a fee is required. This fee is often denominated in the native token of the blockchain system, particularly in the case of cryptocurrencies. The amount of the fee is variable and depends on various factors, including the design and parameters of the blockchain system, as well as the complexity of the transaction itself. Factor such as the size of transaction data and the required computing power can influence the fee calculation.

The transaction fees serves multiple purposes within the blockchain system. Firstly, it helps incentivize network participants, known as validators or miners, to include the transaction in the blockchain by compensating them for their computational efforts and resources. Additionally, transaction fees help prevent network congestion and prioritize transactions based on their value or urgency.

The record of every transaction on the blockchain is commonly referred to as the ledger. The ledger serves as a distributed and immutable database that maintains a transparent and chronological history of all transactions conducted withing the blockchain network. The ledger's distributed nature eliminates the need for a central authority, making it resistant to tampering and providing transparency to all participants. Its immutability ensures that once a transaction is recorded, it cannot be altered or deleted. As a critical component of the blockchain, the ledger plays a pivotal role in establishing trust, accountability, and transparency within the network.

2.2.4 Smart contract

In the context of blockchain technology, smart contracts refer to self-executing programs that run on a blockchain network. These programs are written in specific programming languages, such as Solidity in the case of Ethereum, and are stored and executed on the blockchain. Smart contracts on blockchain platforms like Ethereum enable the automation and enforcement of contractual terms without the need for intermediaries. It is important to note that the term "smart contract" can still be subject to interpretation and can vary depending on the context. Different blockchain platforms may have variations in their implementation and features of smart contracts. Therefore, it is crucial to consider the specific context and platform when discussing smart contracts. In the scope of this thesis, the term "smart contract" refers specifically to the implementation on the Ethereum Virtual Machine [11] compatible blockchain, written in Solidity programming language because they are the most popular combination for developing blockchain applica-

tions at the time of writing, which helps increase the generality of the solution.

The Ethereum Virtual Machine (EVM) plays a vital role within the Ethereum blockchain network as a distributed state machine. It is responsible for executing smart contracts and maintaining the overall state of the entire Ethereum network. By operating on a consensus algorithm, the EVM ensures that all participating nodes agree on the state of the blockchain and its associated smart contracts. It provides a secure and deterministic environment for code execution, enabling developers to construct decentralized applications (dApps) with diverse functionalities. The distributed nature of the EVM guarantees that all network participants have a consistent view of the blockchain's state, enabling seamless interactions and transactions. Through its consensus algorithm, the EVM facilitates the validation and execution of smart contracts, thereby upholding the integrity and dependability of the Ethereum network. Its role as a distributed state machine is essential for the effective operation and prosperity of the Ethereum ecosystem, empowering developers to innovate and create decentralized solutions.

When a contract is written in Solidity, it undergoes compilation, resulting in bytecode that represents the contract's executable instructions. This bytecode is a low-level representation of the contract's logic and functionality. It is this bytecode that is deployed onto the Ethereum network. Once deployed, the bytecode becomes immutable, meaning it cannot be altered. This immutability ensures the integrity and trustworthiness of the contract's behavior. Additionally, the bytecode is publicly accessible, allowing anyone to examine the code and verify its functionality. This transparency fosters trust among participants and enhances the auditability of smart contracts. The process of compiling Solidity code into bytecode and deploying it on the Ethereum network is essential for the execution and proper functioning of smart contracts within the decentralized ecosystem.

In the Ethereum blockchain, there exist two distinct types of accounts: externally owned accounts (EOAs) and contract accounts. EOAs are controlled by private keys and are typically associated with individual users or entities. They are uniquely identified by their Ethereum addresses, which are derived from their corresponding public keys. The Ethereum address, a 20-byte string, serves as a shared identifier for both EOAs and contract accounts. On the other hand, contract accounts, also known as smart contracts, are controlled by program code executed by Ethereum Virtual Machine. These contract accounts are created through transactions initiated by EOAs and possess their own Ethereum addresses. In summary, EOAs represent basic accounts without code or data storage, while contract accounts incorporate both code and data storage capabilities. This distinction between

EOAs and contract accounts forms the core framework enabling various functionalities and interactions within the Ethereum network.

Within the Ethereum state machine, there are three primary types of transactions: value transfers, contract creations, and contract interactions. Value transfers involve the transfer of the native cryptocurrency (Ether) between accounts, enabling transactions similar to traditional currency transfers. Contract creations occur when a user deploys a smart contract to the Ethereum network, resulting in the creation of a new contract account. Contract interactions on the Ethereum blockchain involve executing specific functions within a smart contract, enabling users to interact with the contract's code and access its stored data. These interactions typically occur through execution calls, which are facilitated by the Application Binary Interface (ABI). The ABI is a JSON specification that provides a standardized way to encode the data required for executing calls to a contract. It describes the structure and format of the function calls, enabling developers to properly encode and decode the necessary data for interacting with the contract. By utilizing the ABI, users can effectively communicate with smart contracts on the Ethereum network, ensuring compatibility and consistency in contract interactions.

2.3 IPFS

As mentioned previously, storing data on the blockchain incurs gas fees for each read/write operation. To mitigate unnecessary costs for users, external storage options are commonly utilized. These options typically involve hosting data on server systems or leveraging decentralized networks. For systems prioritizing decentralization and transparency, decentralized storage is preferred. In the proposed architecture, IPFS (InterPlanetary File System) [12] is employed as the external storage solution for the smart contract system. IPFS is a distributed system designed for storing and accessing files, websites, applications, and data. It utilizes content addressing, where files are assigned unique identifiers generated through cryptographic hash functions based on their content. Once uploaded to the IPFS network, files can be accessed publicly by anyone. In the integration with the smart contract system, instead of storing large data on-chain, they are uploaded to IPFS, and their hashes are referenced in the contract as storage variables. This approach significantly reduces the complexity and cost of data manipulation to a constant level. However, it introduces a more intricate business process as a trade-off.

2.4 Decentralized autonomous organization

2.4.1 Introduction

For decades, organizations have operated under centralized structures. Traditional centralized organizational models have long been the prevailing approach, characterized by hierarchical structures and concentrated decision-making power. However, these models have limitations. Power concentration within centralized models often leads to bottlenecks and delays, hindering organizational agility. Moreover, the lack of transparency and accountability can impede effective governance. Additionally, reliance on intermediaries introduces extra costs and potential points of failure, diminishing efficiency and increasing dependency.

In an era marked by rapid technological advancements and the revolutionary potential of blockchain, a new paradigm of organizational structures has emerged: Decentralized Autonomous Organizations (DAOs). Initially conceptualized as Decentralized Autonomous Corporations [13] by Vitalik Buterin, the founder of Ethereum, DAOs have evolved into a distinct organizational model outlined in the Ethereum white paper [11]. This innovative framework aims to address long-standing challenges in traditional governance processes, offering a promising solution for organizations seeking to enhance transparency, autonomy, and efficiency.

The emergence of Decentralized Autonomous Organizations (DAOs) has presented a partial solution to the challenges encountered in traditional governance processes. DAOs possess the potential to facilitate decentralized finance and redefine governance mechanisms. This section aims to delve into the fundamental principles of DAOs, explore their operational mechanisms, examine their benefits, and acknowledge their limitations. By understanding the operation of DAO, we can gain insight into their transformative potential and contribute improvements to achieve the privacy in decision-making within the DAO model on the blockchain.

2.4.2 Governance and decision-making in DAOs

Governance and decision-making in DAOs play a crucial role in shaping their operations and outcomes. Unlike traditional centralized organizations, DAOs leverage blockchain technology and smart contracts to create transparent, decentralized, and autonomous governance systems.

At it cores, the governance work can be broken-down into two main sequential processes: decision-making and execution. The decision-making process involves identifying issues or proposals that necessitate community input or consensus, such as making changes to products, selecting investment partners, or enforcing member bans for illicit activities. This process often includes discussions, debates, and vot-

ing among stakeholders to reach consensus or make important decisions. Various voting mechanisms, such as majority voting, quadratic voting, or token-weighted voting, may be employed to determine the outcome of decisions.

Once decisions are reached, the execution process ensues, entailing the implementation of agreed-upon actions, changes, or policies. In DAOs, execution is often facilitated by smart contracts, which are self-executing code deployed on the blockchain. Smart contracts automatically execute predetermined actions based on the decisions made through the governance process. For instance, if a proposal to allocate funds for a specific project is approved, the smart contract would release the funds accordingly.

The decision-making and execution processes in DAO governance are interdependent and vital for effective functioning. Decision-making ensures collective choices that reflect the will of the DAO community, enabling open participation through ownership of the organization's governance token (a special type of cryptocurrency based on smart contracts, issued by the organization). Execution ensures transparent implementation without intermediaries, facilitated by smart contract automation.

Although this idea holds great promise, a notable challenge in its adoption is the necessity for developers with extensive proficiency in blockchains and smart contracts. The first implementation of this model, known as "The DAO," experienced a significant loss of millions of dollars in 2016 [14] due to a smart contract exploit known as a reentrancy attack. Despite claims of audits by top security specialists, vulnerabilities still existed. The decentralized environment created by blockchain introduces new attack vectors, such as smart contract exploits, malicious proposals, and poorly-designed mechanisms. Furthermore, the immutability of smart contracts, a prominent feature of the system, becomes a vulnerability. Even the most skilled development teams can make oversights, and once vulnerable smart contracts are deployed, rectifying them becomes a challenge without abandoning the existing contracts and migrating the system to a new framework with necessary fixes. Addressing this issue requires techniques to enhance the upgradability of a DAO's smart contract system.

We will examine the DAO modules, particularly the blockchain infrastructure, employed by two leading DeFi projects in the market, Aave and Compound. These projects are regarded as battle-tested architectures within the industry. By studying their implementations, we aim to understand how they incorporate the DAO concept and explore the solutions they have devised to tackle the challenges mentioned

earlier.

Aave [15], a renowned open-source protocol for non-custodial liquidity markets, employs a DAO module to govern its smart contracts. This module comprises governance processes and a set of smart contracts responsible for managing the lending pool protocol, evaluating risks, and allocating community-led funds from Aave's treasury. Any protocol updates or grant request must be approved through Aave Improvement Proposals, subject to community voting. Users interact with the Governance Contract to cast their votes, with the proposal's data and voting results publicly stored. The Governance Strategy implements the voting mechanism, finalizing proposals as either successful or unsuccessful based on the community's decision. Voting power is determined by the user's ownership of Aave tokens prior to the voting period. Timelock contracts have authority over the protocol's smart contracts, will have responsible to execute actions stored in Governance contract if proposals successfully approved. The governance structure of Aave's DAO includes a guardian entity for emergency protection.

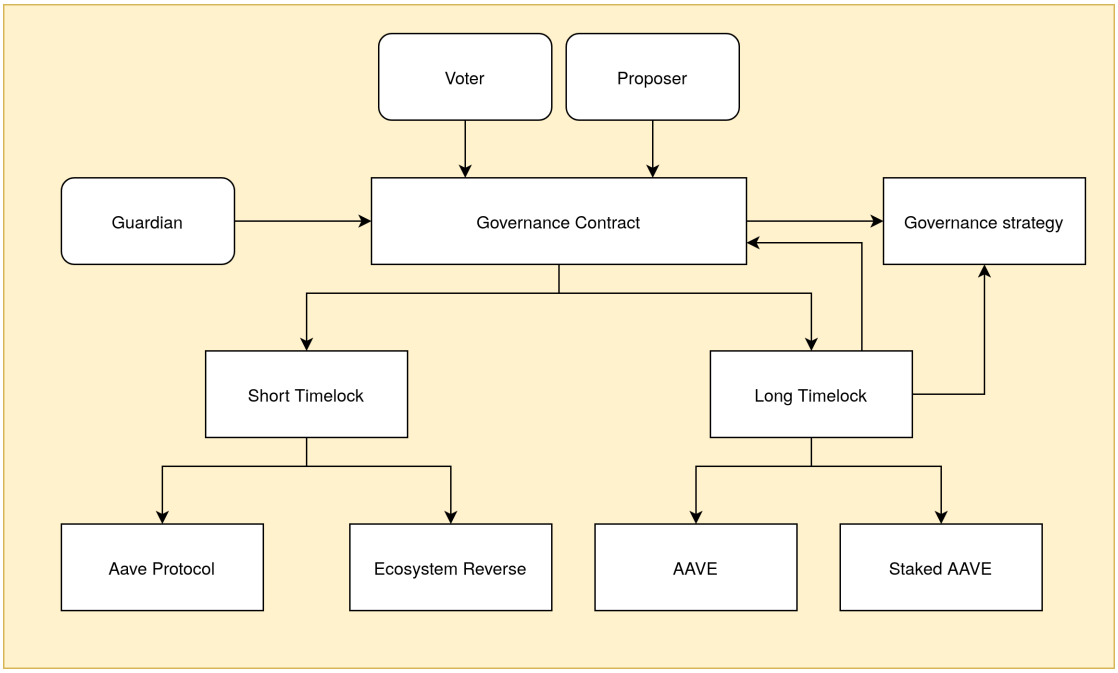

Figure 2.1: Aave Governance module [16]

Compound [17] is also a DeFi lending protocol that uses the DAO model for its Governance module to govern the smart contract system. Similar to Aave and the original DAO concept, Compound implements a workflow centered around improvement proposals.

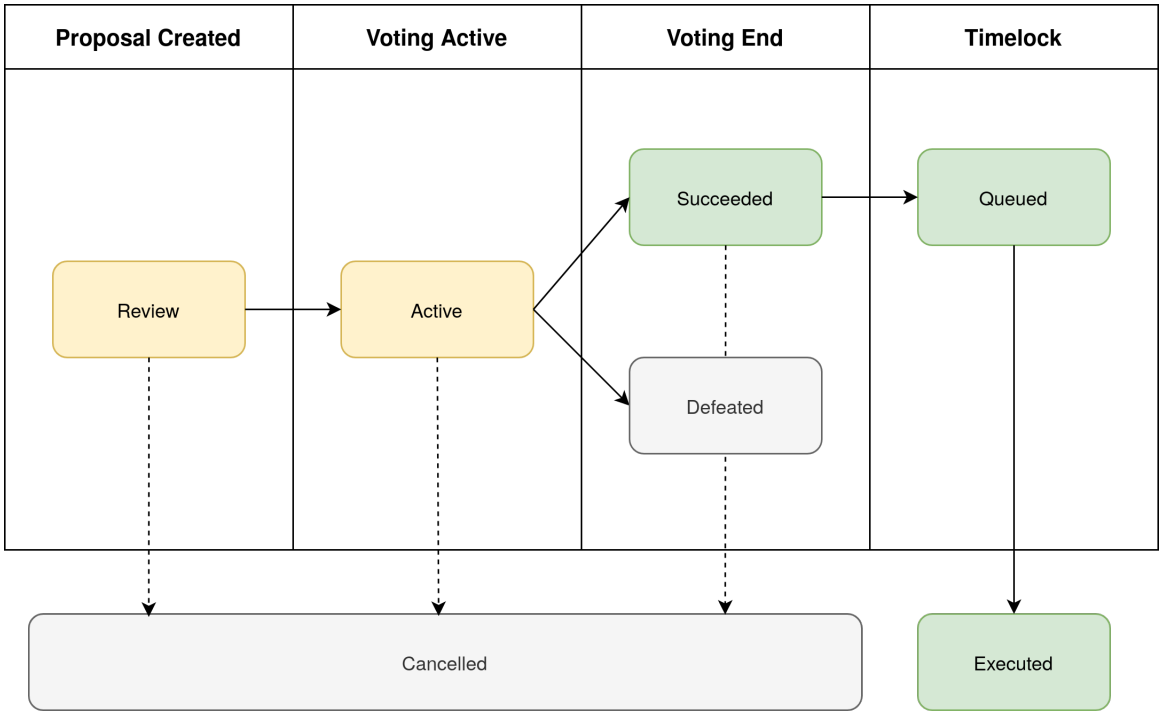

Figure 2.2: Compound's Improvement Proposal lifetime [18]

The approach of Compound is much simpler than Aave. Notably, Compound Governance Contract employs upgradable pattern, enables smart contract upgradability without necessitating costly migration processes. Consequently, Compound's DAO structure has gained widespread adoption and has been frequently forked by DeFi projects on EVM-compatible blockchains. However, in pursuit of simplicity, the module lacks flexibility, implementing only a single configuration for governance processes and relying on a single timelock contract to govern all smart contracts within the system. This limitation may be a consideration for organizations with intricate structures and diverse ecosystems encompassing multiple products.

CHAPTER 3. PRIVACY GUARD DAO

3.1 Existing solutions

The main idea of the “Privacy Guard DAO” is to construct a DAO model where users can participate in selecting the projects they wish to invest in and have the right to engage in the governance process of those projects. The term "Privacy Guard" in this context signifies that the information regarding their decision-making process, including their investment choices and decisions within the governance process of each project, will be kept private. However, it is important to note that, as of now, *there is no existing DAO model that successfully achieves information concealment during the governance process for participating individuals in terms of privacy*.

This DAO model can be seen as the investment DAO and can be known as "Ventures DAO". Ventures DAO represents a community of investors who pool their capital together to engage in investment activities in the market. Currently, there are several prominent Ventures DAOs in the market, including BitDAO [19], The LAO [20],. . . .

For The LAO, this is a notable project in the realm of the investment DAO. It operates as a member-directed venture capital fund organized in the United States, with a focus on compliance with U.S. laws. Members of The LAO can pool their capital, invest in various projects, and share in the proceeds generated from those investments. The LAO functions as a legal entity, specifically a Delaware limited liability company, with its administration primarily facilitated through an online application (DApp) and related smart contracts. Since its establishment in April 2020, The LAO has provided backing for over 130 projects within the Ethereum and blockchain ecosystem.

3.2 Purpose

The objective of the Privacy Guard DAO model proposed in this thesis is to address privacy issues for investors, encompassing both the privacy of the amount of money they have invested and the privacy of voting on project operations through proposals. The approach outlined in this thesis offers the advantage of facilitating the operation of a DAO (Decentralized Autonomous Organization) for investors who have made financial contributions in a convenient manner, with just one transaction. Furthermore, it ensures privacy for all investment decisions, providing an additional benefit for participants.

The concept of the Privacy Guard DAO stems from a practical necessity: the desire to invest in the Company A while maintaining strict confidentiality regarding the number of shares owned and the voting patterns concerning the company's operations. By implementing this Privacy Guard DAO model, investors can safeguard their privacy while actively participating in the investment and governance processes within the DAO.

The Privacy Guard DAO model includes a design for a DAO model, an infrastructure for the creation and management of DAOs on the blockchains, and an application providing an interface for user interactions. These components empower the admins of the organization in delivering and managing their organization in a DAO structure. Additionally, they enable the members to transparently and easily participate and observe the governance processes, while ensuring the confidentiality of each member's decisions during their engagement in the DAO's governance processes.

3.3 Actors

There are three actors in the proposed system, covering all end users targeted to use the provided model.

Actor	**Description**
Unconnected Account	Users who haven't connected their wallet to the client app
Connected Account	Users who have connected their wallet to the client app
Committee Account	Users who have connected to a client app and have the committee role in the smart contract

Table 3.1: System actors

3.4 Business processes

3.4.1 General business processes

The Privacy Guard DAO model encompasses two primary activities: investment and governance within DAOs.

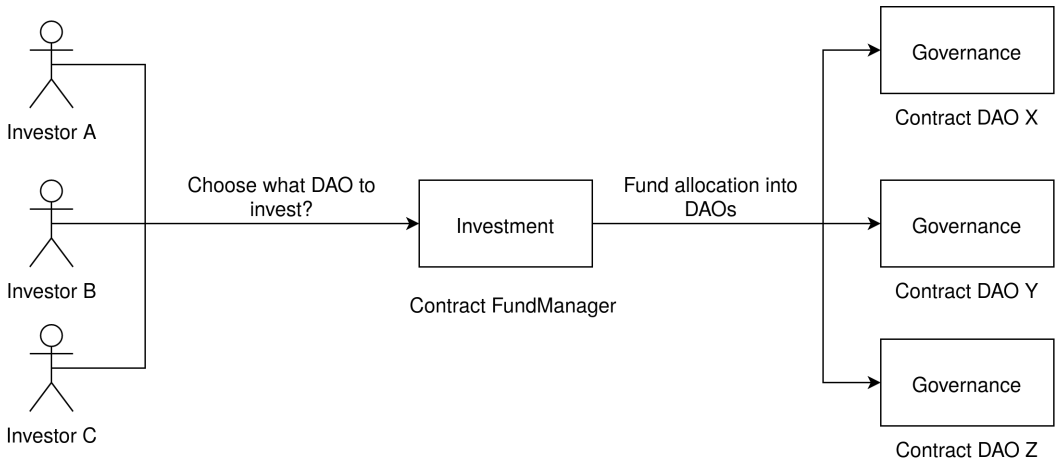

Figure 3.1: General process for Privacy Guard DAO model

The Privacy Guard DAO model achieves both privacy in the investment and governance processes within DAOs. This model ensures that external observers are unable to ascertain the specific investment amount contributed by Investor A to DAO X, nor do they possess knowledge of Investor A's voting preferences on proposals within DAO X. To achieve this privacy, Privacy Guard DAO model implement a "Committee" structure, comprising dedicated members who serve the community through the provision of "Distributed Key Generation" mechanism that facilitates privacy in the investment and governance processes within DAOs.

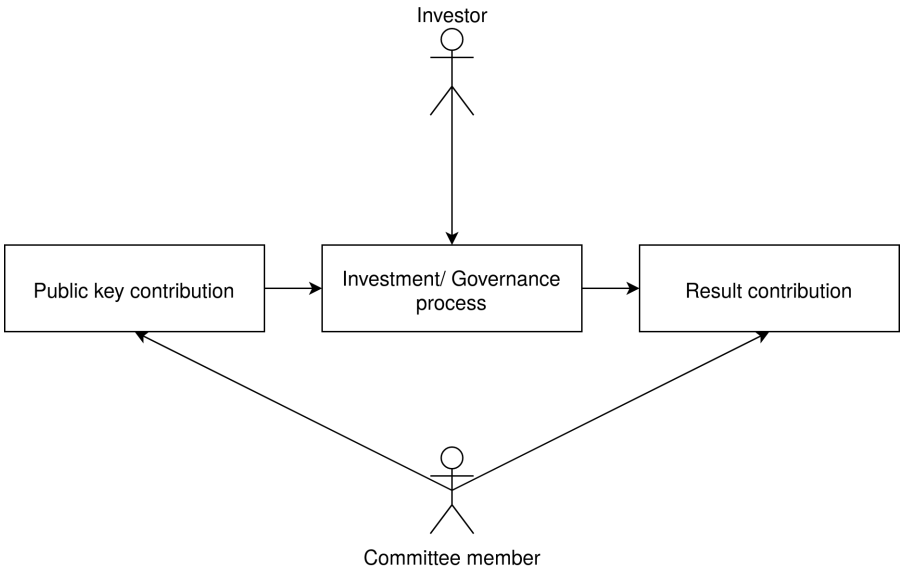

Figure 3.2: Distributed key generation mechanism

The Distributed Key Generation mechanism comprises two primary processes: public key contribution and result contribution. The public key contribution process is designed to establish a pair of public and private keys. This process strictly prohibits any individual, including committee members, from having access to the

private key. Each committee member possesses only partial knowledge of the private key, ensuring its confidentiality. Following the public key contribution, the investment or governance processes can utilize this public key to conceal sensitive information pertaining to investors' decision. Upon the conclusion of the investment or governance processes, the participation of committee members in the result contribution process becomes crucial to achieve the final outcome. Similar to the public key contribution process, strict confidentiality is maintained, prohibiting anyone, including committee members, from obtaining knowledge of the private key. The only information obtained by committee members after this process is the final result of the investment or governance process.

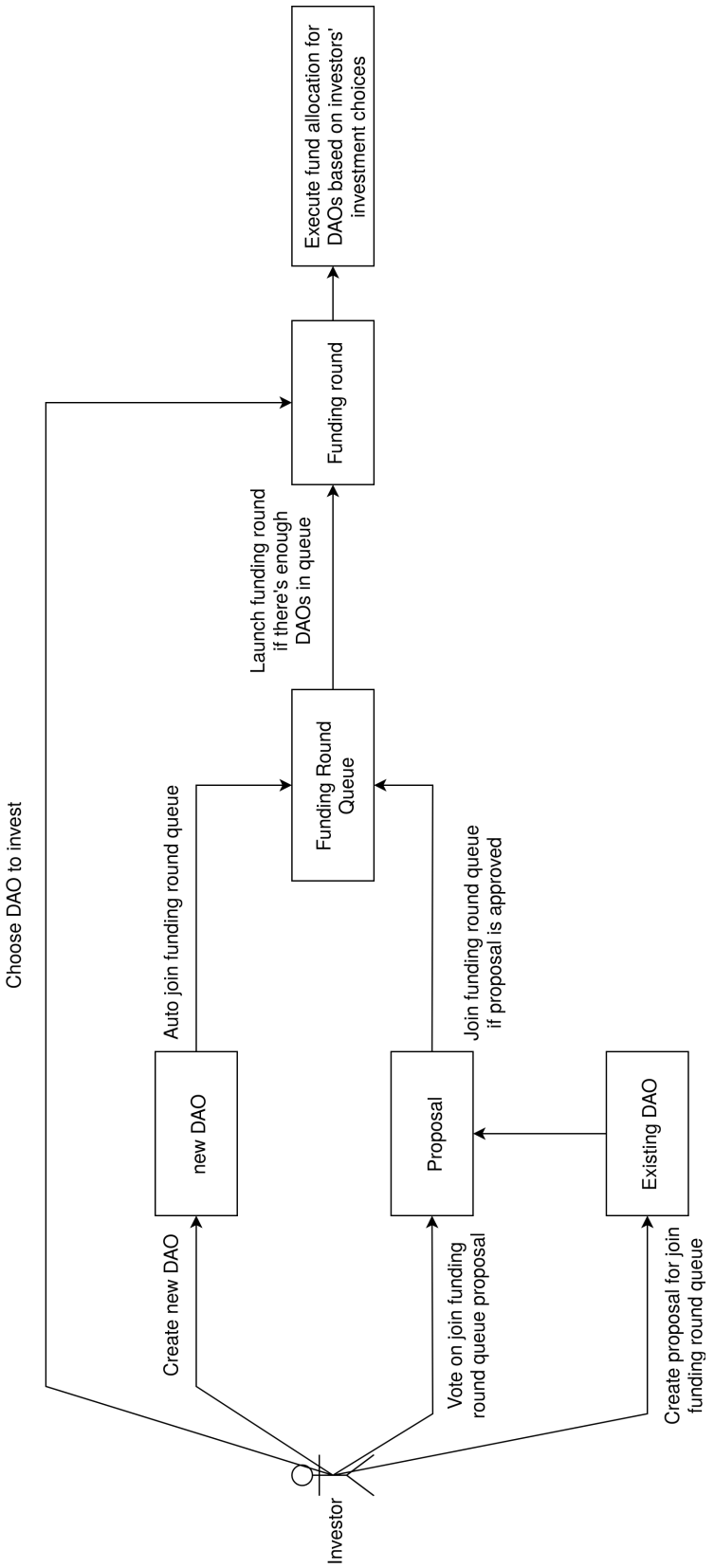

Figure 3.3: Overall investment process in Privacy Guard DAO model

The investment process within the Privacy Guard DAO model constitutes an essential activity through which investors raise capital for new projects or invest in existing projects. Each project within the Privacy Guard DAO is referred to as a "DAO," and investors are required to make investments to acquire governance rights in the corresponding DAO. This investment activity plays a pivotal role in facilitating the growth and development of projects within the Privacy Guard DAO ecosystem.

Through the investment processes, investors acquire the right to participate in the governance processes of respective DAOs. DAOs that are in the Privacy Guard DAO model should follow the following processes. Any modifications or updates to smart contracts must be approved through the "Improvement Proposal" process.

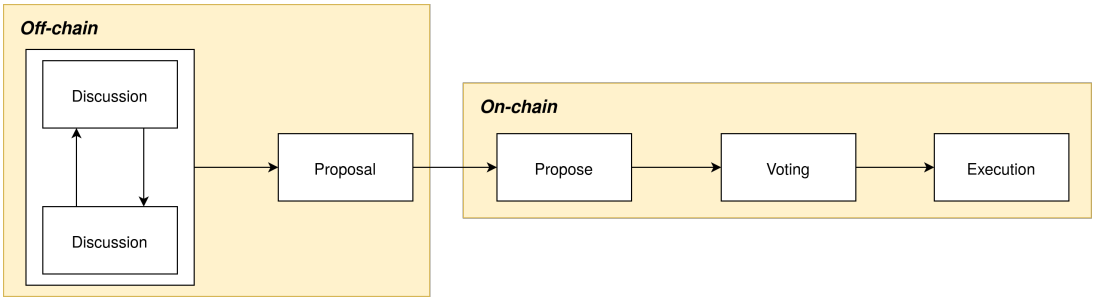

Figure 3.4: Recommended process for an Improvement Proposal in DAO

The "Improvement Proposal" process comprises two distinct stages: off-chain and on-chain. The off-chain stage occurs outside of decentralized environments and focuses on optimizing governance processes, while the on-chain stage takes place on blockchains, where transactions are recorded publicly. During the off-chain stage, proposed changes to the DAO's product undergo comprehensive presentation, discussion, and review by stakeholders, ensuring correctness, necessity, and security. Once thoroughly assessed, the proposal is specified in text and actions for the subsequent on-chain stage. On-chain, the proposal requires majority approval from stakeholders to be executed. The details related to the on-chain stage will be provided by the following diagram.

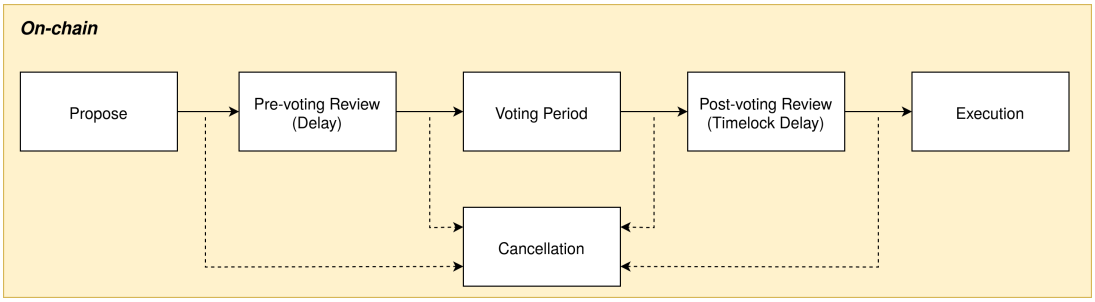

Figure 3.5: Details on the on-chain process

In the pre-voting review stage, the proposal's essential data will be initialized and stored on-chain by smart contracts. After the short period of pre-voting review stage, anyone has governance power can participate in voting period. If the proposal is approved, it will be queued to waiting for execution. The execution is implemented by timelock mechanism. After waiting for the post-voting process to elapse, the approved proposal can be executed by anyone to implement the agreed changes. In addition, during the post-voting process, the cancellation mechanism can be activated by the organization if any problems or potential exploits are detected.

3.4.2 Detail business processes

In this section of the thesis, business activity diagrams will be presented to help determine services to be provided by the Privacy Guard DAO model.

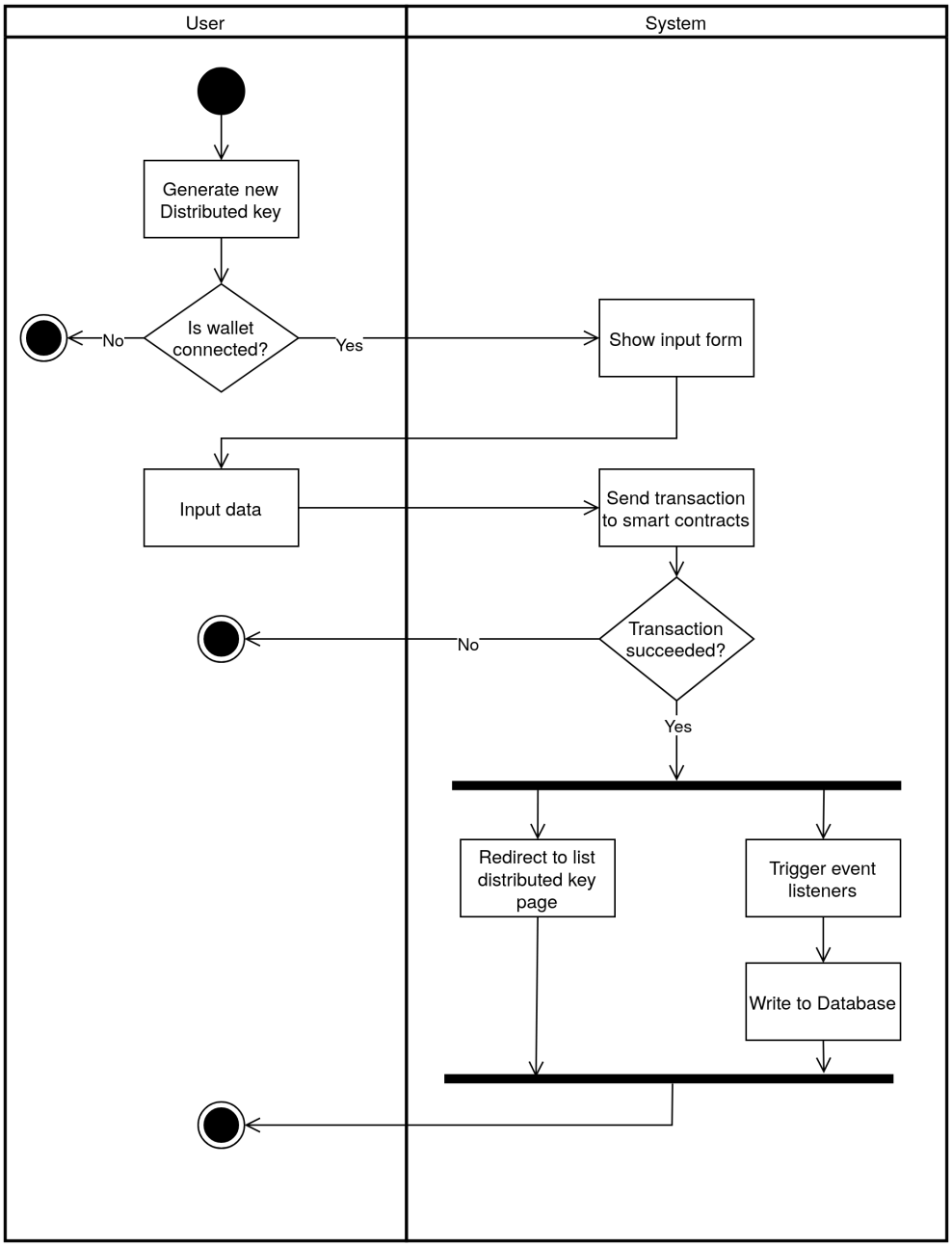

Figure 3.6: Distributed Key Generation process

The Distributed Key Generation mechanism plays a vital role in ensuring privacy within both the investment and governance processes. Because of the importance of Distributed key, so generate new distributed key process requires actors with the Committee Account role to succeed. These actors are responsible for generating and managing distributed keys, thereby upholding the integrity and security of the mechanism.

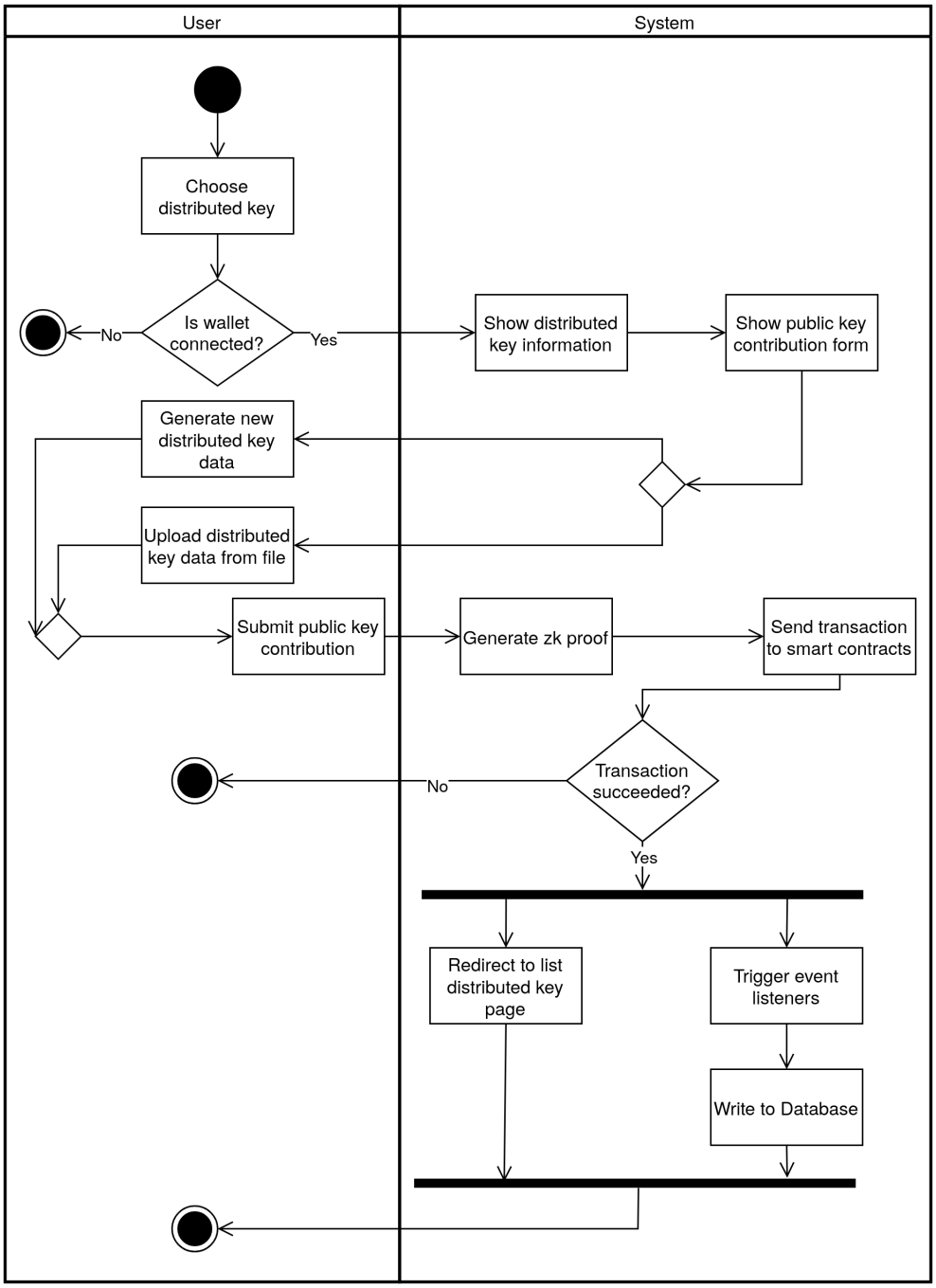

Figure 3.7: Public key contribution process

After the process of generate new distributed key, a new distributed key is established within the smart contracts. However, the key remains inactive and requires contributions from committee members to obtain the public key, which is utilized in investment and governance processes. To ensure data accuracy, this contribution process is protected by zero knowledge proof. After a sufficient number of committee members have submitted their contribution, the public key will be generated successfully, enabling investment or governance processes to utilize this public key.

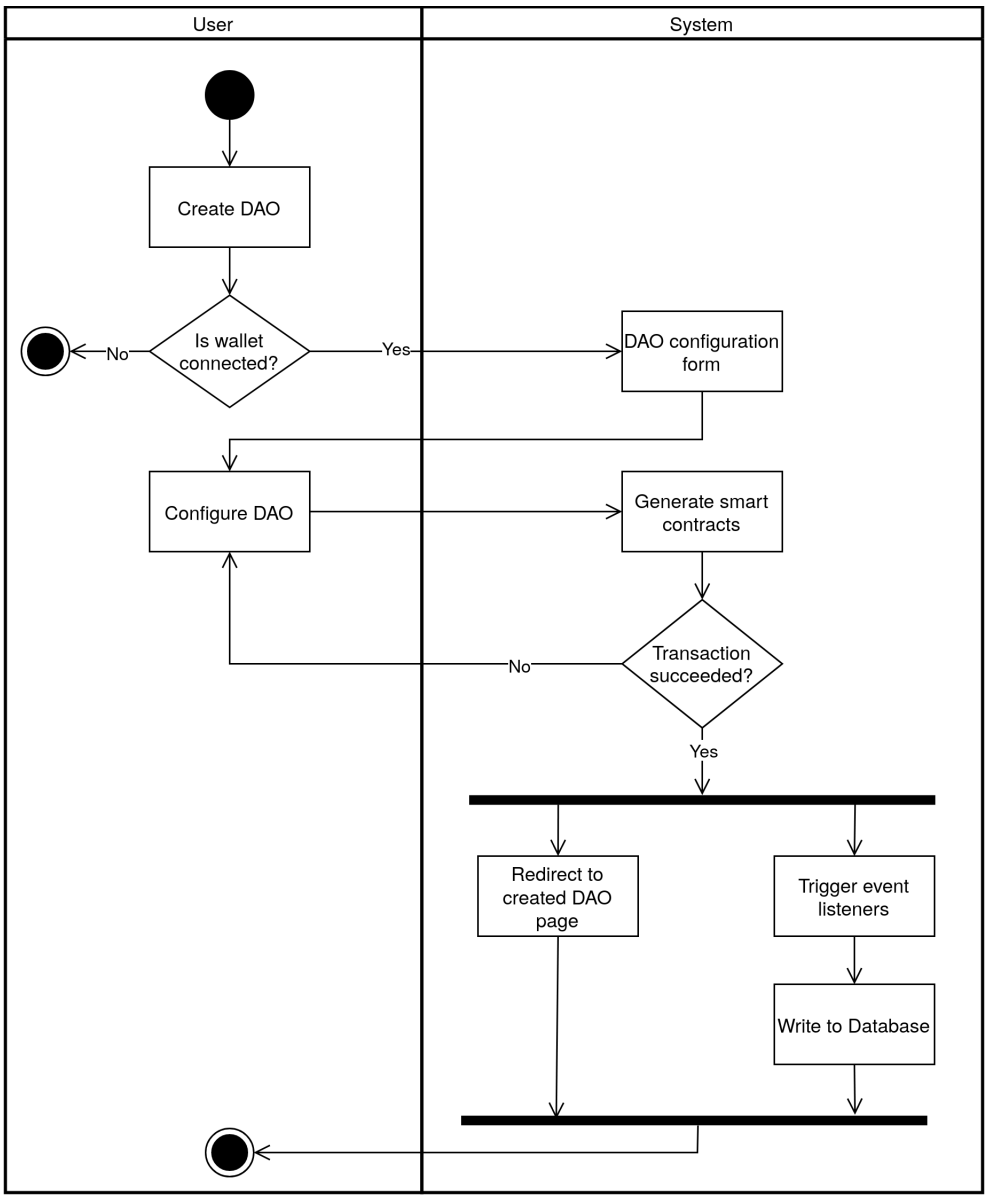

Figure 3.8: DAO creation process

The DAO creation process allows users to create a new DAO for their organizations or their projects. To perform DAO creation, users are required to connect their wallet to the client application. During the DAO creation process, users can configure parameters for their DAO. Following a successful DAO creation process, the newly formed DAO is automatically added to the funding round queue, awaiting investment opportunities.

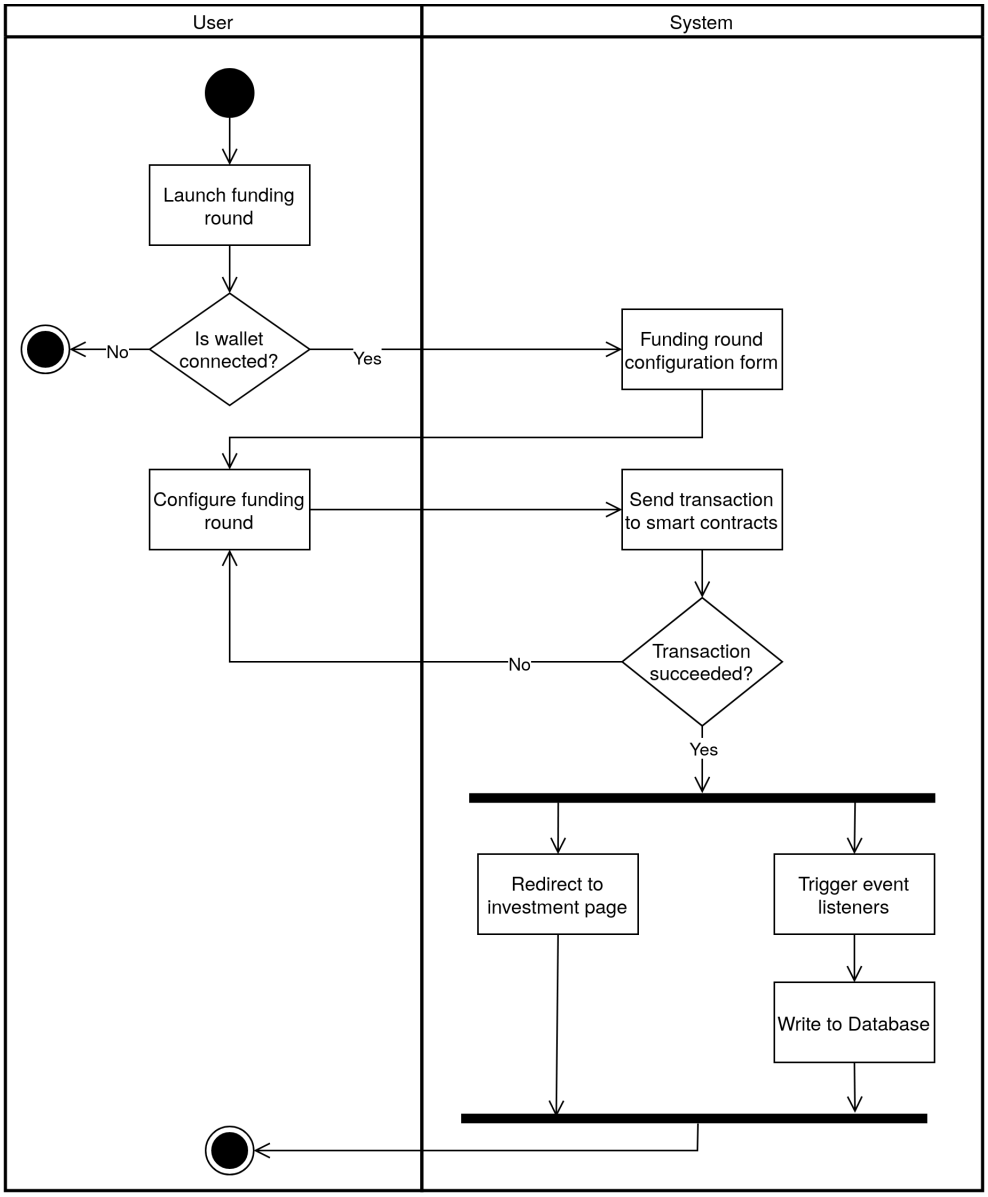

Figure 3.9: Funding round launching process

In the Privacy Guard DAO model, when a sufficient number of DAOs have entered the funding round queue, the Committee Account has the capability to initiate a new funding round. DAOs will be taken one by one from the queue to participate in the funding round. This mechanism ensures a fair and orderly allocation of funding opportunities among the DAOs within the Privacy Guard DAO ecosystem.

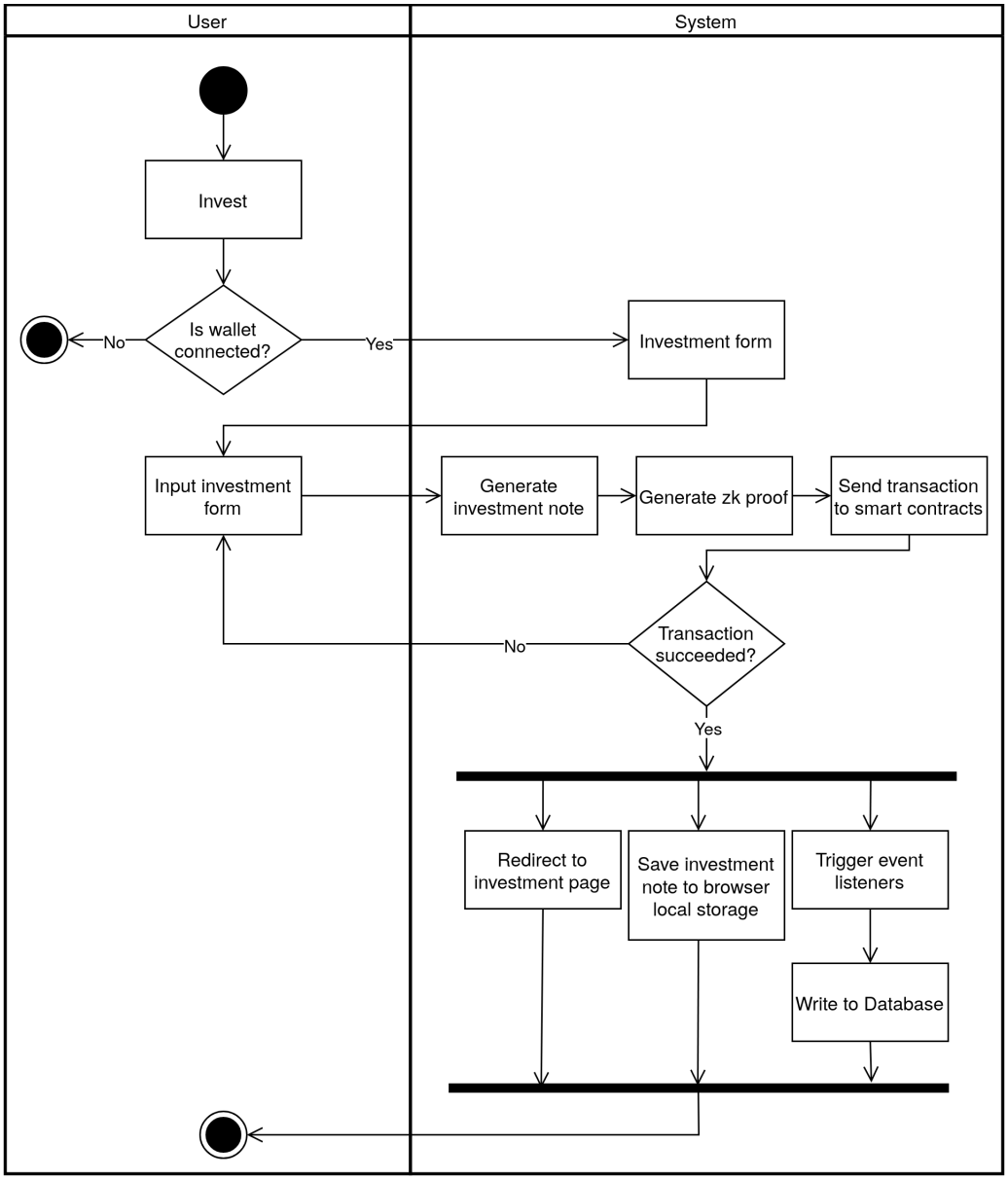

Figure 3.10: Investment process

During an active funding round, users have the opportunity to engage in the investment process within the Privacy Guard DAO model. This process requires actors with the Connected Account role to succeed. Users possess the freedom to select and invest in any desired project or DAO. To achieve the privacy, the investment process utilizes zero knowledge proof techniques, ensuring that the information about which DAO a user invests in remains concealed. Following an investment, it is crucial for users to retain an "investment note," as this data holds significant importance for future participation in governance processes related to that particular DAO.

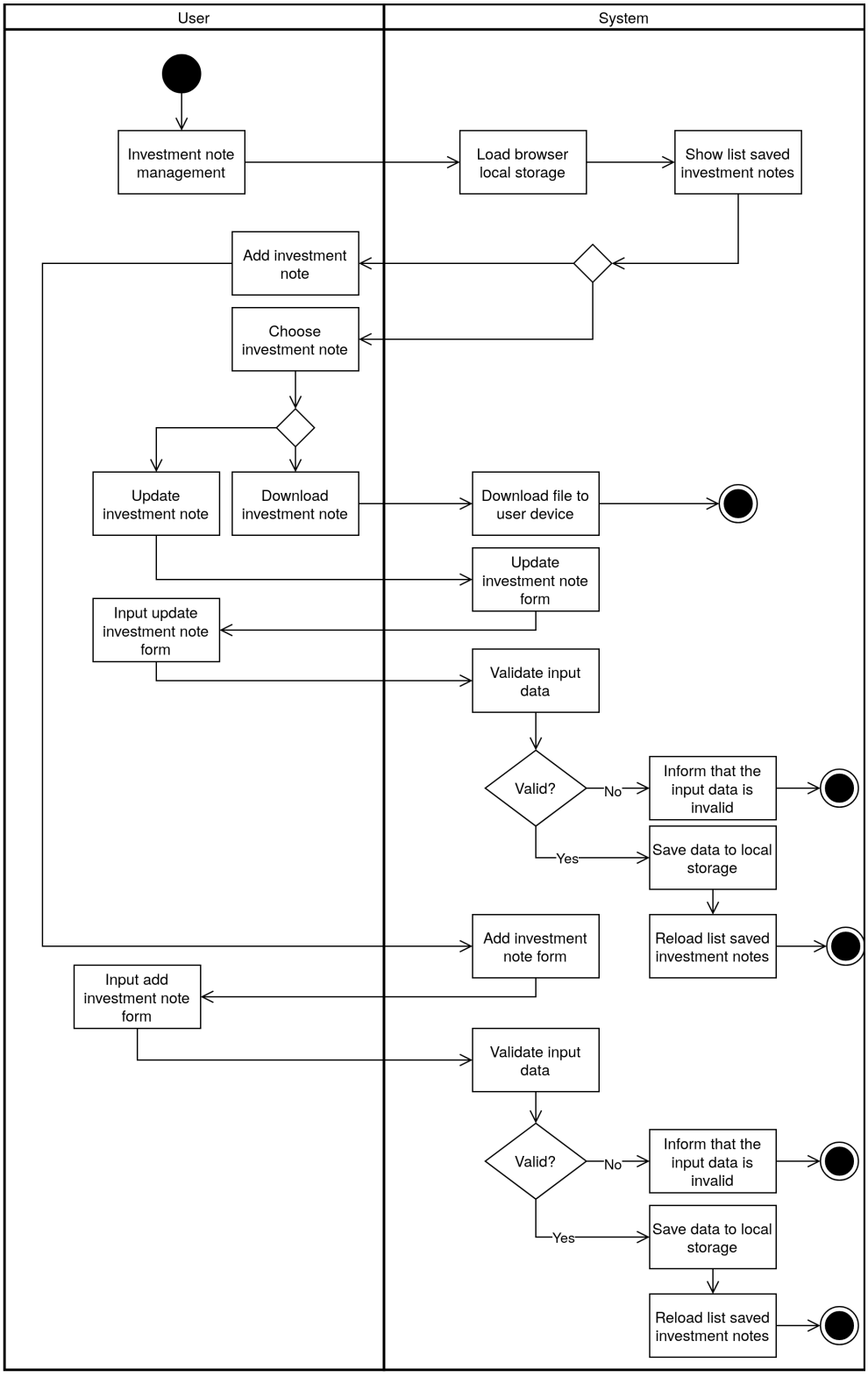

Figure 3.11: Investment note management process

The investment notes generated by the investment processes are automatically saved into the browser local storage. Users have the ability to manage these investment notes, including options to download the files for backup purposes, modify the details of the notes, and add investment notes from their files into the browser local storage. However, caution is advised when performing these actions, as the

information contained within these notes is crucial for users to have the right in the governance processes of the DAO in which they have invested.

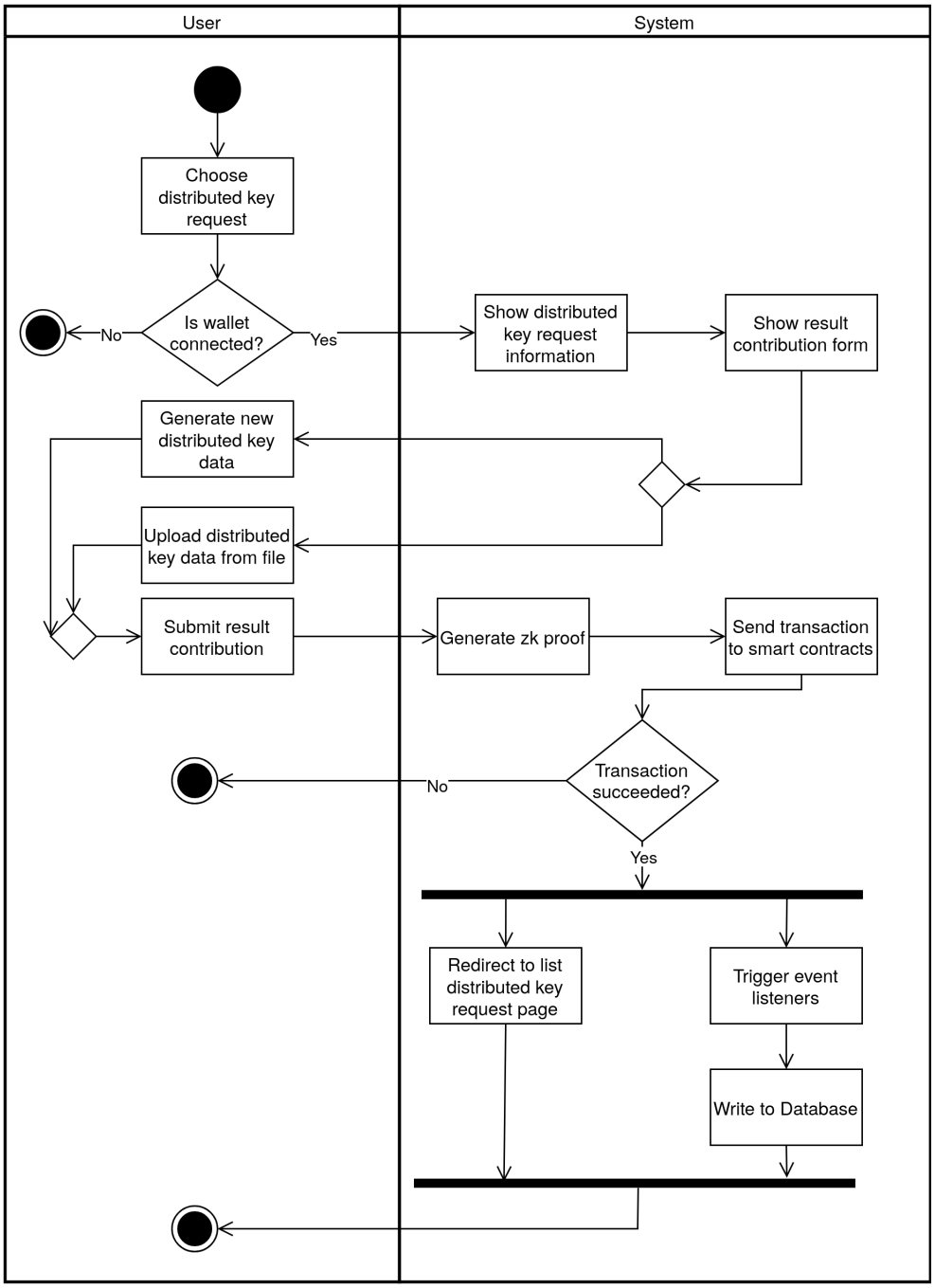

Figure 3.12: Result contribution process

Upon the completion of the investment process, the smart contract automatically generates a distributed key request, which serves as an invitation for committee members to participate in calculating the final result of the investment process. The result contribution process involves determining the allocation of funds to the DAOs that participated in the funding round. To preserve privacy and ensure data accuracy, zero knowledge proof techniques are employed. These techniques conceal sensitive information regarding investors' previous choices from exposure and

verify the correctness of the data submitted by committee members.

Similar to the public key contribution process, the result contribution process also necessitates a sufficient number of committee members to submit their contributions. Once an adequate number of contributions has been received, the final result of the investment process can be calculated successfully. Subsequently, the investment amounts will be appropriately distributed back to the corresponding DAOs. This ensures an accurate allocation of funds, reflecting the outcomes of the investment process within the Privacy Guard DAO model.

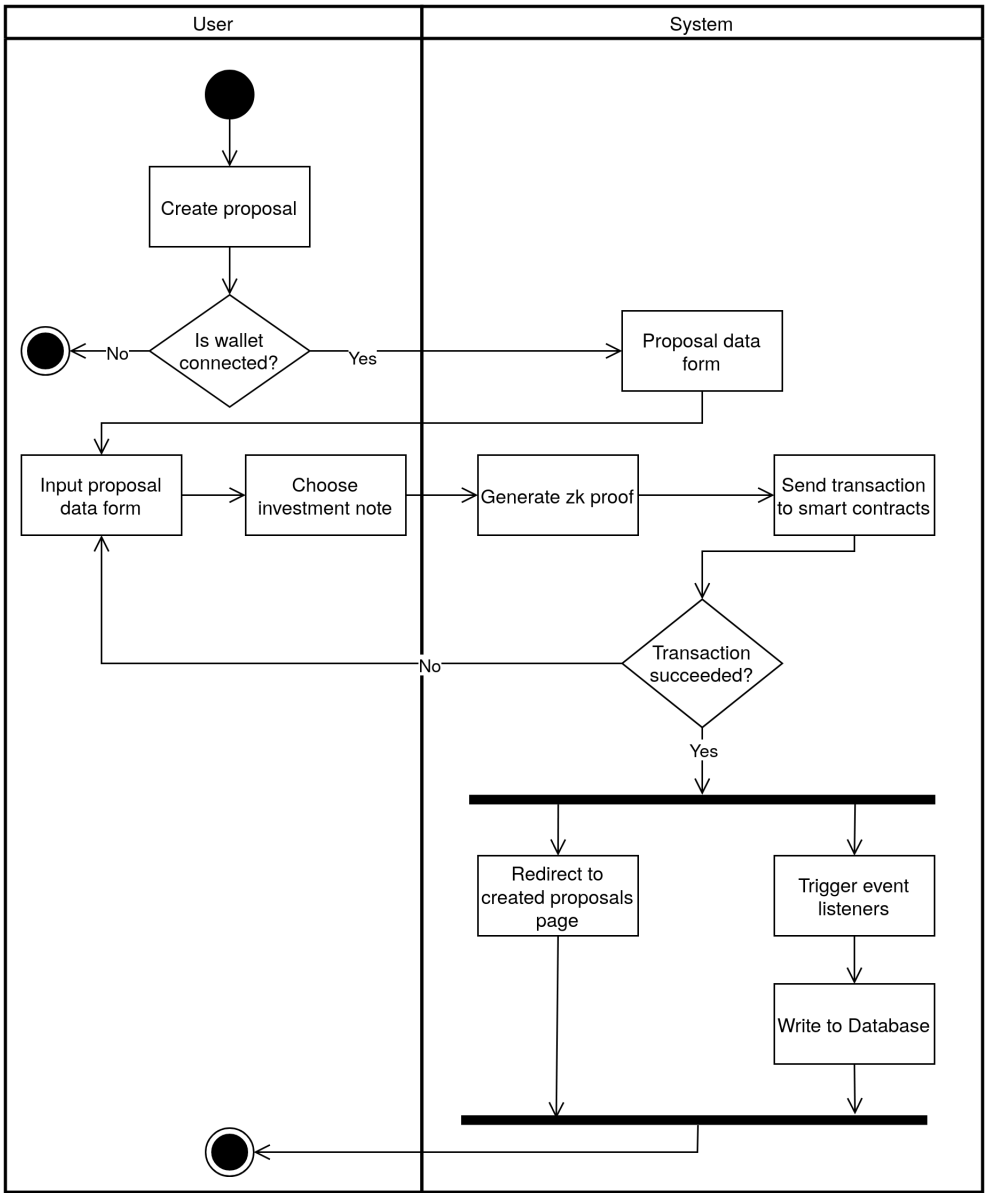

Figure 3.13: Proposal creation process

The core workflow of a DAO revolves around the "Improvement Proposal" mechanism. Participants within a DAO have the ability to suggest modifications or updates to the organization via the application client. This includes information, detail description about the proposal, and actions to be executed if the proposal

receives is approved after the voting period. And it is indispensable that users must choose an investment note, thereby generating a zero knowledge proof to prove that they are a member of the DAO.

Furthermore, DAOs that seek additional funding opportunities can participate in the funding round queue by submitting proposals and waiting for approval. This mechanism allows DAOs to continue raising funds and engage with potential investors within the Privacy Guard DAO model.

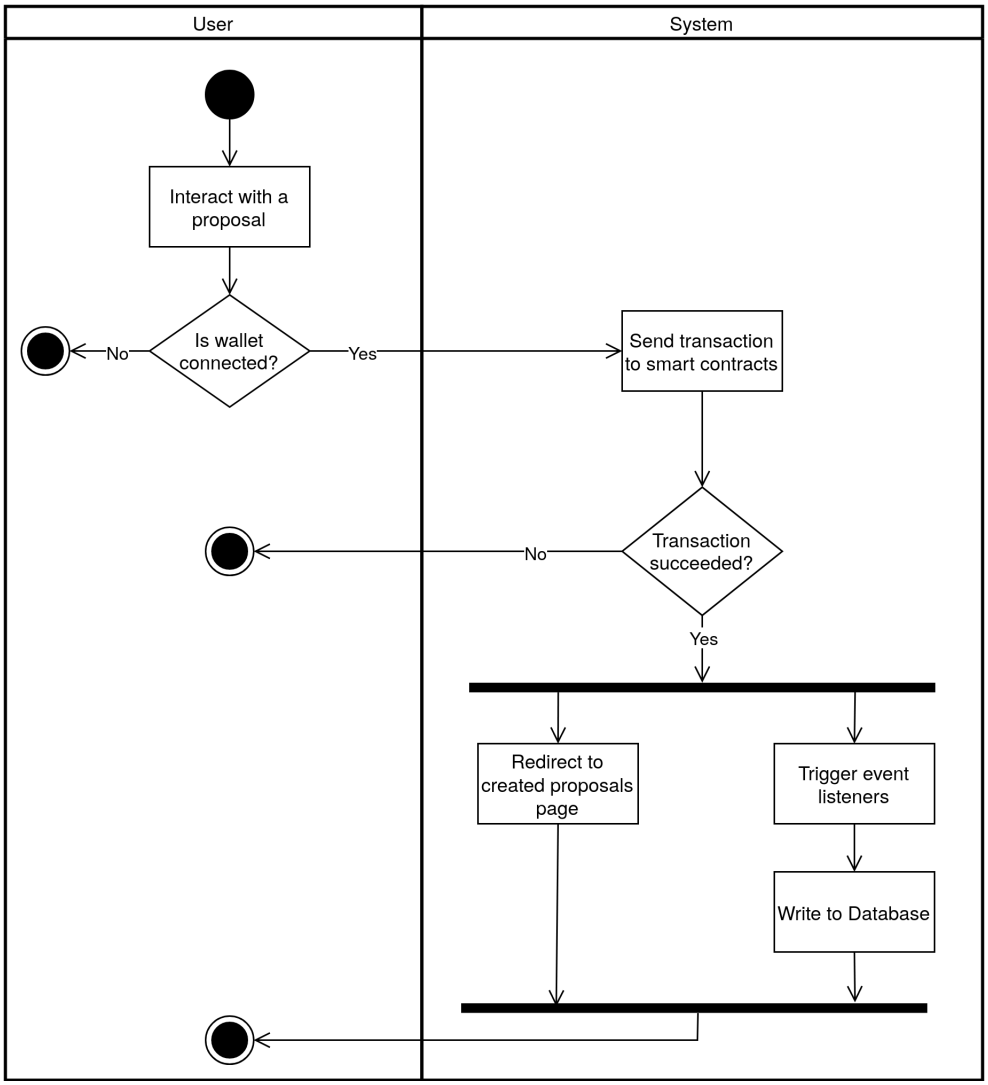

Figure 3.14: Proposal interaction process

Once a proposal for the organization's improvement was proposed and open to the public, anyone with interest can interact with it through the smart contract system. The lifetime of a proposal in this DAO model follows most of the existing solutions: after creation, the proposal might be delayed a few blocks until the voting period goes live; then based on the voting result, the proposal if succeeded can be queued into the timelock for delay and wait for execution call to be finalized. The available interactions with a proposal are: reading data, queuing into timelock,

calling for execution, and casting a vote.

Once a proposal for organizational improvement is proposed, individuals with interest can interact with it using the smart contract system. The lifecycle of a proposal in this DAO model follows a typical pattern: upon creation, the proposal may undergo a brief delay until the voting period commences. Based on the voting outcome, a successful proposal can be queued to the timelock, where it awaits a execution call. Interactions with a proposal include reading data, queuing it into the timelock, performing an execution call, and casting a vote.

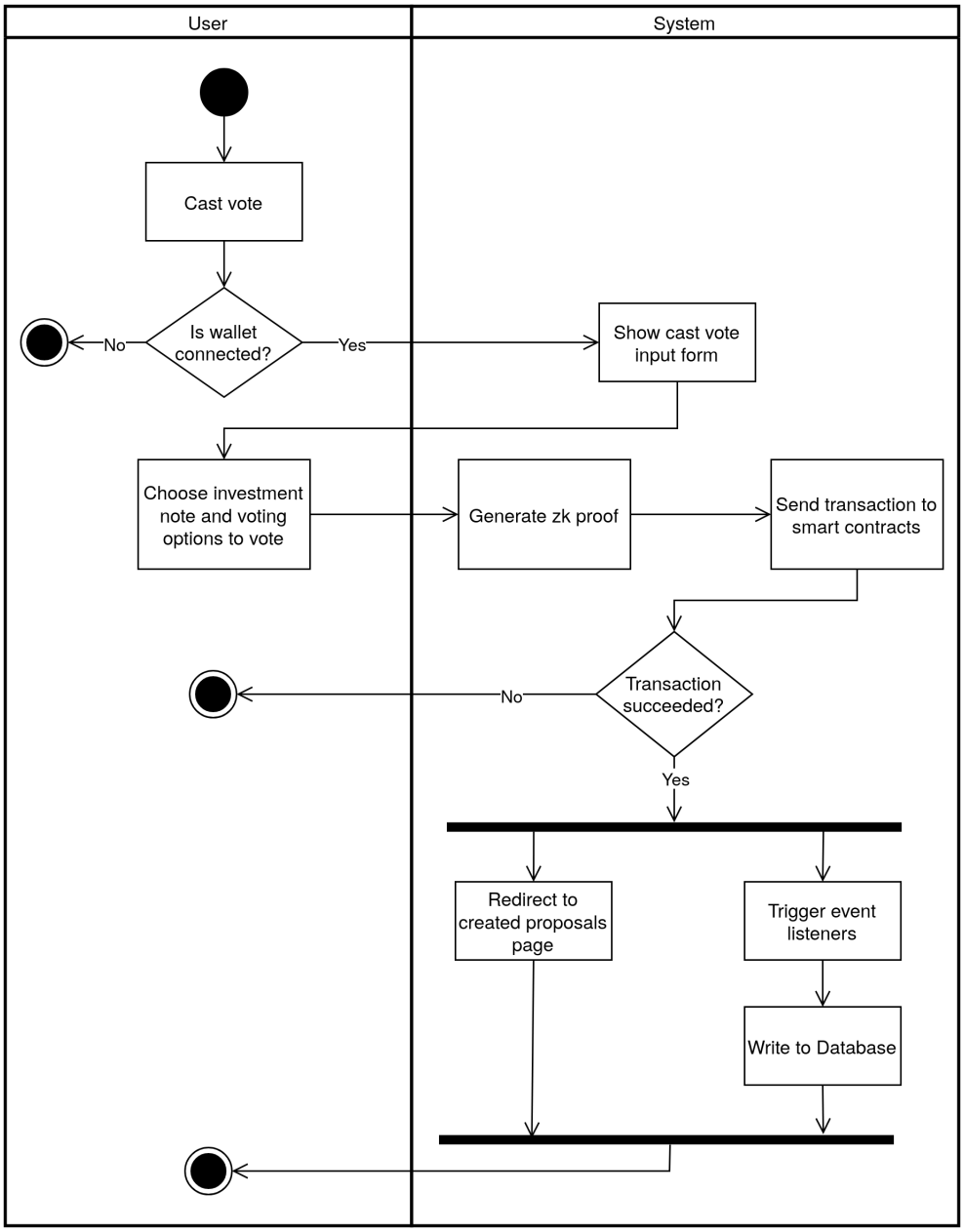

Figure 3.15: Vote casting process

Vote casting plays a pivotal role in the provided service, granting shareholders the ability to actively participate in the governance process and influence the trajectory of the DAO. Users have three options when voting on a proposal: "For"

to express support, "Against" to indicate objection, and "Abstain" for neutrality. After that, users need to generate a zero knowledge proof to prove that they have the voting power in DAO, and conceal the information about their voting options as well as their voting power. Furthermore, zero knowledge techniques also provide protection against double-voting, ensuring that users cannot exploit the same investment note to cast multiple votes.

Once the voting period concludes, a distributed key request is automatically generated by the smart contract, similar to the process that occurs after the investment process ends. The result contribution process takes place similarly, where committee members submit their contributions to determine the outcome. The result of the result contribution process will determine whether the proposal is approved or not.

3.5 Functional requirements

This section gives an overview of the functionality of the Privacy Guard DAO model.

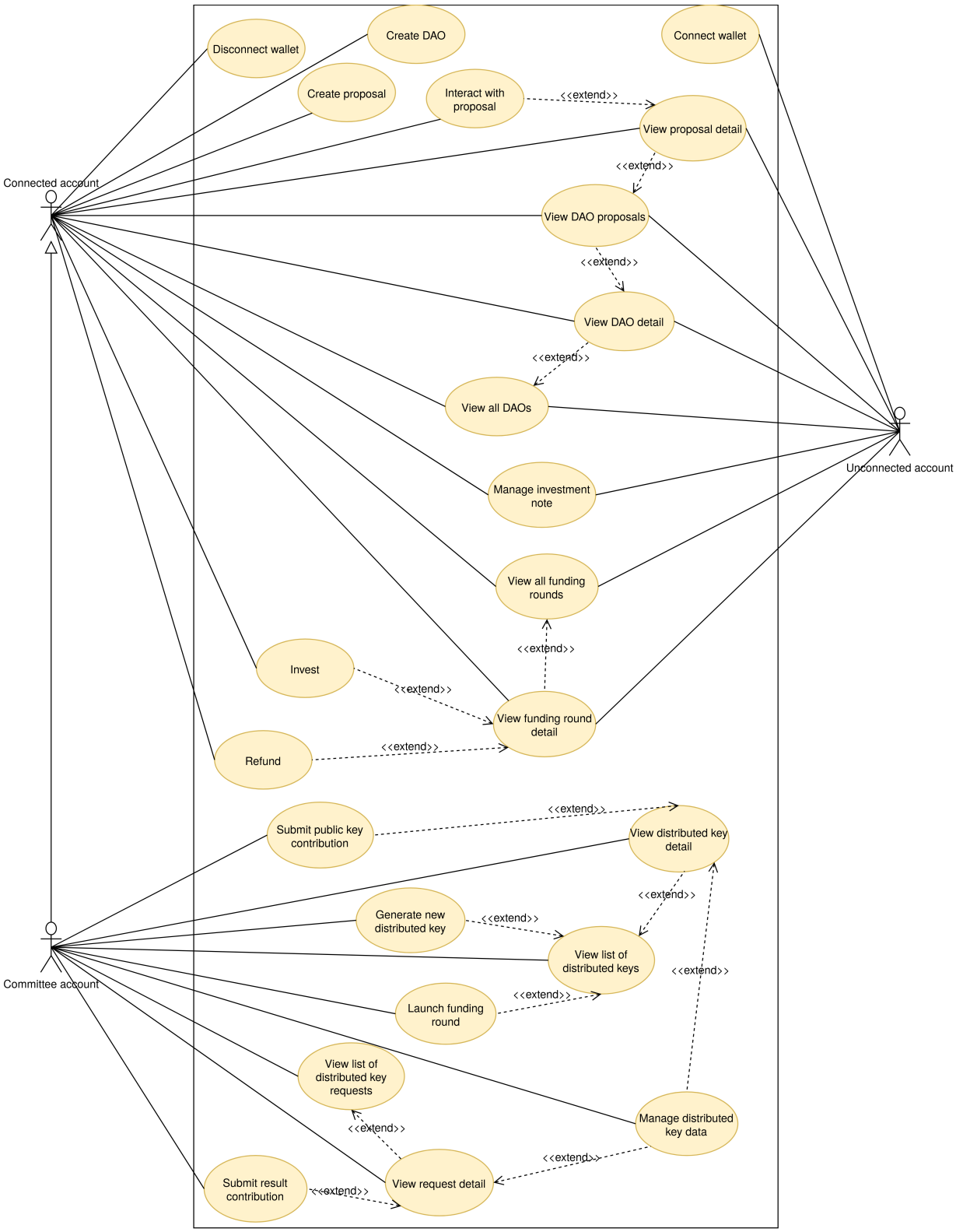

Figure 3.16: System general use case diagram

Unconnected accounts have the ability to search for DAOs, access detailed information, view existing proposals and funding rounds, as well as manage their

investment notes stored in their browser local storage. However, any non-read interactions require a wallet connection for identification purposes.

On the other hand, individuals who have connected their wallet to the client app can perform various actions such as creating a new DAO, generating proposals for existing DAOs, and interaction with the proposals.

For committee accounts within the Privacy Guard DAO model, they possess the ability to generate new distributed keys and submit contributions to these keys. It is indispensable to make result contributions to obtain the final results in the investment or governance process. In addition, they can manage distributed key data used for contribution work.

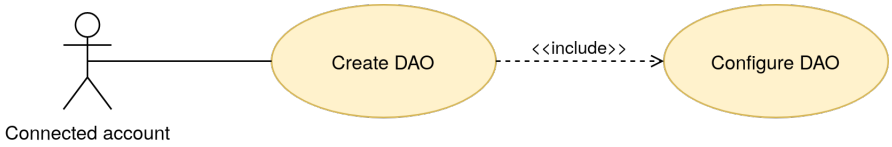

Figure 3.17: Create DAO use case decomposition

The DAO creation requires users to configure parameters and options for their DAO.

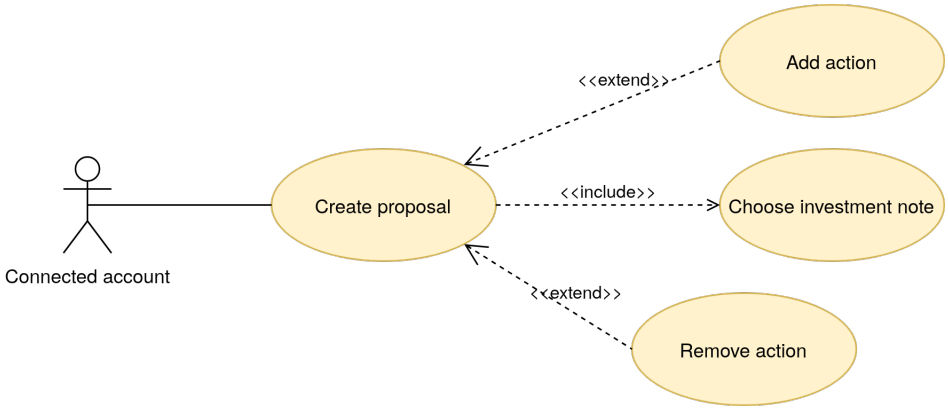

Figure 3.18: Create Proposal use case decomposition

The proposal creation is similar to the DAO creation. Instead of configurations, users have options to specify actions for the proposal to execute upon successful approval. And it is indispensable that they must choose an investment note to prove themselves a member of DAO.

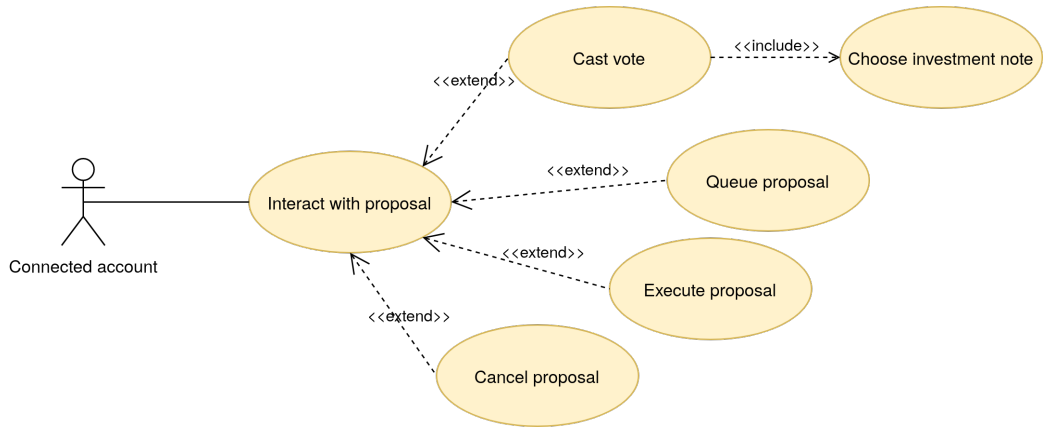

Figure 3.19: Interact with proposal use case decomposition

The proposal interaction use case has four actions that connected accounts can perform: cast a vote to voice their opinion, queue the proposal for delay then execute when allowed, and cancel the proposal. The cancel interaction is only for a limited number of users. And to ensure the legitimacy of the vote, users are required to choose an investment note as proof of their voting rights

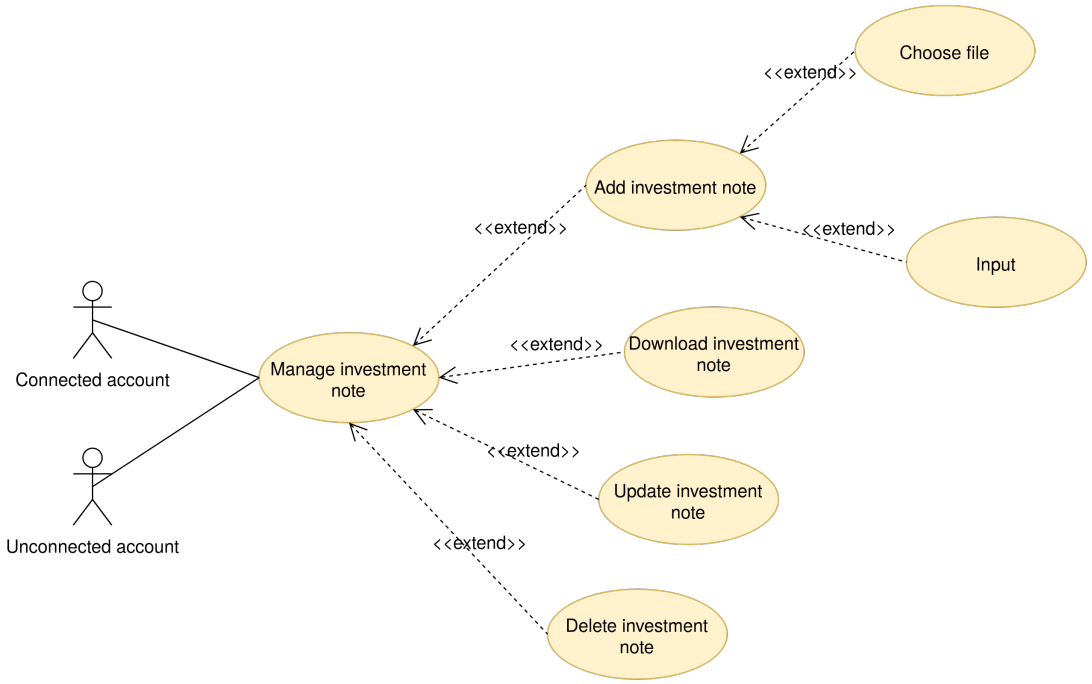

Figure 3.20: Manage investment note use case decomposition

The investment note management use case includes actions for users to manage investment notes stored in their browser local storage, as well as allowing them to download files for backup or exchange purposes.

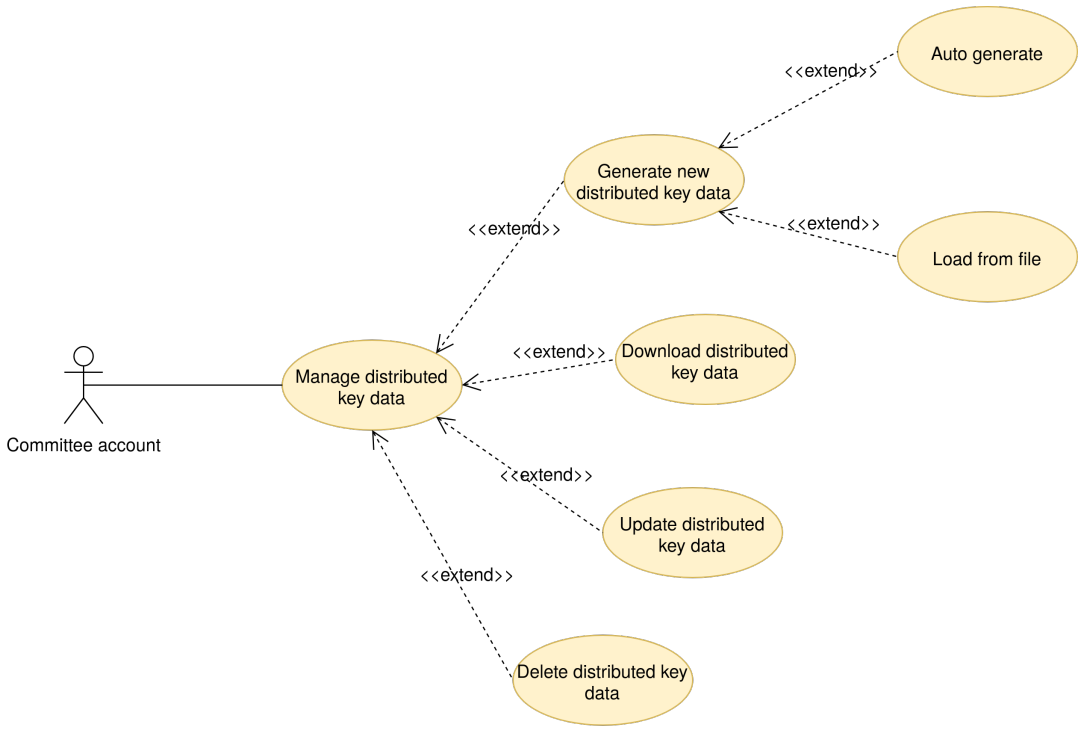

Figure 3.21: Manage distributed key use case decomposition

As mentioned above, the tasks of committee members is to submit contributions to serve the community. So they can manage the distributed key data used for contribution purposes. This management includes randomly generating new data, or selecting from files they already have, as well as allowing editing and downloading for backup or exchange purposes.

The detailed specification of the above use cases can be found in Appendix A.

3.6 Non-functional requirements

In addition to the system's functionality, several non-functional requirements are outlined in the following table, categorized and specified by their respective criteria:

Categories	**Criteria**
Security	• All smart contracts must adhere to common security practices to mitigate known exploits and vulnerabilities. • Interactions with smart contracts must incorporate role restrictions to ensure proper access control. • Testing processes should comprehensively cover the main flow of smart contracts to identify and address potential security issues. • The system must not require users' private keys in any form to prevent unauthorized access and enhance user privacy. • The provided model and smart contracts should be deployed on a testnet environment for thorough testing and auditing before deployment on the mainnet.
Transparency	• The smart contract source code must be transparent and open-source, allowing for public scrutiny, code review, and audits.
Scalability	• The model should be able to be provided on multiple blockchains that are EVM-compatible.

Usability	• Clear instructions, error messages, and informative feedback should be provided to enhance the user experience. • The user interface should be intuitive and user-friendly, ensuring ease of navigation and interaction for users of all levels of technical expertise.

Table 3.2: Non-functional requirements specification

In addition to the above non-functional requirements, a private decision-making system must meet the following requirements:

In Investment process:

- Privacy is preserved forever; e.g: designs where investor identity is exposed when calculate final result, or at any point after investment are not acceptable.
- The Investor only has to make a single transaction to make an investment.
- When the investor performs invest to a funding round, an observer cannot tell for which DAO the funds are invested.
- Impossible to calculate fund allocation result before the investment period ends.
- Avoid centralized dependencies: any off-chain computation should run open source code and allow multiple parties to participate, to optimize for censorship resistance and redundancy.

In Governance process:

- Privacy is preserved forever; e.g: designs where voter identity is exposed when counting votes, or at any point post vote casting are not acceptable.
- The voter only has to make a single transaction to make a vote casting.
- When the voter performs a vote casting, an observer cannot tell which options chosen by the voter, and how much voting power of the voter.
- Impossible to tally a proposal's votes before the voting period ends.

- Avoid centralized dependencies: any off-chain computation should run open source code and allow multiple parties to participate, to optimize for censorship resistance and redundancy.

4.1 Algorithm design

4.1.1 Initial settings

In the settings of the Privacy Guard DAO model, the chosen elliptic curve is Baby Jubjub [21], and the corresponding prime p aligns with the suborder of the Baby Jubjub curve. The variables n_c and n_u indicate the number of members in the committee and the number of users participating in investment or governance processes, respectively. For cryptographic purposes, the Pedersen Hash function [22] is utilized. Additionally, a Merkle tree is initialized and utilized across the investment and governance processes, with the Poseidon hash function [23], [24] applied.

4.1.2 Public key contribution process

The public key contribution process is based on Sel phase and Dis phase of the (t, n_c) DKG scheme. This process can be divided into two rounds:

Round 1:

1. Each committee member i chooses t values $(a_{i,0}, a_{i,1}, \ldots, a_{i,t-1}) \leftarrow Z_p$ at random. Then a polynomial of degree at most $t-1$ is defined as:

$$f_i(x) = \sum_{j=0}^{t-1} a_{i,j} \cdot x^j$$

2. Each committee member i computes a public commitment

$$C_i = (C_{i,0}, C_{i,1}, \ldots, C_{i,t-1})$$

where

$$C_{i,j} = a_{i,j} \cdot G \quad \text{for } j = 0 \ldots t-1,$$

and sends this public commitment to smart contract. The smart contract verifies that all $C_{i,j}$ are points on Baby Jubjub curve.

3. After n_c committee members submit their commitments and they are verified as valid points on the Baby Jubjub curve, the smart contract automatically computes the public key $PK = \sum_{i=1}^{n_c} C_{i,0}$.

Round 2:

1. Each committee member i computes $f_i(j)$ for $j = 1, 2, \ldots, n_c$
2. Each committee member i keeps $(i, f_i(i))$ as their secret and uses $C_{i,0}$ as their public key for data exchange.
3. Committee member i perform asymmetric encryption on $f_i(j)$ using $C_{j,0}$ to obtain $ENC(f_i(j), C_{j,0})$. Then, committee member i generates a ZKP to prove that:
   - $f_i(j) \cdot G = \sum_{k=0}^{t-1} C_{i,k} \cdot j^k$
   - $ENC(f_i(j), C_{j,0})$ is the result of asymmetric encryption on $f_i(j)$ using the public key $C_{j,0}$, ensuring that committee member j can decrypt by private key $a_{j,0}$.
4. Each committee member i sends all $ENC(f_i(j), C_{j,0})$ for $j = 1, 2, \ldots, n_c$, where $j \neq i$, along with the corresponding ZKP to the smart contract. Along with the stored values $C_{i,j}$, the smart contract is responsible for validating the validity of $ENC(f_i(j), C_{j,0})$ and the accompanying ZKP.
5. After all committee members send valid data to the smart contract, committee member i performs decryption using their private key $a_{i,0}$ and calculations to obtain their secret $sk_i = \sum_{j=1}^{n_c} f_j(i)$.

After Round 1 (in the public key contribution process), the public key PK is computed and can be used for investment or governance processes.

4.1.3 Investment process

During the investment process, investors only need to send one transaction to make an investment. For each funding round, let's assume there are m DAOs allowed to invest. The smart contract determines the order of these DAOs and represents them as a vector, denoted by $\mathbf{id} = (id_1, id_2, \ldots, id_m)$. Here, id_i represents the ID of each DAO for $i = 1, 2, \ldots, m$. Additionally, the smart contract initializes a total investment amount variable, v_{total}, to zero. Furthermore, two vectors $(\mathbf{R}, \mathbf{M}) \in (\mathbb{G}^m, \mathbb{G}^m)$, are initialized as follows:

$$\mathbf{R} = (\mathcal{O}, \mathcal{O}, \cdots, \mathcal{O})$$
$$\mathbf{M} = (\mathcal{O}, \mathcal{O}, \cdots, \mathcal{O})$$

When an investor, let's say Investor i, wants to invest v_i in the DAO with ID id_{DAO}, the index of id_{DAO} in the vector $\mathbf{id}$ is denoted by k. Additionally, a one-hot vector $\mathbf{o_i}$ is used to represent the index k of id_{DAO} in the vector $\mathbf{id}$.

The process for investment is as follows:

1. The smart contract stores publicly accessible values: PK, m, v_{total}, $\mathbf{id}$, $\mathbf{R}$ and $\mathbf{M}$. Additionally, v_i represents the publicly known amount sent by an investor along with their investment transaction to the smart contract.
2. Investor i randomly chooses a vector $\mathbf{r_i} \in Z_p^m$. They then compute $\mathbf{R_i}$ and $\mathbf{M_i}$ as follows:

$$\mathbf{R_i} = \mathbf{r_i} \cdot G$$
$$\mathbf{M_i} = (\mathbf{o_i} \times v_i) \cdot G + \mathbf{r_i} \cdot PK$$

3. Investor i randomly selects a nullifier $null$ and computes a commitment C_i as follows:

$$C_i = H(null, id_{DAO}, v_i)$$

4. Investor i keeps id_{DAO}, $null$, $\mathbf{o_i}$, and $\mathbf{r_i}$ private, securely storing them. They then send $\mathbf{R_i}$, $\mathbf{M_i}$, C_i, and a ZKP to the smart contract. The ZKP is used to verify the following:
   - $\langle id, \mathbf{o_i}^T \rangle = id_{DAO}$ and $C_i = H(null, \langle id, \mathbf{o_i}^T \rangle, v_i)$
   - The same $\mathbf{r_i}$ is used in the computation of $\mathbf{R_i}$ and $\mathbf{M_i}$.
   - In the computation of $\mathbf{M_i}$, the value of PK matches the value stored in the smart contract, the value of v_i matches the amount sent along with the investment transaction, and the value of $\mathbf{o_i}$ corresponds to the value used in the computation of C_i.
5. If the smart contract validates the transaction and the ZKP sent with it is valid, the smart contract will perform computations as follows:
   - Accumulate the sum:

$$v_{\text{total}} = v_{\text{total}} + v_i$$
$$\mathbf{R} = \mathbf{R} + \mathbf{R_i}$$
$$\mathbf{M} = \mathbf{M} + \mathbf{M_i}$$

   - Insert commitment C_i to the Merkle Tree and compute the new root.

Let's assume there are n_u investors participating in the funding round. We can

denote the aggregated values as $\mathbf{r} = \sum_{i=1}^{n_u} \mathbf{r_i}$ and $\mathbf{v} = \sum_{i=1}^{n_u} v_i \times \mathbf{o_i}$. This means that $\mathbf{R} = \mathbf{r} \cdot G \in \mathbb{G}^m$ and $\mathbf{M} = \mathbf{v} \cdot G + \mathbf{r} \cdot PK \in \mathbb{G}^m$.

At the conclusion of the result contribution process, the value of $\mathbf{v}$ will be revealed as the final result.

4.1.4 Governance process

Similar to the investment process, casting a vote in the governance process only requires investors to send a single transaction. Let's assume that the DAO with ID id_{DAO} opens voting to approve the proposal with ID $id_{proposal}$. Investors have three options when voting on a proposal: "For" to express support, "Against" to indicate objection, and "Abstain" for neutrality. For each proposal, the smart contract need to initialize two vectors $(\mathbf{R}, \mathbf{M}) \in (\mathbb{G}^3, \mathbb{G}^3)$ as follows:

$$\mathbf{R} = (\mathcal{O}, \mathcal{O}, \mathcal{O})$$
$$\mathbf{M} = (\mathcal{O}, \mathcal{O}, \mathcal{O})$$

In previous funding rounds, let's say Investor i invested an amount v_i for the DAO id_{DAO}. As long as investor i still has stored the nullifier $null$ used for that investment, they can vote on proposal $id_{proposal}$. Additionally, a one-hot vector $\mathbf{o_i}$ is used to represent their choice, k is the index of coordinate with value 1 in $\mathbf{o_i}$.

The process for vote casting is as follows:

1. The smart contract stores publicly accessible values: PK, id_{DAO}, $id_{proposal}$, $\mathbf{R}$ and $\mathbf{M}$.
2. Investor i randomly chooses a vector $\mathbf{r_i} \in Z_p^m$. They then compute $\mathbf{R_i}$ and $\mathbf{M_i}$ as follows:

$$\mathbf{R_i} = \mathbf{r_i} \cdot G$$
$$\mathbf{M_i} = (\mathbf{o_i} \times v_i) \cdot G + \mathbf{r_i} \cdot PK$$

3. Investor i uses stored $null$ value to compute nullifier hash $nullhash$ as follows:

$$nullhash = H(null, id_{proposal})$$

   $nullhash$ is used to prevent double-voting.

4. Investor i sends $\mathbf{R_i}$, $\mathbf{M_i}$, $nullhash$ and a ZKP to the smart contract for vote casting. The ZKP is used to verify the following:

- Commitment $C_i = H(null, id_{DAO}, v_i)$ is in a valid path to the root of the Merkle Tree stored in the smart contract.
- The same $\mathbf{r_i}$ is used in the computation of $\mathbf{R_i}$ and $\mathbf{M_i}$.
- In the computation of $nullhash$, the value of $id_{proposal}$ matches the value stored in the smart contract, the value of $null$ matches the value used in the path validation of commitment C_i.
- In the computation of $\mathbf{M_i}$, the value of PK matches the value stored in the smart contract, and the value of v_i matches the value used in the path validation of commitment C_i.

5. If the smart contract validates the transaction and the ZKP sent with it is valid, the smart contract will perform computations as follows:
   - Accumulate the sum:

$$\mathbf{R} = \mathbf{R} + \mathbf{R_i}$$
$$\mathbf{M} = \mathbf{M} + \mathbf{M_i}$$

   - Mark $nullhash$ already used to vote on proposal $id_{proposal}$ to prevent double-voting.

Let's assume there are n_u investors participating in the funding round. We can denote the aggregated values as $\mathbf{r} = \sum_{i=1}^{n_u} \mathbf{r_i}$ and $\mathbf{v} = \sum_{i=1}^{n_u} v_i \times \mathbf{o_i}$. This means that $\mathbf{R} = \mathbf{r} \cdot G \in \mathbb{G}^m$ and $\mathbf{M} = \mathbf{v} \cdot G + \mathbf{r} \cdot PK \in \mathbb{G}^m$.

At the conclusion of the result contribution process, the value of $\mathbf{v}$ will be revealed as the final result.

4.1.5 Result contribution process

At the end of the investment or governance process, two vectors $\mathbf{R}$ and $\mathbf{M}$ will be stored and publicly accessible in the smart contract. The result contribution process is based on Rec phase of the (t, n_c) DKG scheme. This process requires the participation of t committee members and can be divided into two phases: tally contribution and final result contribution.

Let H be the index set of committee members participating in the process. H contains unique values of i ranging from 1 to n, with $|H| = t$.

The tally contribution proceeds as follows:

1. The index i of each committee member participating in the tally contribution

is publicly accessible.

2. Each committee member i in H computes $\mathbf{D_i}$ as:

$$\mathbf{D_i} = sk_i \cdot \mathbf{R}$$

3. Each committee member i in H sends $\mathbf{D_i}$ along with a ZKP to the smart contract. The ZKP is used to verify the following:

   - The validity of the decryption of all $ENC(f_j(i), C_{i,0})$ for $j = 1, 2, \ldots, n_c$, where $j \neq i$. It is verified as:

$$f_j(i) = DEC(ENC(f_j(i), C_{i,0}), a_{i,0})$$

     The private key $a_{i,0}$ is a pair with the public key $C_{i,0}$ that committee member i sent to the smart contract in round 1 of the public key contribution process.

   - The validity of index i with its corresponding $f_i(i)$, as follows:

$$f_i(i) \cdot G = \sum_{k=0}^{t-1} C_{i,k} \cdot i^k$$

   - The correctness of sk_i used in the computation of $\mathbf{D_i}$, which is calculated as:

$$sk_i = \sum_{j=1}^{n_c} f_j(i)$$

   - The matching of $\mathbf{R}$ used in the computation of $\mathbf{D_i}$ with the $\mathbf{R}$ stored in the smart contract.

4. Once all members in H have sent $\mathbf{D_i}$, the smart contract performs the following computations:

   - $\mathbf{D} = \sum_{i \in H} \lambda_i \cdot \mathbf{D_i}$, where $\lambda_i = \prod_{i \in H, j \neq i} \frac{j}{j-i}$ is the Lagrange coefficient.

   - $\mathbf{v} \cdot G = \mathbf{M} - \mathbf{D}$, where $\mathbf{v}$ is the vector representing the final result to be determined. In the case of the investment process, the coordinates will indicate the amount invested in each DAO. In the case of the governance process, the coordinates will indicate the voting power of each option.

The purpose of the final result contribution process is to solve for $\mathbf{v}$ in the equa-

tion $\mathbf{v} \cdot G$. In the case of the investment process, we have an additional parameter, v_{total}, which represents the total amount invested. In the context of the Privacy Guard DAO model, v_{total} is not excessively large, allowing us to brute-force $\mathbf{v} \cdot G$ in order to obtain $\mathbf{v}$. Once we have solved for $\mathbf{v}$, we send it to the smart contract along with a ZKP to verify that $\mathbf{v}$ satisfies the equation $\mathbf{v} \cdot G$ stored in the smart contract.

4.2 System design

4.2.1 Overall system architecture

In accordance with the system requirements, a system design is proposed for the infrastructure of the Privacy Guard DAO model. The following diagram illustrates the primary components of the overall architecture and their interrelationships.

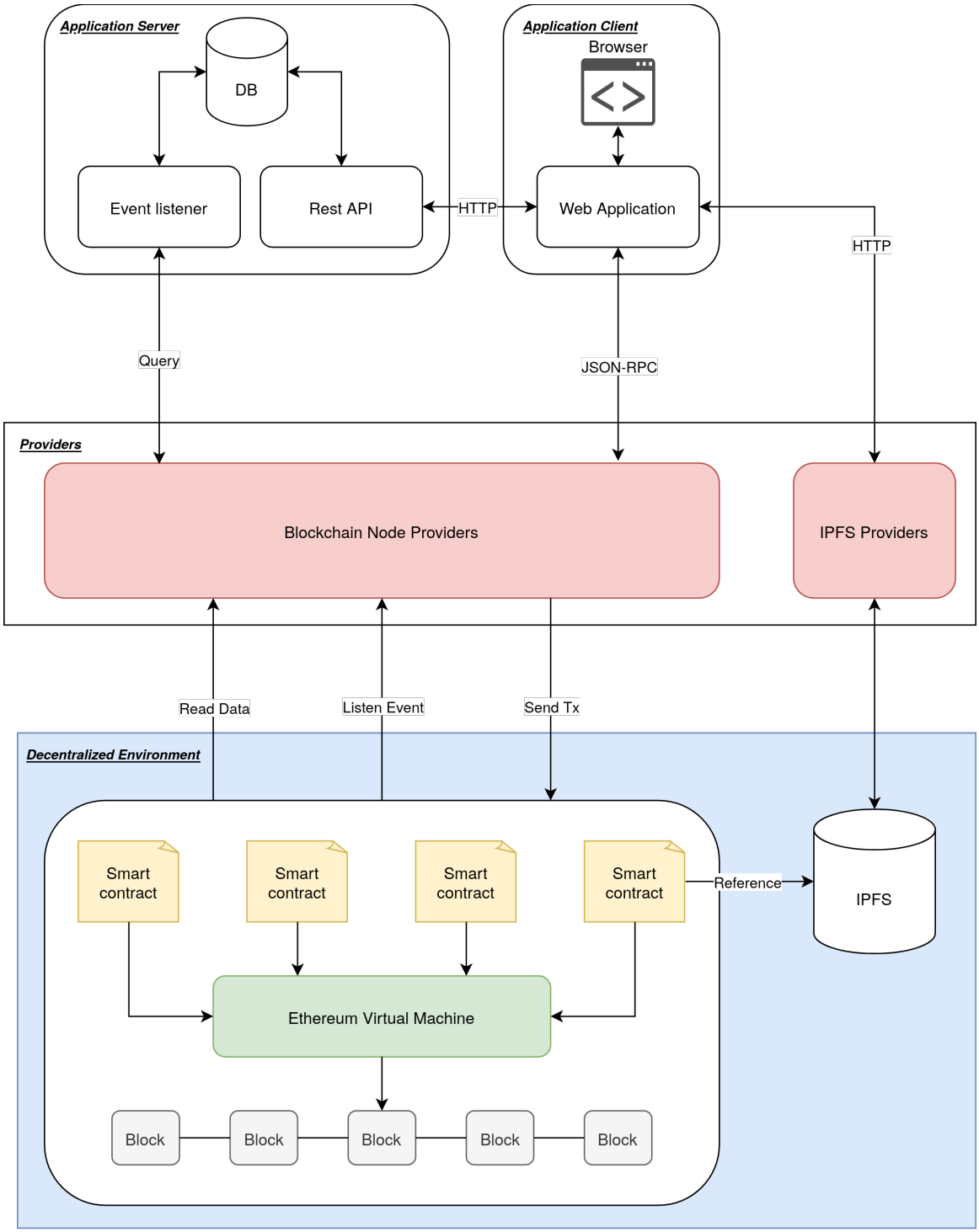

Figure 4.1: Overall system architecture

Four key components are presented in the diagram: decentralized environments encompassing EVM-compatible blockchains and IPFS, providers serving as in-

terfaces between the application layers and decentralized environments, the application server for collecting and combining data from blockchains to optimize responses for data requests from the application client, and the application client for user interactions. In the subsequent sections, a thorough examination of each component's design and implementation will be presented.

4.2.2 Decentralized environments

In the Privacy Guard DAO model design, decentralized environments includes EVM-compatible blockchains and decentralized storage services - IPFS, with smart contracts playing a center role in the system. This smart contract system implements zero knowledge proof verification mechanism. This mechanism aim to conceal the information for the user information while upholding the integrity and accuracy of the data submitted by the users. Additionally, the model incorporates a configurable governance process for DAOs.

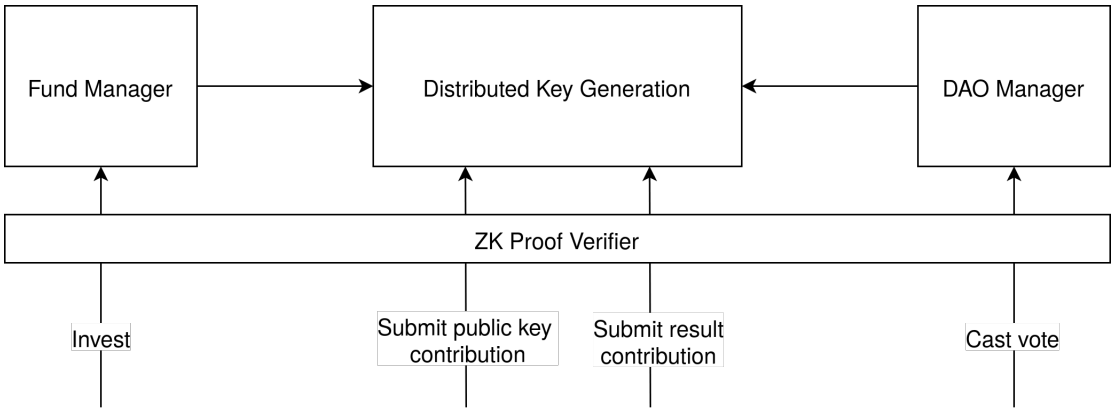

Figure 4.2: Smart contract system

The Smart contract system of the Privacy Guard DAO model consist of four components

ZK Proof Verifier comprises smart contracts that implement the mechanism for verifying zero-knowledge proofs. These contracts are the foundation for "private" in the Privacy Guard DAO model. In all processes within the model that require the privacy, users are required to generate zero-knowledge proofs to accompany their transactions. The ZK Proof Verifier contracts utilize these proofs to validate the accuracy of the data submitted by users and prevent any fraudulent activities.

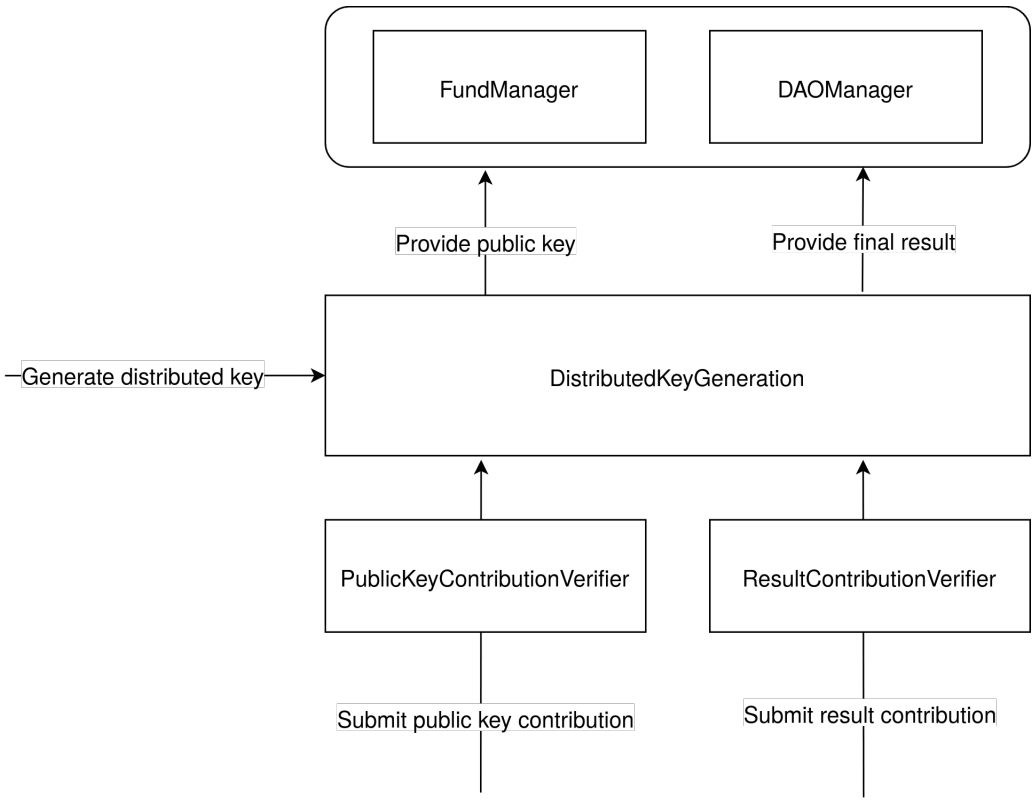

Figure 4.3: Distributed Key Generation smart contract module

Distributed Key Generation contract is responsible for committee members to generate distributed key, submit contribution for establishing public key, as well as contribution to obtaining final result of investment or governance processes.

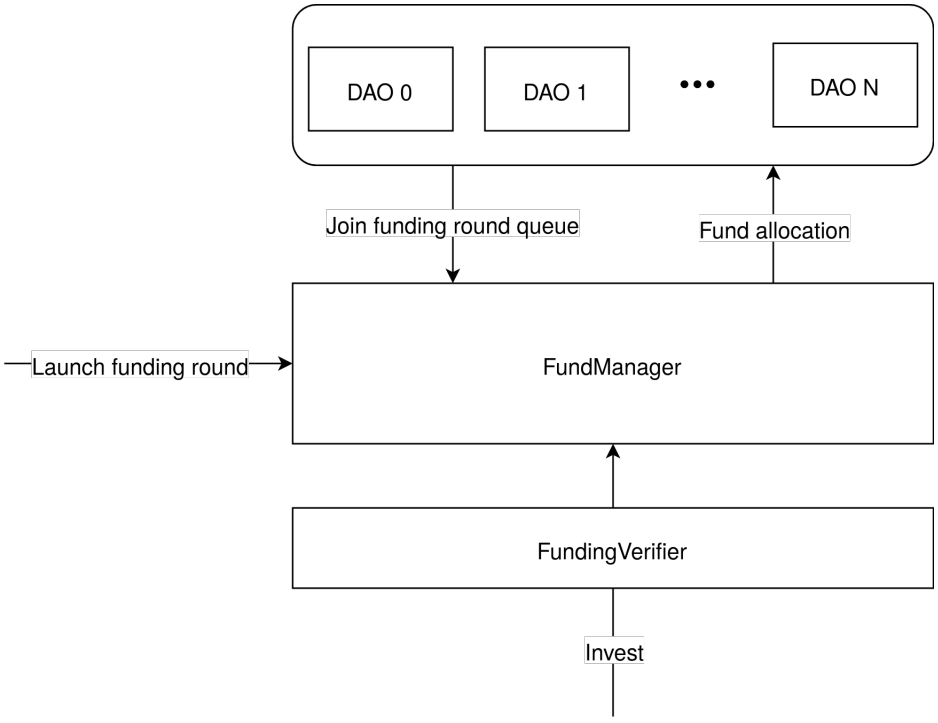

Figure 4.4: Fund Manager smart contract module

Fund Manager contract serves as a platform for funding rounds. This smart contract will receive fund from investors, and then when the funding round ends, this contract will perform fund allocation to DAOs that participated in funding rounds. It also manages a funding round queue, where DAOs wait for their turn to receive investments.

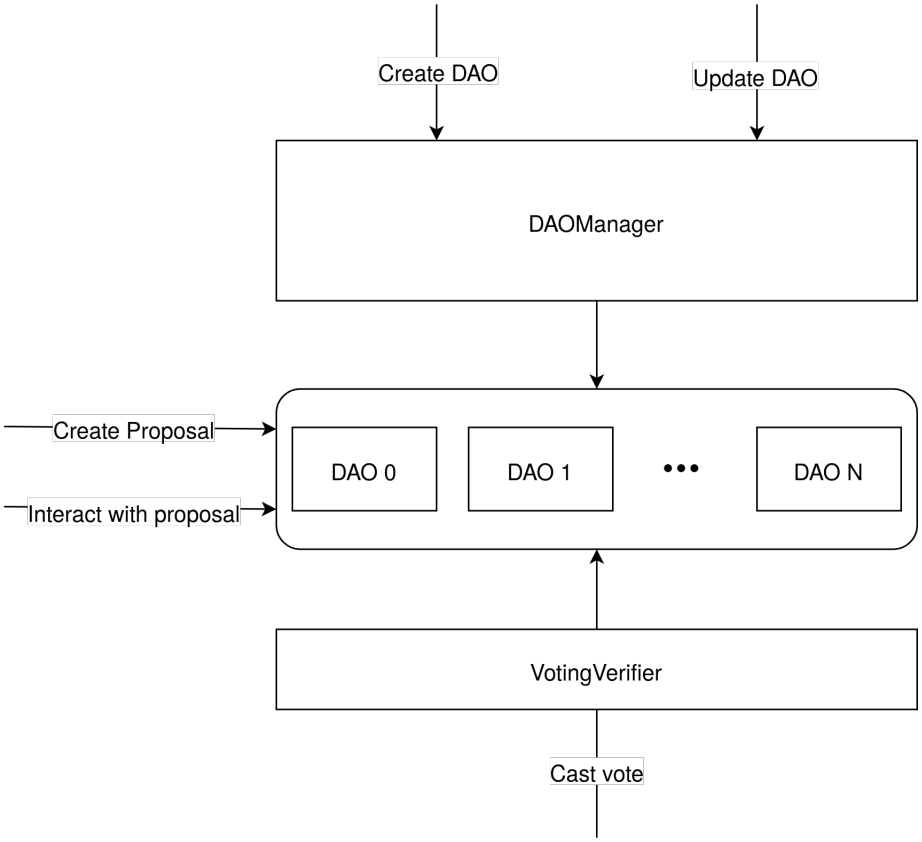

Figure 4.5: DAO Manager smart contract module

DAO Manager consist of smart contracts have responsible for create and update, as well as record all existed DAOs in the Privacy Guard DAO model. Additionally, these smart contracts implement voting and execution mechanism for governance processes.

In order to implement the proposed system architecture design, class diagrams are utilized to model the structure, storage, and interface of the smart contract system. In the context of modeling, smart contracts can be regarded as analogous to classes in OOP, contracts can have relationships such as inheritance, association, composition, and aggregation, based on the "is-a," "has-a," and "part-of" concepts. The structure class diagram focuses on representing contracts by name to illustrate the relationships among them within the system. The class diagrams in this thesis adhere to the UML 2.0 standard [25] for notation.

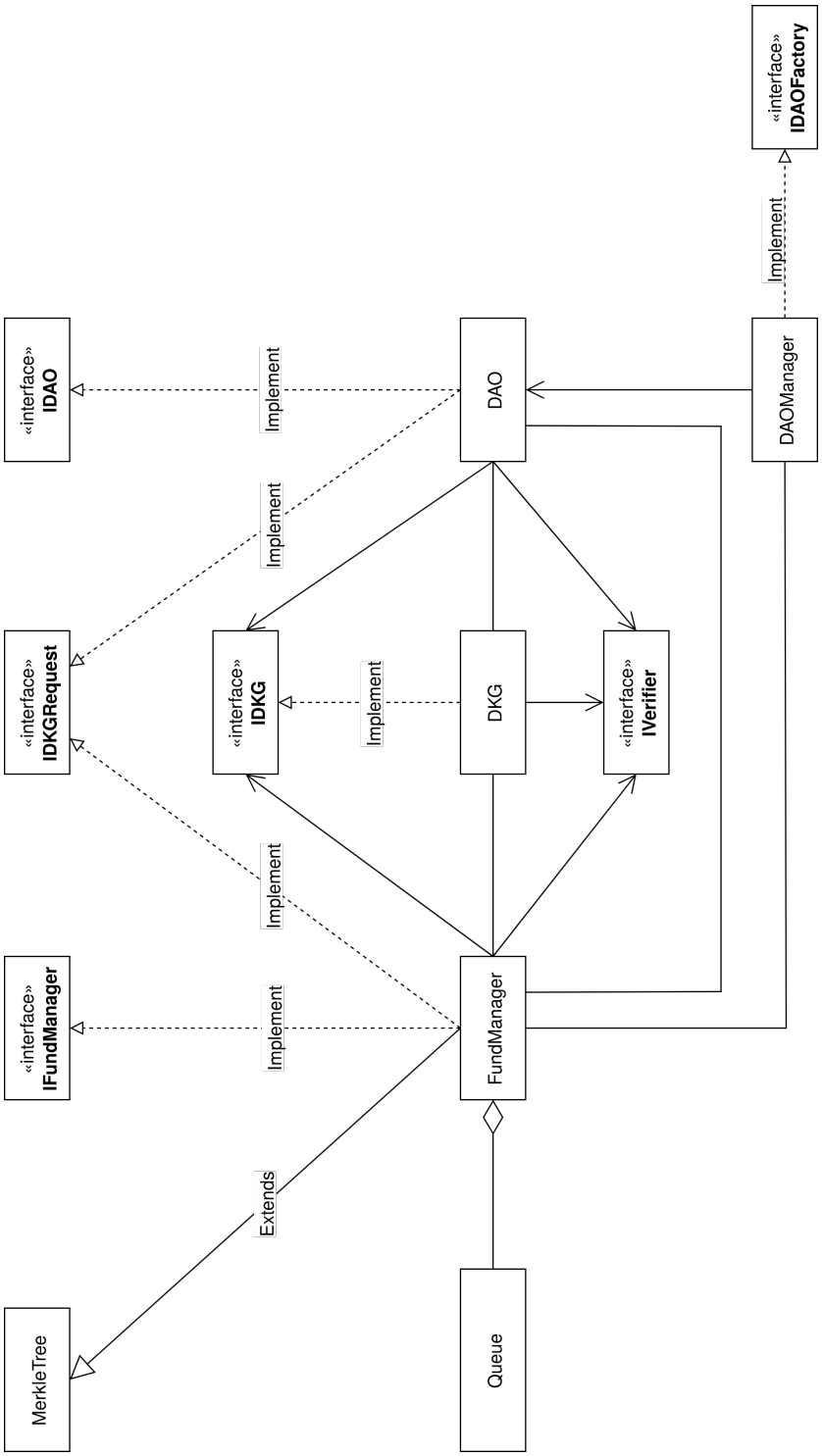

Figure 4.6: Structure class diagram

In a smart contract, data is stored using storage variables, which are analogous to attributes in a class. These storage variables can have three types of access modifiers: private, internal, and public. Private variables are accessible only within the contract itself, while internal variables can be accessed within the contract and by inherited contracts. On the other hand, public variables can be accessed both internally and externally to the contract.

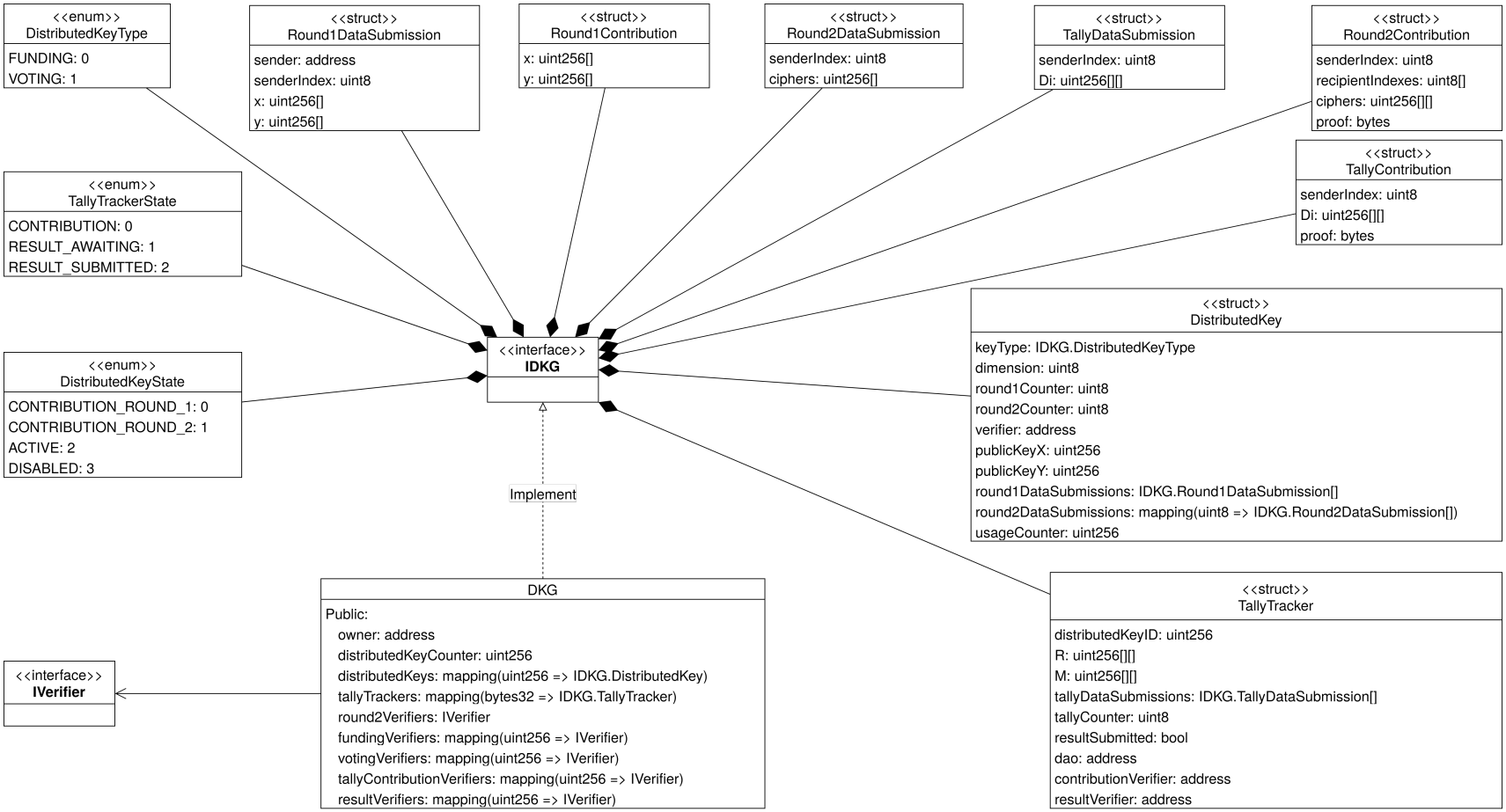

Figure 4.7: Storage class diagram - Distributed Key Generation

Contract DKG is a crucial component in the Privacy Guard DAO model, providing distributed key to achieve privacy in both investment and governance processes. The contract DKG is owned by the FundManager contract, and certain operations within DKG require permission from FundManager. DKG encompasses verifiers that play a role in validating ZKPs for various processes, such as public key contribution, result contribution, investment, and governance. It also serves as a repository for the generated distributed keys and handles requests for final results from investment and governance processes. The interface IDKG defines the data structure utilized by the DKG contract and provides an access interface for smart contracts to utilize the services provided by DKG.

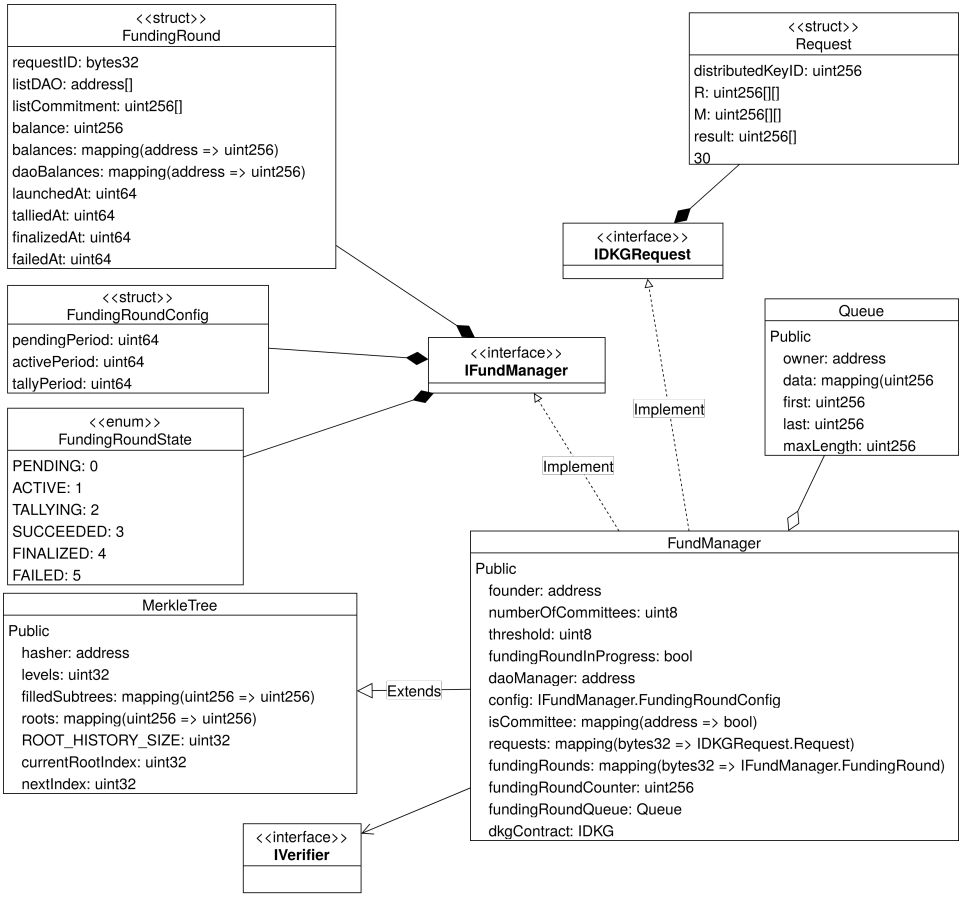

Figure 4.8: Storage class diagram - Fund Manager

Contract FundManager is a pivotal component responsible for managing investment processes within the Privacy Guard DAO model. It inherits the MerkleTree contract, which employs a Merkle tree data structure to maintain a record of user investment history. FundManager also includes a queue mechanism to determine the sequence in which DAOs participate in funding rounds. As privacy in investment processes relies on distributed keys, FundManager implements the IDKGRequest interface to track completed and ongoing funding rounds.

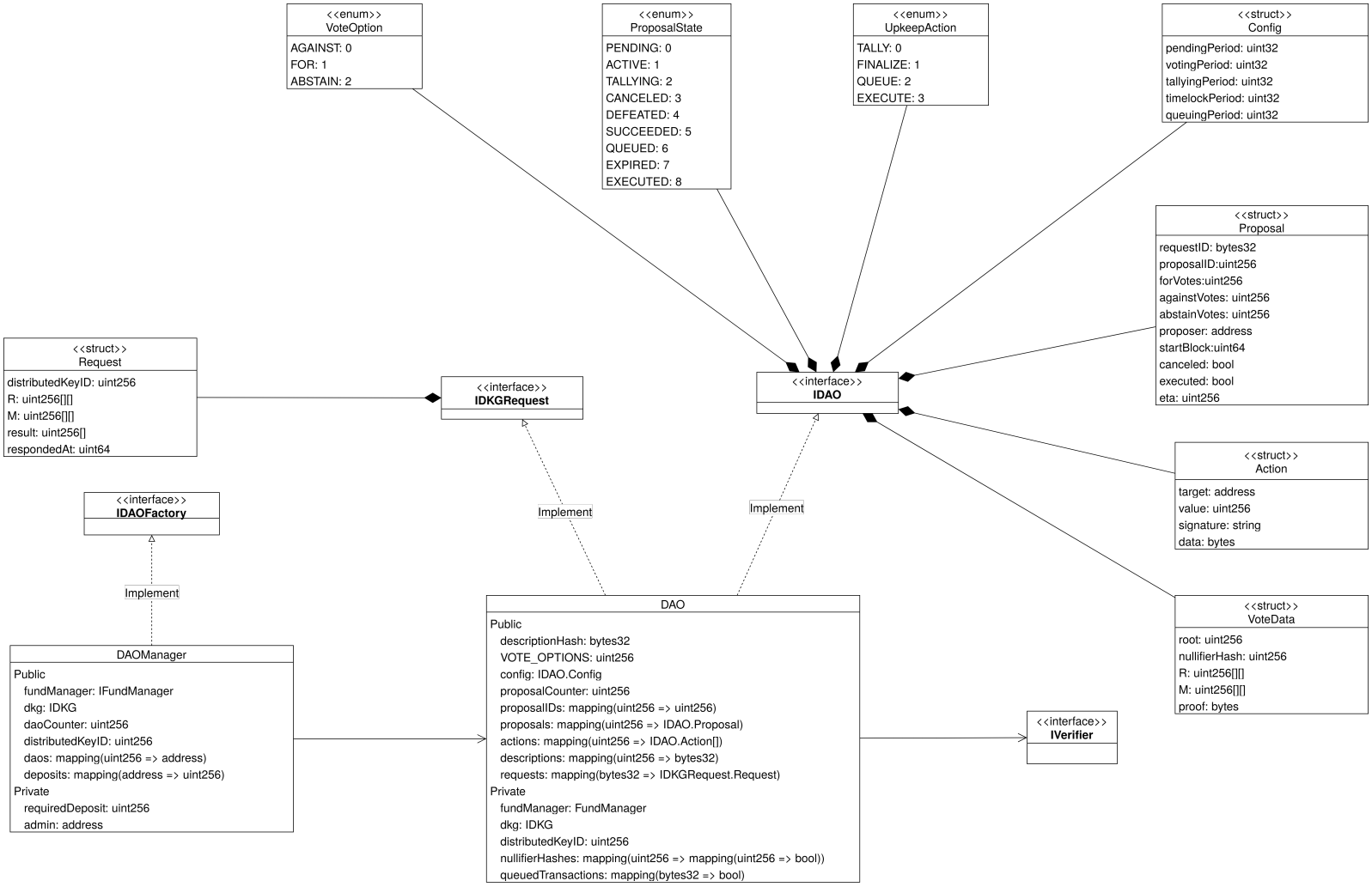

Figure 4.9: Storage class diagram - DAO Manager

The governance process in the Privacy Guard DAO model involves two key smart contracts: DAOManager and DAO. Contract DAOManager is responsible for creating DAO contracts for new projects in the model and managing existing DAOs. It requires knowledge of the FundManager interface to enable DAOs' participation in the funding round. Contract DAO serves as the repository for investor funds and facilitates the "Improvement Proposal" processes. Similar to the FundManager contract, the DAO contract implements the IDKGRequest interface to track completed and ongoing proposals. Additionally, the IDAO interface defines the data structures used in the "Improvement Proposal" processes.

In addition to defining data structures in smart contracts, interfaces also specify functions, events, and modifiers that define how contracts interact with each other and external actors, such as users. While implemented contracts may have different underlying logic, they remain compatible with the original interface and adhere to a standardized method of function call. Similar to storage variables, functions have four types of access modifier: private, public, internal and an additional access modifier is external to specify that the function can only be called from outside the contract. Similar to storage variables, functions have four access modifiers: private, public, internal, and an additional modifier, external, which limits the function to be call only from outside the contract. Events are emitted to log data, while modifiers define code blocks that check conditions for functions utilizing them.

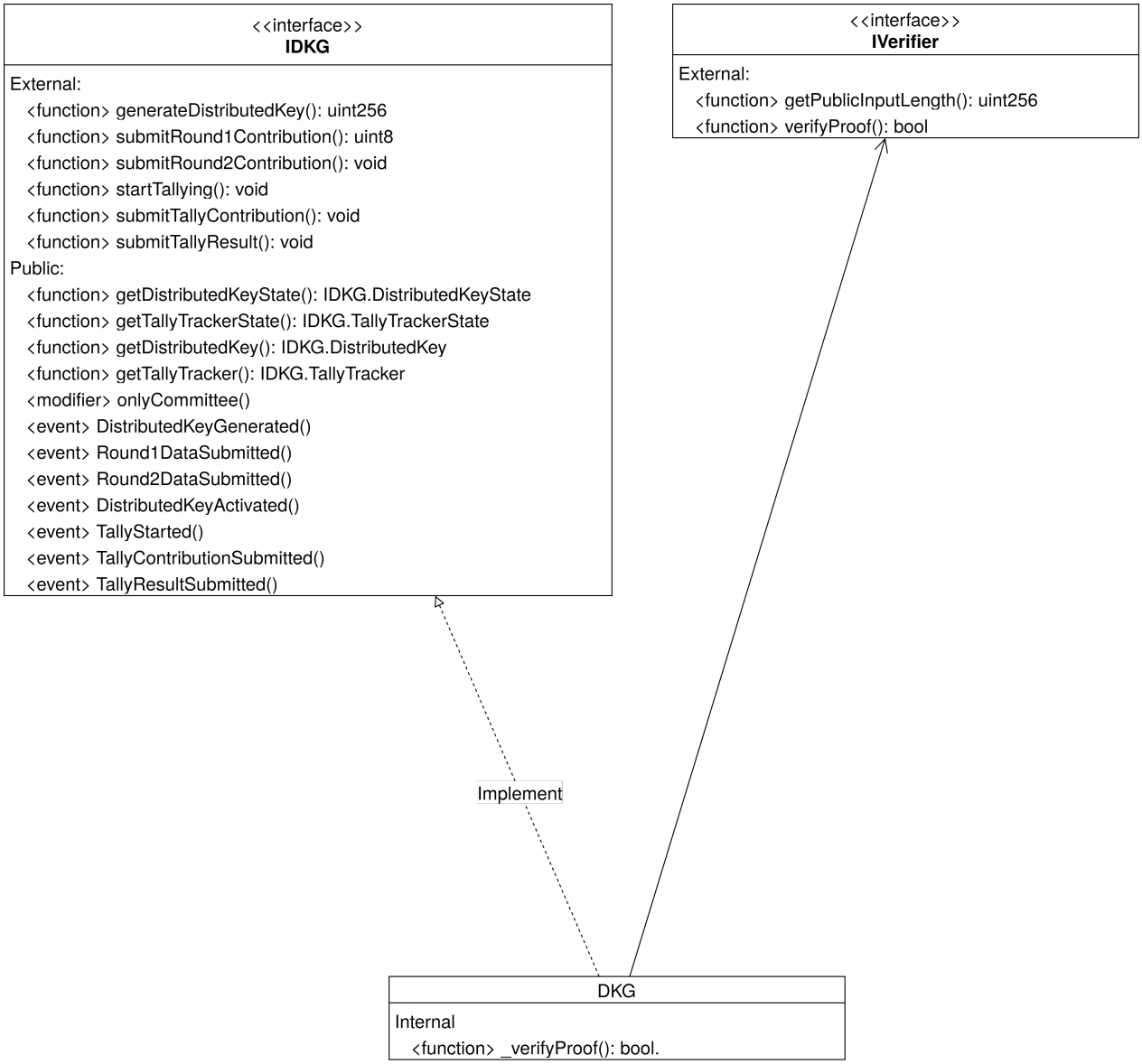

Figure 4.10: Interface class diagram - Distributed Key Generation

The IDKG interface encompasses functions for generating new distributed keys, submitting contributions for establishing public keys, and obtaining final results in both investment and governance processes. It also includes corresponding events to log and query relevant information.

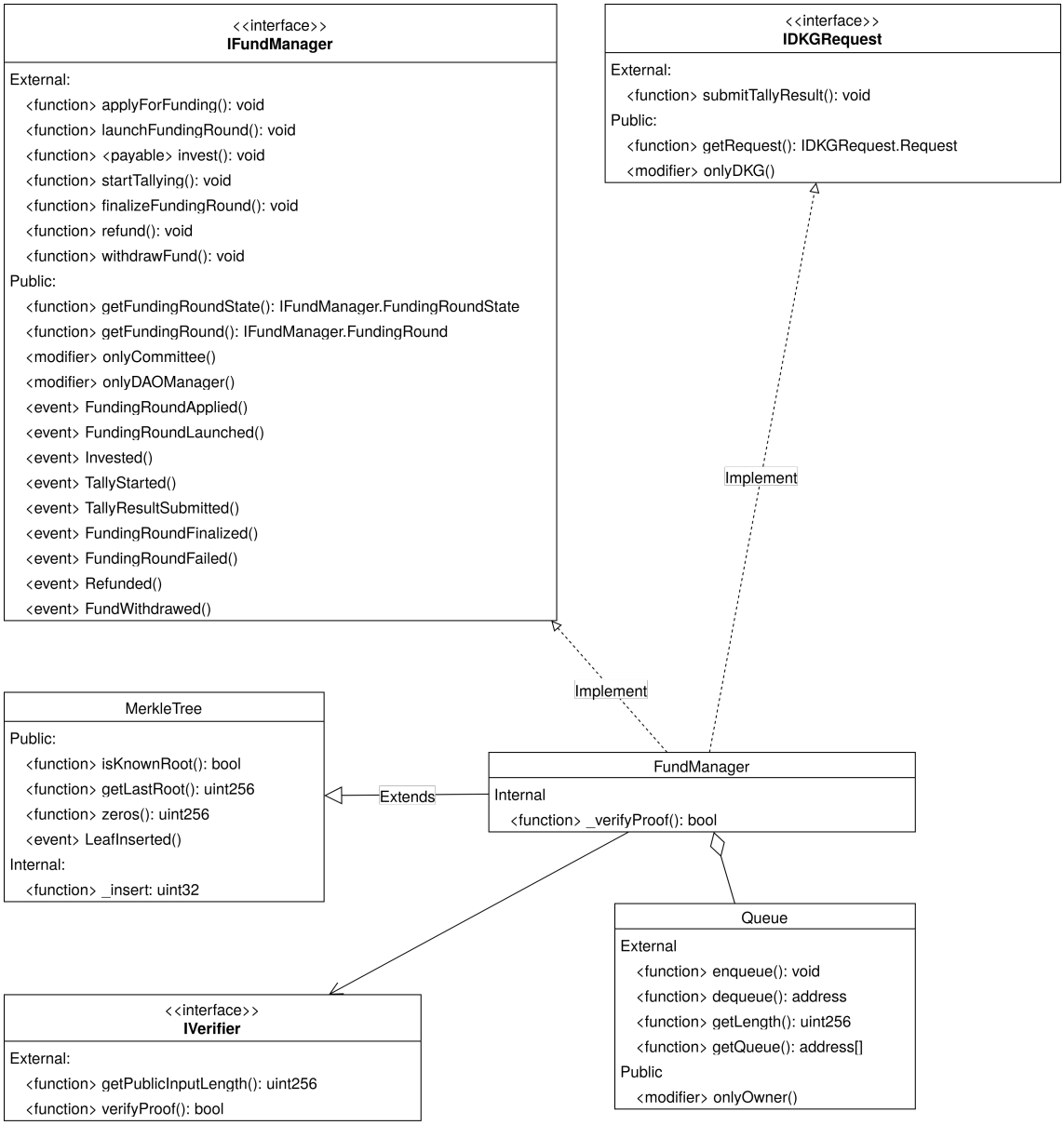

Figure 4.11: Interface class diagram - Fund Manager

The IFundManager interface provides interfaces for handling investment processes and managing funding rounds from launching to finalization. Contract FundManager also implements the IDKGRequest interface to ensure privacy through the use of distributed keys. Moreover, it extends the MerkleTree contract to incorporate the Merkle tree data structure, which tracks user investments. The MerkleTree contract offers a function to insert new leaves when users make investments. Additionally, FundManager utilizes the Queue contract to maintain the funding round queue, where DAOs are queued and await their turn to participate in funding rounds.

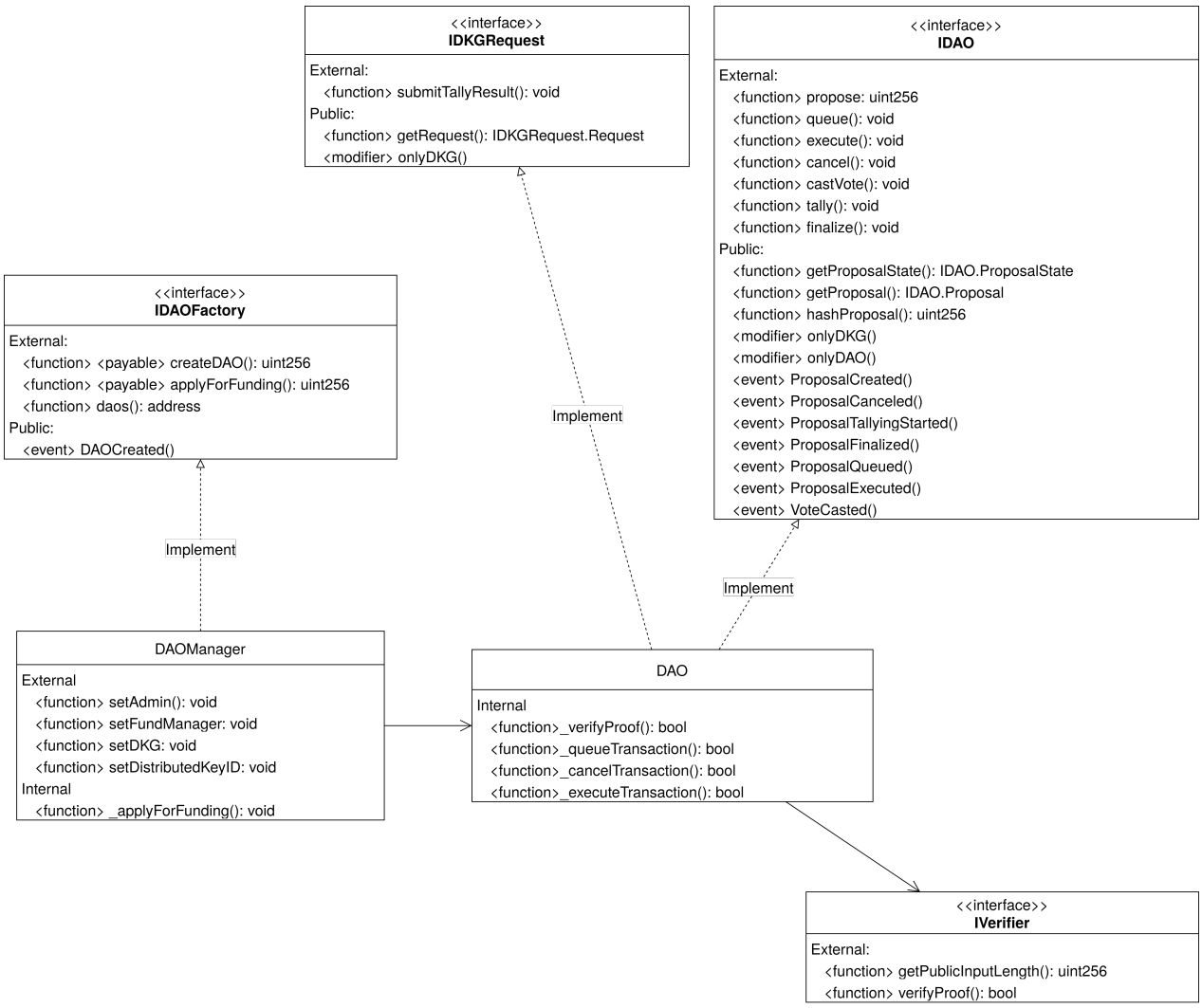

Figure 4.12: Interface class diagram - DAO Manager

In the governance process, the contracts DAOManager and DAO play essential roles. The IDAOFactory interface provides interfaces for creating new DAOs, while the DAO contract implements the IDAO interface, which includes functions for managing proposals, such as create, vote, queue, and execute. Similar to the FundManager contract, the DAO contract also implements the IDKGRequest interface to ensure privacy through the utilization of distributed keys in the "Improvement Proposal" processes.

The detailed description of storage variables will be presented in the Appendix B.

4.2.3 Application Server

In the Privacy Guard DAO model, the smart contract system provides all the core features. The application server's role is to support the smart contract system in optimizing the user experience. Its main functions include listening for events from the smart contract system, logging them to the database, and serving as a con-

venient entry point for the application client to enable easy querying. This ensures that users can enjoy a seamless and optimized experience.

A portion of the on-chain data will be stored in the application server, allowing the application client to query this data through a REST API instead of directly querying the smart contracts. The data provided by the application server is publicly accessible information that anyone can query from the blockchain provider nodes. However, utilizing the application server reduces the computational cost and effort required for data queries by the application client. It provides a more efficient and streamlined approach to accessing the desired data.

The diagram below depicts the architecture design for the application server, presenting the various components and their interactions.

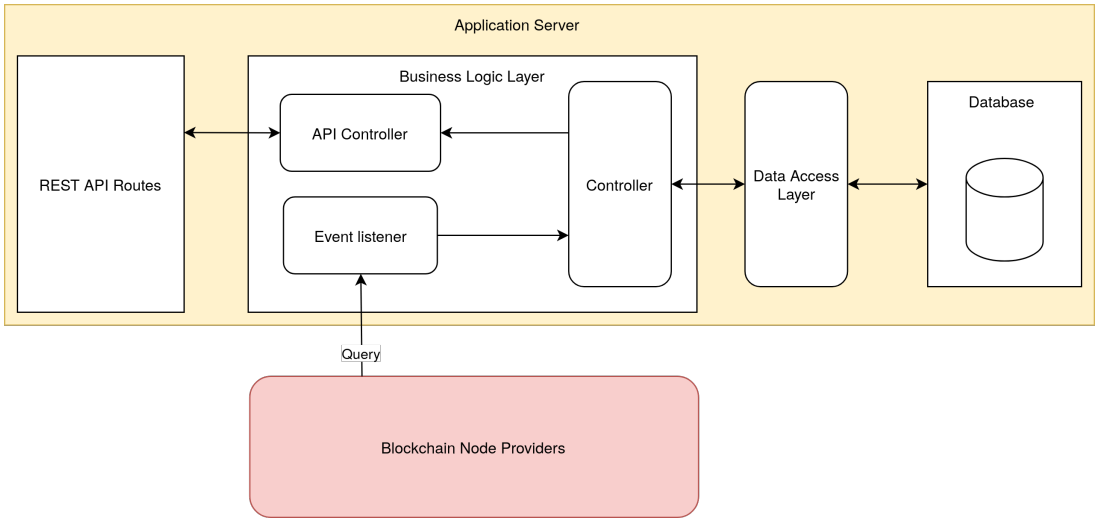

Figure 4.13: Application server architecture

a, Event listener

In EVM-compatible blockchains, smart contracts have the ability to emit events during processing function calls, and these events are stored in the transaction's log on the blockchain. These logs or emitted events can not be directly queried from smart contracts, but Ethereum clients offer an API that allows querying and listening to these data structures. This mechanism provides applications with an alternative method for retrieving data from the blockchain, bypassing direct contract interactions, and also serves as a solution for handling large on-chain data.

In the proposed design of the Privacy Guard DAO model, actions that interact with contracts and result in changes have to emit events containing relevant data. The application server is responsible for listening to these events and storing them in the database. For instance, during the investment process, the application server listens for the insertion of a new leaf into the MerkleTree and provides an API

for users to query the path of that leaf. The path of the leaf in the MerkleTree is essential for participating in the governance process and serves as proof that users have the necessary rights in the DAO.

b, Database design

The database for the proposed model is used to log events and serve to build the Merkle tree so that users can query the path of their leaves. With this purpose, the database design need the simplicity and low query-cost, leading to the adoption of a NoSQL database. The database design consists of five collections, each serving a specific purpose. The details of these collections are depicted in the following diagram.

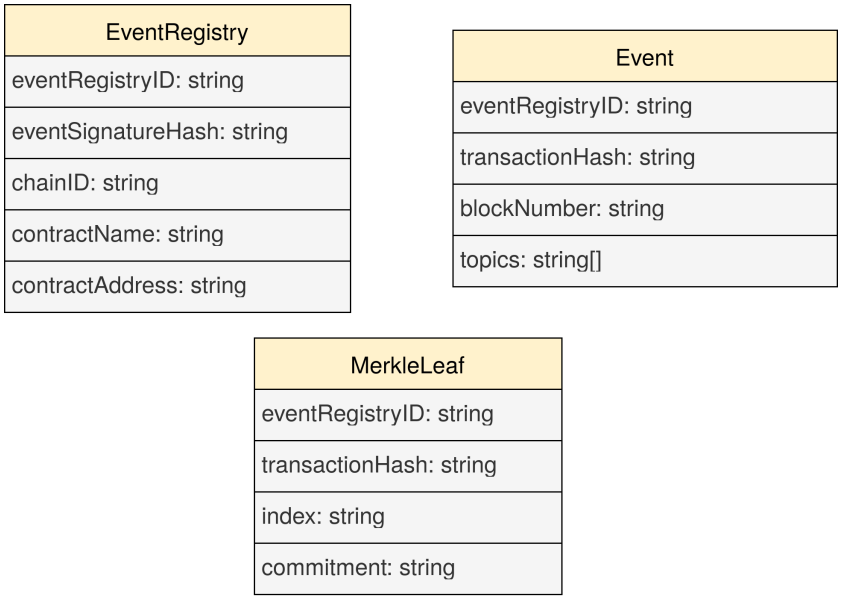

Figure 4.14: Database collections

EventRegistry collection is responsible for storing the properties of events that the model will listen for. It includes essential attributes such as contractAddress and eventSignatureHash, which are used to determine the specific emitted events that will be monitored by the system. This information is crucial for ensuring that the model listens to the relevant events and effectively captures the desired data.

Event collection is responsible for storing the events that have been registered in the EventRegistry. It includes attributes such as transactionHash, which identifies the transaction that emitted the event. Additionally, the collection stores indexed data emitted by events in the form of property topics. These indexed properties allow for efficient querying and retrieval of specific event data.

MerkleLeaf collection is responsible for storing data related to events of leaf insertion into the Merkle Tree. This collection specifically stores the filtered relevant events from the Event collection. By storing this data in the MerkleLeaf collection,

the system can effectively track and manage the leaf insertion events, which are essential for building and maintaining the Merkle Tree structure.

All of these collections are not accessed directly by users; instead, they are provided through a REST API. The data in these collections is constantly updated by the Event Listener module to ensure synchronization between the database server and the blockchain. Some fields in these collections duplicate the data stored in the smart contract system, but they are included to optimize the user experience when using the web application. If there are any inconsistencies between the two environments, users have public access to the blockchains and can report them for an update from the application server. Detailed descriptions of the key-value pairs for these collections can be found in the Appendix C

c, REST API design

REST (Representational State Transfer) is an architectural style commonly employed in the design of networked applications, utilizing HTTP messages [26] as its foundation. It is characterized by its stateless and data-oriented nature, focusing on the provision of system resources through standardized and uniform interfaces defined by URIs (Uniform Resource Identifiers). To ensure security, all APIs are designed as POST methods, allowing for secure transmission of data.

URI	**HTTP Method**	**Description**
/ipfs/upload	POST	Upload a file to IPFS storage
/daos	POST	Retrieve related data of all DAOs
/daos/:daoID/proposals	POST	Retrieve all proposals of a specific DAO
/daos/:daoID/proposals/:proposalID	POST	Retrieve detailed data of a specific proposal
/investment/paths	POST	Retrieve the paths of all leaves in the Merkle tree
/investment/funding-rounds	POST	Retrieve related data all existed funding rounds

/zk/public-key-contribution	POST	Retrieve files for generating ZKP in the public key contribution process
/zk/tally-contribution	POST	Retrieve files for generating ZKP in the tally contribution process
/zk/final-result-contribution	POST	Retrieve files for generating ZKP in the final result contribution process
/zk/vote-casting	POST	Retrieve files for generating ZKP in the vote casting process
/zk/investment	POST	Retrieve files for generating ZKP in the investment process
/committee/distributed-keys	POST	Retrieve related data of all generated distributed keys
/committee/distributed-key-requests	POST	Retrieve related data of all generated distributed key requests
/committee/brute-forces-result	POST	Brute forces the result from a result vector for distributed key requests

Table 4.1: List of APIs

4.2.4 Application client

The application client serves as the user's gateway to the features of the Privacy Guard DAO model, accessible through a web application on the client's browser. Its primary functions include user authentication, facilitating user interactions with the application server and decentralized environments. By using the application client, users can interact with smart contracts without requiring extensive technical knowledge.

a, Authentication and authorization on decentralized environments

In traditional web applications, user authentication commonly involves methods like username-password or one-time password. However, in web3 applications,

users possess an EOA and can authenticate themselves by proving ownership of the account through their private keys, without disclosing any secret information. Users log in to a web3 application using their cryptographic key pairs (public and private keys), and this authentication process occurs on the client's browser, ensuring transparency to the application server.

Each EOA is identified by its unique address, which serves as the representation of that account's information. Smart contracts have responsibilities for authorizing and determining the functions that can be accessed by a specific account address. By specifying the appropriate access controls and permissions within the smart contract, the authorization process ensures that only authorized accounts can invoke specific functions.

The functionalities mentioned are commonly facilitated by a cryptocurrency wallet, also known as a crypto wallet. This software program securely stores users' private keys on the client side and offers an interface for interacting with the blockchain. Users can perform various operations through the wallet, including signing transactions, sending funds, transferring tokens, and more. Crypto wallets can take the form of browser extensions, desktop applications, or mobile applications, providing users with convenient access to their blockchain assets.

b, The generation of zero-knowledge proofs

The generation of zero-knowledge proofs (ZKPs) exclusively on the application client is a fundamental aspect of upholding the confidentiality of sensitive user data within the Privacy Guard DAO model. By generating ZKPs independently of any external servers, users maintain complete concealment over their investment details and voting choices, ensuring their data remains private and secure. This approach not only enhances the overall privacy and security of the system but also instills a sense of trust and confidence in the decentralized nature of the model.

The privacy of user's data achieved through ZKPs generation on the application client provides several benefits. Firstly, it empowers users with sole authority over their sensitive information, preventing any unauthorized access or exposure to third parties. Secondly, the decentralized trust established by this process fosters verifiability and transparency in decision-making within the Privacy Guard DAO model. Model can validate the correctness and accuracy of user's data through ZKPs without the need for the user to provide this data specifically.

In summary, by employing ZKPs generation on the application client, the Privacy Guard DAO model achieves a robust privacy framework, user empowerment, enhanced verifiability, and increased efficiency, all contributing to a more secure

governance system. The details about performance specification of ZKPs can be found in the Appendix D.

c, User interface

The UI of the application client follows the design principles [27] outlined below:

- The structure principle: The UI components are organized purposefully, considering the requirements of the business processes. Related components within a process are grouped together, promoting intuitive navigation and efficient user interaction.
- The visibility principle: All system functions are accessible through the UI, ensuring that users can easily access and utilize the available features.
- The simplicity principle: The business processes are implemented in a straightforward and clear manner. The UI provides sufficient information to users, including success or error messages, tooltips, and other relevant details.

The structure of the application client's web pages is presented in a tree-like format, illustrating the flow of access to each page. This is followed by a description table that provides detailed information about each page, outlining its specific functionalities and features.

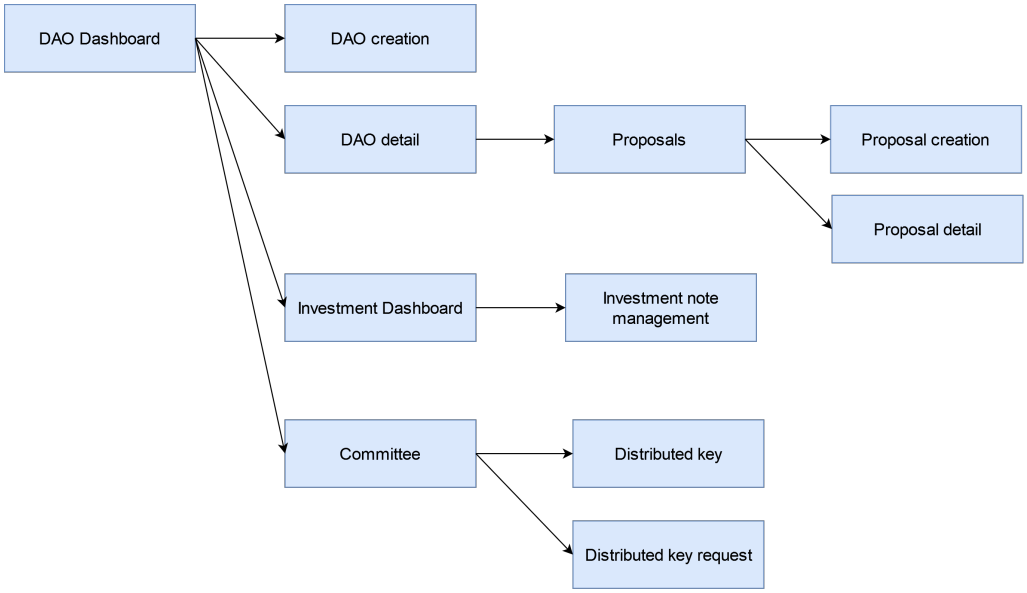

Figure 4.15: Web pages tree diagram

Pages	**Description**
System Dashboard	• The page contains a list of all DAO created in the system and their basic information. • It defines a consistent layout for all downstream pages, with functionality to connect and disconnect their wallet. • Users can search for DAO by name or filters. • Users can choose to create a new DAO or see details of an existing DAO, which will both redirect to another page.
Create DAO	• The page consists of two components, a form for DAO creation contract function call and a form for uploading DAO's information to IPFS. • If the creation process succeeded, it will redirect user to another page.
DAO Dashboard	• The page consists of two components, one displays the information of the selected DAO, retrieved from the IPFS and the smart contract storage, one list all events related to the DAO logged by the application server. • Users can choose to redirect to other web pages for more information about the DAO.
Contracts	• The page generates an interface for interactions with the DAO's core smart contract system by function calls.

Proposals	• The page list all existing proposals in a DAO with basic information. • Users can search for a proposal by name or filters. • Users can select to see details of a proposal or create a new proposal, which will redirect to another page.
Create Proposal	• The page consists of two components, a form for the proposal's actions input and a form for uploading the proposal's description to IPFS. • If the creation process succeeded, it will redirect user to another page.
Proposal Detail	• The page displays the information of the selected proposal, retrieved from the IPFS and the smart contract storage. • Users can cast their vote for the selected proposal on this page.
Proposal Detail	• The page displays the information of the selected proposal, retrieved from the IPFS and the smart contract storage. • Users can cast their vote for the selected proposal on this page.
Investment Dashboard	• The page displays a list of all completed and ongoing funding rounds. • Users have the ability to view detailed information about any funding round.

Investment note management	• The page displays a list of all investment notes stored in the browser local storage. • Users can add new investment notes to the list. • Users have the ability to delete or update existing investment notes.
Distributed key	• The page displays a list of all existing distributed keys. • Committee members can manage contribution data stored in the browser local storage: generate new data, add data from a file, or delete existing data. • The page provides features for committee members to manage their contributions to the distributed keys by submitting their contributions.
Distributed key request	• The page displays a list of all existed distributed key requests. • Committee members can manage contribution data stored in the browser local storage: generate new data, add data from a file, or delete existing data. • The page provides features for committee members to manage their contributions to the distributed key requests by submitting their contributions.

Table 4.2: Web pages description

5.1 Technologies

Various of tools, libraries, and frameworks are employed to develop a prototype demonstration for the proposed model presented in this thesis. Technologies utilized are detailed in the following table.

System	**Technology**	**Note**
Integrated development environment (IDE)	Visual Studio Code	
Version control environment	Git & Github	
Public blockchains	Sepolia (Ethereum testnet)	
Smart contract programming language	Solidity	Version $\geq$ 0.8.0
Smart contract development	Hardhat	
Arithmetic circuit programming language for ZK-SNARK	Circom	Version $\geq$ 2.0.0
Arithmetic circuit development for ZK-SNARK	snarkjs	Version $\geq$ 0.7.0
Blockchain node and IPFS providers service	Infura	Free-tier
Database	MongoDB	
Application server runtime environment	NodeJS	Version $\geq$ 16.14.0
Application client library	ReactJS	
Application programming language	JavaScript and TypeScript	
Client crypto wallets	Metamask	

Web server	NGINX	

Table 5.1: Core technologies for development

The smart contracts in this thesis are implemented using the Solidity programming language, supported by the Hardhat framework for tasks such as editing, compiling, debugging, testing, and deploying. The smart contract system is deployed on Sepolia, an Ethereum testnet. This decision is driven by Ethereum's prominence in both market capitalization and the number of organizations utilizing it. To interact with these contracts, the prototype application leverages the free-tier service provided by Infura.

For building ZKPs, the arithmetic circuits are programmed using the Circom language and compiled through the Circom compiler. Alongside, the implementation of ZK-SNARK in JavaScript, snarkjs, is employed. Snarkjs incorporates all the necessary tools to conduct trusted setup multi-party ceremonies, including the universal powers of tau ceremony and circuit-specific ceremonies for the second phase. This library is an ES module, allowing direct importation into larger projects using Rollup or Webpack. The low-level cryptography operations are performed directly in wasm and leverage worker threads to parallelize computations, ensuring high performance with benchmarks comparable to host implementations.

Web application development is facilitated using widely adopted platforms like ReactJS, NodeJS, and MongoDB. For seamless and secure client crypto wallet connectivity, Metamask is employed. Lastly, the developed prototype is deployed on a web server with NGINX, offering enhanced accessibility and usability for end-users.

5.2 Demonstration

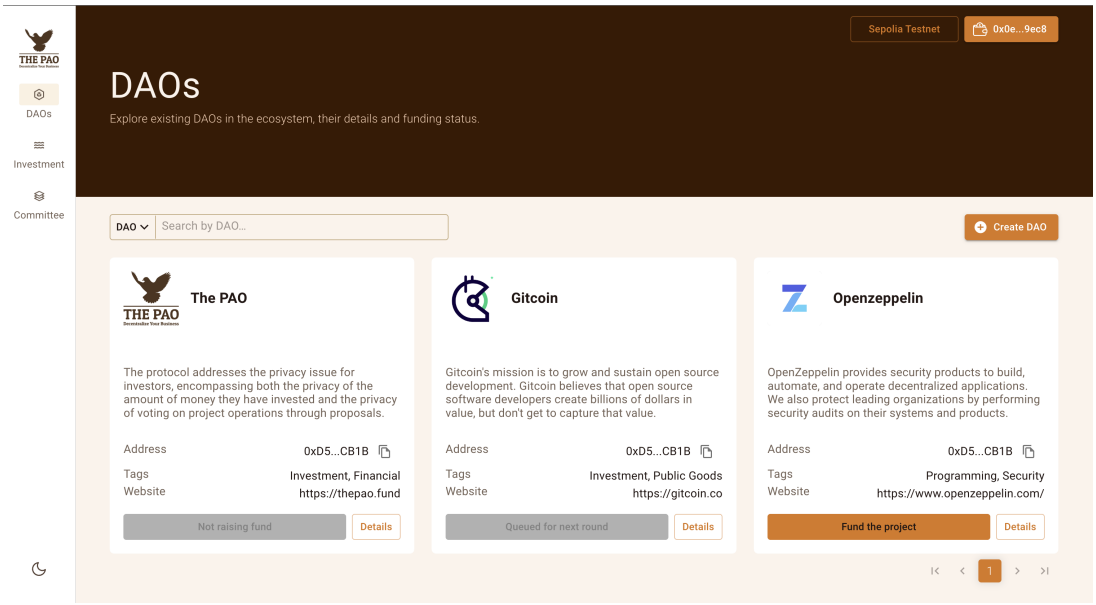

Figure 5.1: DAO dashboard page

The DAO dashboard page lists all DAOs in the system, displaying essential information such as the DAO contract address, website URL, or description. Users can easily search for DAOs by name and access detailed information about each DAO.

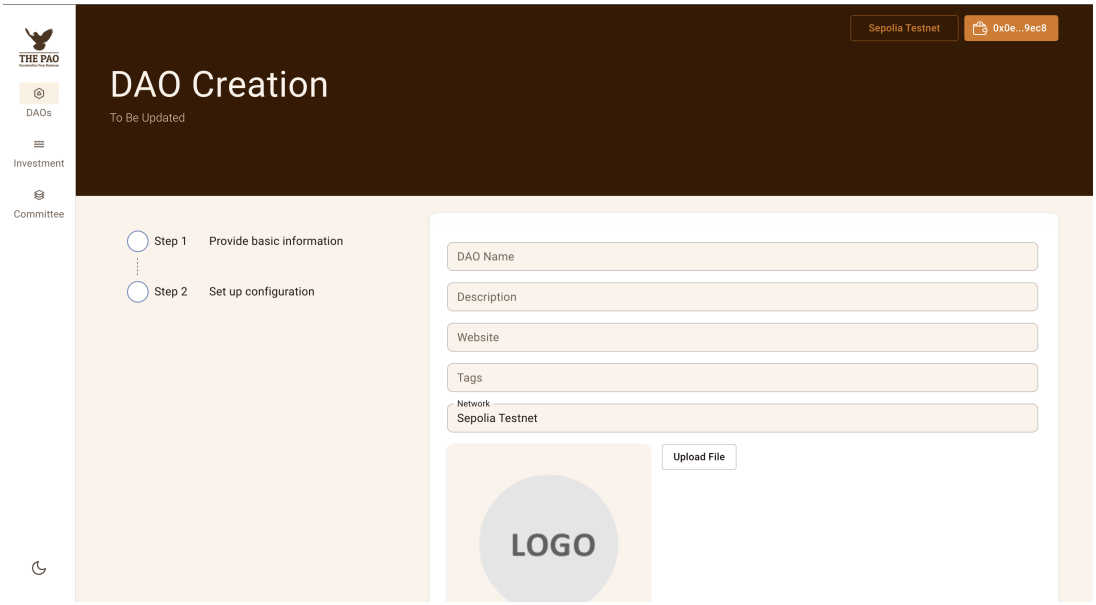

Figure 5.2: DAO creation page

Anyone who connects their wallet to the application client can create a new DAO. The process involves providing basic information and configuring parameters for the DAO.

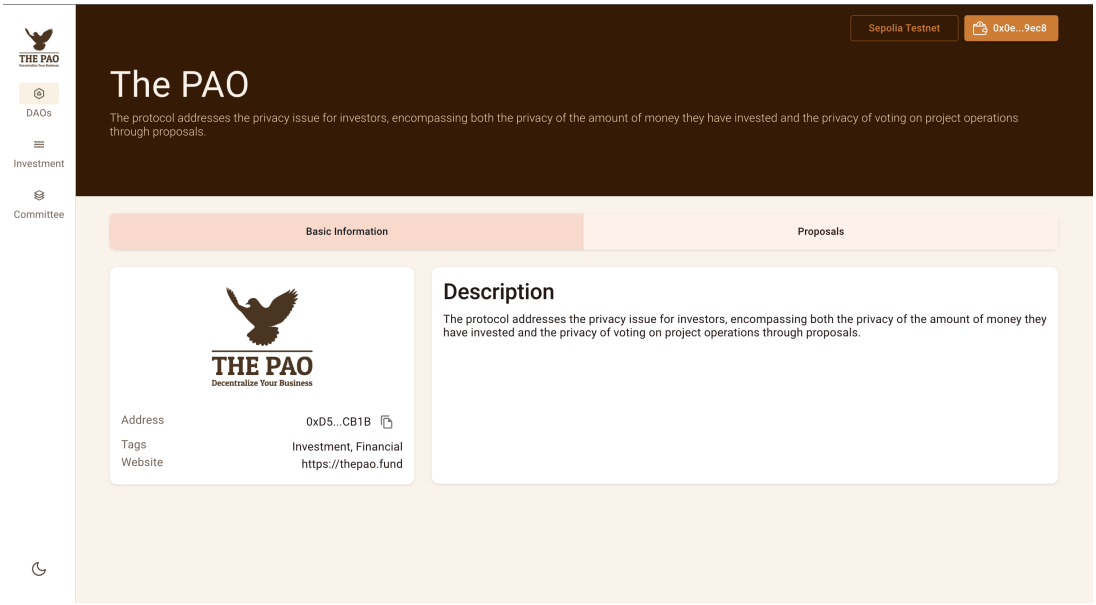

Figure 5.3: DAO detail page

The DAO detail page presents comprehensive information about a specific DAO, retrieved directly from IPFS. Users can also explore all the proposals associated with the DAO from this page.

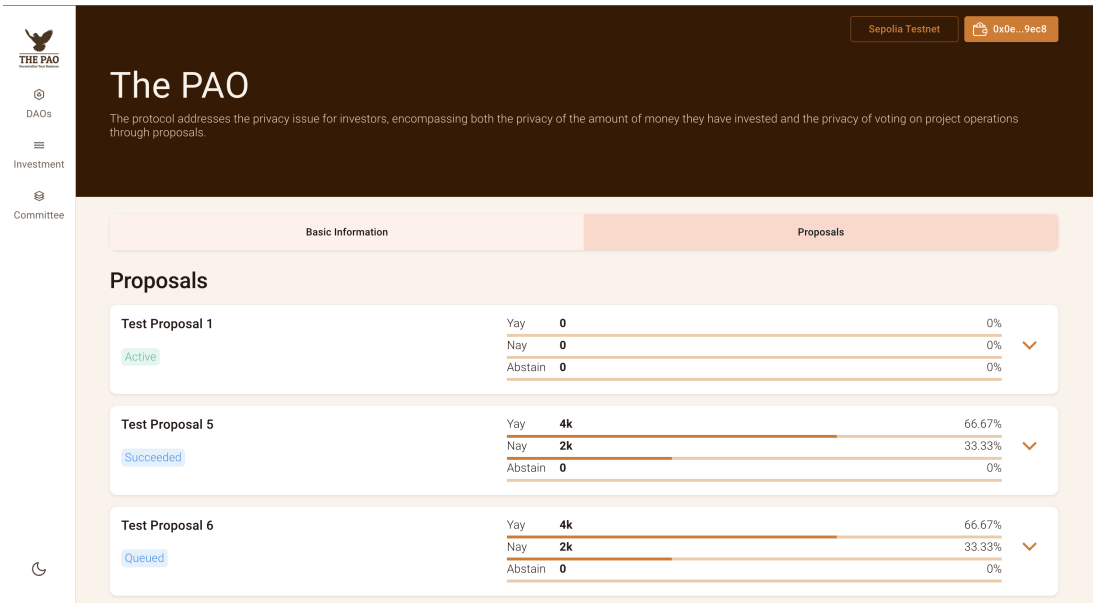

Figure 5.4: Proposals page

The Proposals page lists all existing proposals for a DAO, providing essential details such as the title, status, and voting results.

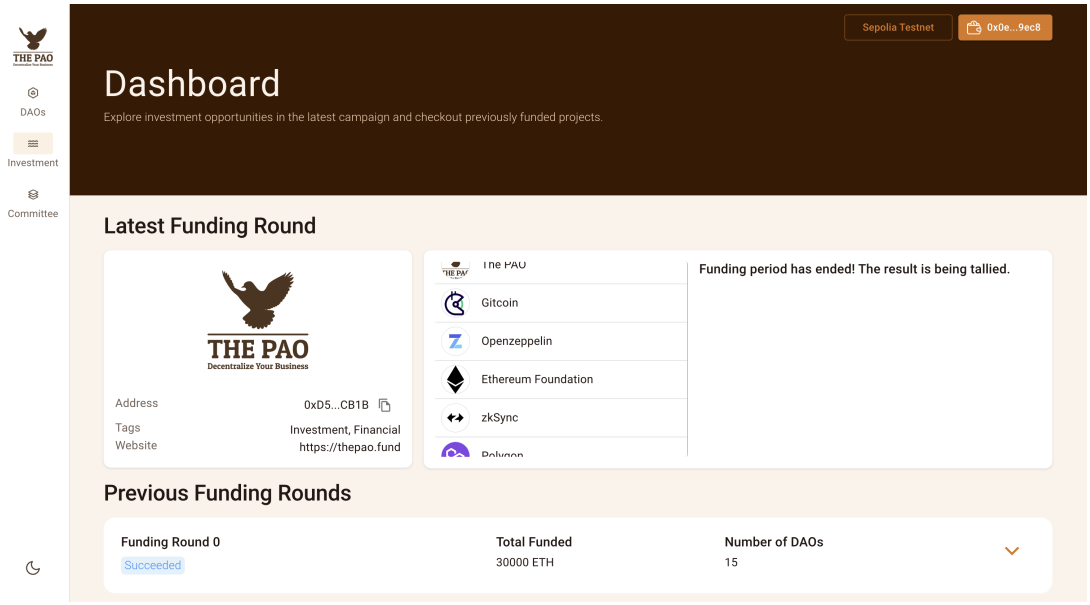

Figure 5.5: Investment dashboard page

The investment dashboard page displays all launched funding rounds in the system, presenting crucial information like the status, total funded amount, and number of participating DAOs for each round.

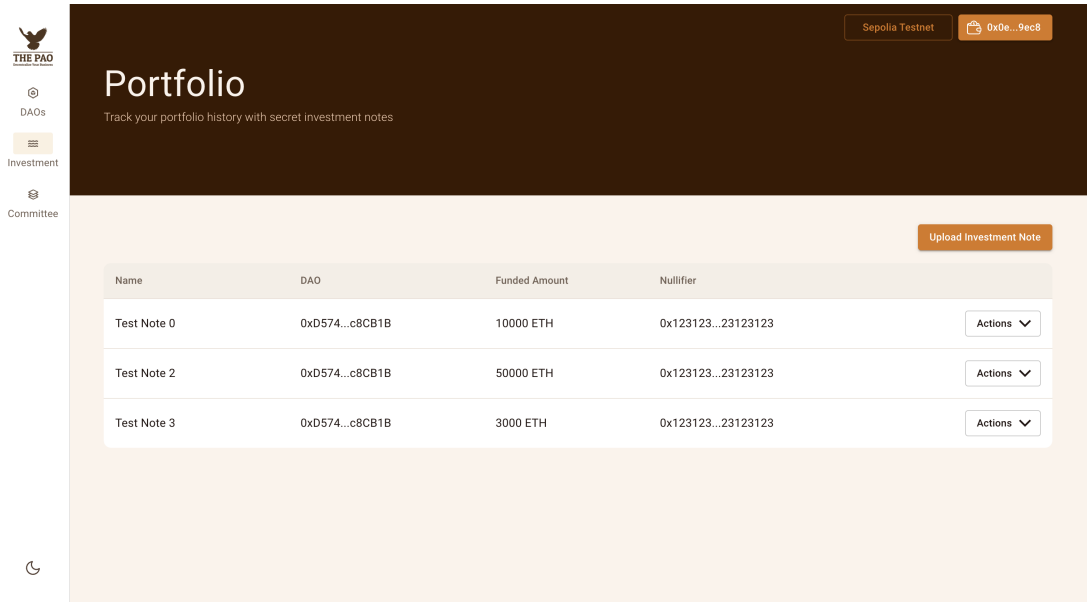

Figure 5.6: Investment note management page

The system offers an investment note management page to assist users in managing their investment notes. Here, users can upload, delete, and update their existing investment notes.

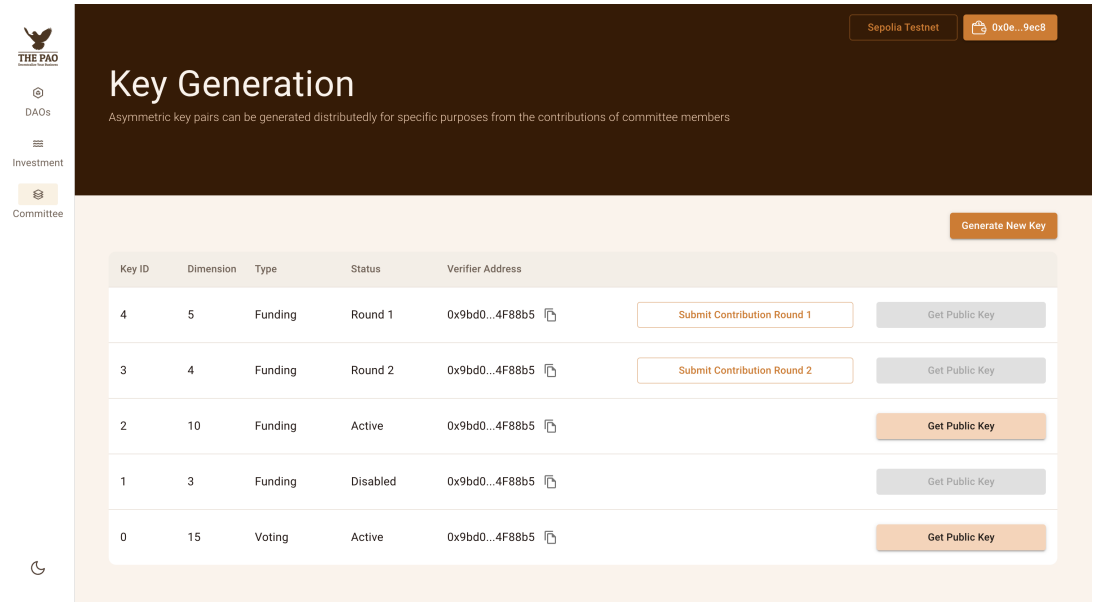

Figure 5.7: Distributed key page

For committee members, the distributed key page displays a list of generated distributed keys. It provides essential information such as the status and key type, and allows committee members to interact and submit contributions.

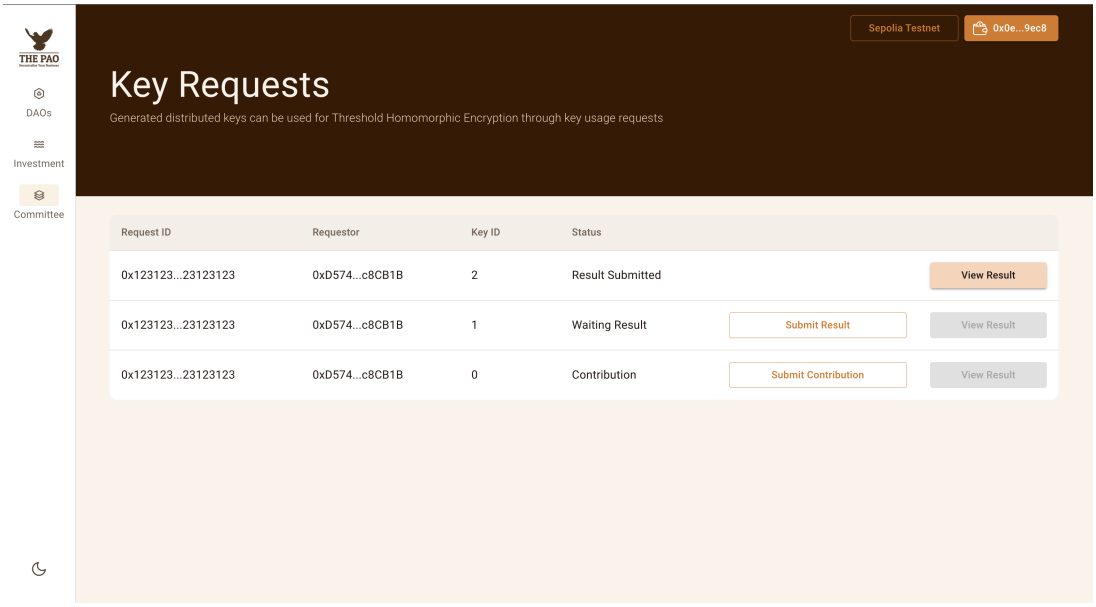

Figure 5.8: Distributed key request page

The resulting requests when investments or governance processes using distributed keys will be displayed on the distributed key request page. Additionally, this page allows committee members to submit their contributions for result contribution processes.

CHAPTER 6. CONCLUSION AND FUTURE WORK

6.1 Contribution

This thesis has successfully achieved its objectives, which encompass introducing the DAO concept and integrating it with Zero-knowledge Proof technology to propose a solution for the Privacy Guard DAO model. The implementation covers all the specified requirements and offers a well-structured design for a DAO model with investment purposes. The proposed design encompasses all fundamental components essential for a Ventures DAO, enabling participants to make investment choices and subsequently participate in the governance processes of the corresponding project. Most importantly, these processes ensure privacy and confidentiality, providing a robust and secure framework for decentralized decision-making.

The core foundation of the Privacy Guard DAO model lies in the combination of distributed key generation mechanism and Zero-knowledge technology. This combination enables the utilization of public key cryptography without the need for privileged trusted parties. Consequently, encrypted information can be stored on the blockchain without involving individuals or centralized organizations to maintain the private key. The participation of committee members resembles validator nodes in the decentralized operation of the blockchain, ensuring a robust and transparent system.

The integration of distributed key generation mechanism and Zero-knowledge technology forms a novel Ventures DAO model, operating under the "Improvement Proposal" workflow. Notably, this model also facilitates the creation of individual DAOs for organizations or startups without requiring to write code or deploy contract by themselves. Additionally, the proposed design optimizes gas fees by incorporating IPFS, a decentralized storage platform, for off-chain storage of large-size data.

To enhance user experience, the system cleverly utilizes an application server. The server serves two main purposes: listening to blockchain data, including event emissions, and streamlining user interactions by efficiently querying and processing decentralized data. The acquired data is stored in the server's database, then presented on the client side, facilitating easy and transparent access to governance activities for organizations' shareholders. This approach ensures a seamless and user-friendly interface while preserving the transparency and publicity of the blockchain.

6.2 Future Work

The DAO model presented in this thesis seeks to encompass fundamental functionalities for organizations. However, there remains significant potential for improvement and the necessity for future research and development.

One area of focus for future work involves expanding the Privacy Guard DAO model to accommodate more diverse and complex use cases. This entails exploring additional business processes and functionalities that organizations may require in their decentralized decision-making and governance. Enhancements can be made to support a wider range of investment types and voting mechanisms, such as stock mechanisms or quadratic voting. Additionally, support for more token standards like ERC-20 [28], ERC-721 [29], and ERC-1155 [30] should be considered to ensure adaptability to various organizational needs.

Another crucial aspect to consider is the complexity of the current model's business processes, which may hinder accessibility for general users. Additionally, the active participation of committee members plays a significant role in the system. Future designs should strive for simplicity, ease of access, and reduced reliance on committee members, promoting inclusivity and user-friendliness.

In conclusion, while this thesis has made significant progress in achieving privacy for decision-making on blockchains, ongoing efforts in research and development are vital to fully capitalize on its potential. By addressing the outlined areas for improvement, the model can evolve into a more versatile, accessible, and user-centric solution, furthering the advancement of decentralized organizational frameworks.

REFERENCE

[1] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[2] G. R. Blakley, "Safeguarding cryptographic keys," in *Managing Requirements Knowledge, International Workshop on*, IEEE Computer Society, 1979, pp. 313–313.

[3] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," in *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, IEEE, 1985, pp. 383–395.

[4] T. P. Pedersen, "A threshold cryptosystem without a trusted party," in *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*, Springer, 1991, pp. 522–526.

[5] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*, Springer, 1987, pp. 369–378.

[6] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2012, pp. 326–349.

[7] S. Haber and W. S. Stornetta, *How to time-stamp a digital document*. Springer, 1991.

[8] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, 2008.

[9] V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the theory and application of cryptographic techniques*, Springer, 1985, pp. 417–426.

[10] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, pp. 36–63, 2001.

[11] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[12] Protocol Labs, *What is ipfs?* [Online]. Available: `https://docs.ipfs.io/concepts/what-is-ipfs`.

[13] V. Buterin, *Bootstrapping a decentralized autonomous corporation: Part i. bitcoin magazine (2013)*.

[14] Q. DuPont, "Experiments in algorithmic governance: A history and ethnography of "the dao," a failed decentralized autonomous organization," in *Bitcoin and beyond*, Routledge, 2017, pp. 157–177.

[15] Avara UI Labs Ltd., *Aave website*. [Online]. Available: `https://docs.aave.com/`.

[16] Avara UI Labs Ltd., *Aave docs governance*. [Online]. Available: `https://docs.aave.com/governance`.

[17] Compound Lab, Inc., *Compound website*. [Online]. Available: `https://compound.finance/`.

[18] Compound Lab, Inc., *Compound docs - governance*. [Online]. Available: `https://compound.finance/docs/governance`.

[19] *Bitdao website*. [Online]. Available: `https://www.bitdao.io/`.

[20] *The lao website*. [Online]. Available: `https://thelao.io/`.

[21] B. WhiteHat, J. Baylina, and M. Bellés, "Baby jubjub elliptic curve," *Ethereum Improvement Proposal, EIP-2494*, vol. 29, 2020.

[22] J. Baylina and M. Bellés, "4-bit window pedersen hash on the baby jubjub elliptic curve,"

[23] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for {zero-knowledge} proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 519–535.

[24] L. Grassi, D. Khovratovich, and M. Schofnegger, "Poseidon2: A faster version of the poseidon hash function," *Cryptology ePrint Archive*, 2023.

[25] B. Bauer and J. Odell, "Uml 2.0 and agents: How to build agent-based systems with the new uml standard," *Engineering applications of artificial intelligence*, vol. 18, no. 2, pp. 141–157, 2005.

[26] M. Masse, *REST API design rulebook: designing consistent RESTful web service interfaces*. " O'Reilly Media, Inc.", 2011.

[27] L. L. Constantine and L. A. Lockwood, *Software for use: a practical guide to the models and methods of usage-centered design*. Pearson Education, 1999.

[28] F. Vogelsteller and V. Buterin, "Eip 20: Erc-20 token standard," *Ethereum Improvement Proposals*, vol. 20, 2015.

[29] E. William, S. Dieter, E. Jacob, and S. Nastassia, "Eip-721: Erc-721 non-fungible token standard," *Ethereum Improvement Proposals*, no. 721, 2018.

[30] W. Radomski, A. Cooke, P. Castonguay, J. Therien, E. Binet, and R. Sandford, "Eip 1155: Erc-1155 multi token standard," *Ethereum, Standard*, 2018.

[31] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, Springer, 2016, pp. 305–326.

APPENDIX

A. USE CASES DESCRIPTION

The detailed specifications for use cases in the Privacy Guard DAO model are presented in the following tables, covering both the main flows and alternative flows (if any) for each scenario.

<table>
<tr><td>Name</td><td colspan="2">Connect wallet</td></tr>
<tr><td>Actors</td><td colspan="2">Unconnected Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User accessed the application</td></tr>
<tr><td>Summary</td><td colspan="2">User connect their wallet to the application</td></tr>
<tr><td>Main flow</td><td colspan="2"><table><tr><th>Step</th><th>Action</th></tr><tr><td>1</td><td>User clicks on Connect wallet button</td></tr><tr><td>2</td><td>The application prompts to connect wallet to the application</td></tr><tr><td>3</td><td>User accepts the connection</td></tr><tr><td>4</td><td>The application handles the connection and reload the UI</td></tr></table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table><tr><th>Step</th><th>Action</th></tr><tr><td>3a</td><td>User denies the connection</td></tr></table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">The application updates the UI for the connected wallet</td></tr>
</table>

Table A.1: Connect wallet use case specification

<table>
<tr><td>Name</td><td colspan="2">Disconnect wallet</td></tr>
<tr><td>Actors</td><td colspan="2">Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User accessed the application and connected their wallet</td></tr>
<tr><td>Summary</td><td colspan="2">User disconnects their wallet from the application</td></tr>
<tr><td>Main flow</td><td colspan="2"><table><tr><th>Step</th><th>Action</th></tr><tr><td>1</td><td>User clicks on the Disconnect wallet button</td></tr><tr><td>2</td><td>The application handles the disconnection and reloads the UI</td></tr></table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">The application updates the UI for the unconnected wallet</td></tr>
</table>

Table A.2: Disconnect wallet use case specification

Name	View all funding rounds	
Actors	Unconnected, Connected Account, Committee Account	
Summary	User views all funding rounds in the system	
Main flow	**Step**	**Action**
	1	User accesses the application
	2	User clicks on Investment button
	3	The application gets data of all funding rounds
	4	The application renders a list of all funding rounds to the UI

Table A.3: View all funding rounds use case specification

Name	View funding round detail	
Actors	Unconnected Account, Connected Account, Committee Account	
Pre-conditions	User is viewing the list of funding rounds in the system	
Summary	User views detailed information of a specific funding round	
Main flow	**Step**	**Action**
	1	User selects a funding round from the list
	2	The application gets detailed information of a funding round selected by the user
	3	The application updates the UI to show detailed information of the selected funding round

Table A.4: View funding round detail use case specification

<table>
<tr><td>Name</td><td>Invest</td></tr>
<tr><td>Actors</td><td>Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td>User is viewing detailed information of a funding round, and that funding round has an active investment period</td></tr>
<tr><td>Summary</td><td>User makes an investment for a funding round</td></tr>
<tr><td>Main flow</td><td>
<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User fills the investment form and then submits</td></tr>
<tr><td>2</td><td>The application prompts the user to accept sending the transaction</td></tr>
<tr><td>3</td><td>User accepts sending the transaction</td></tr>
<tr><td>4</td><td>The application waits for the transaction to be confirmed and then reloads the UI</td></tr>
<tr><td>5</td><td>The application automatically stores the investment note to browser local storage</td></tr>
</table>
</td></tr>
<tr><td>Alternative Flow</td><td>
<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>3a</td><td>User denies sending the transaction</td></tr>
<tr><td>4a</td><td>The transaction has failed, and the application client prompts the user</td></tr>
</table>
</td></tr>
<tr><td>Post-conditions</td><td>The funding round gets one more investment. And the new investment note is added to browser local storage</td></tr>
</table>

Table A.5: Invest use case specification

<table>
<tr><td>Name</td><td colspan="2">Invest</td></tr>
<tr><td>Actors</td><td colspan="2">Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing detailed information of a funding round, and that funding round has failed</td></tr>
<tr><td>Summary</td><td colspan="2">User makes a refund for a failed funding round</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User clicks on Refund button</td></tr>
<tr><td>2</td><td>The application prompts the user to accept sending the transaction</td></tr>
<tr><td>3</td><td>User accepts sending the transaction</td></tr>
<tr><td>4</td><td>The application waits for the transaction to be confirmed and then reloads the UI</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>3a</td><td>User denies sending the transaction</td></tr>
<tr><td>4a</td><td>The transaction has failed, and the application client prompts the user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">The failed funding rounds refunds to user</td></tr>
</table>

Table A.6: Refund use case specification

<table>
<tr><td>Name</td><td colspan="2">Manage investment note</td></tr>
<tr><td>Actors</td><td colspan="2">Unconnected Account, Connected Account, Committee Account</td></tr>
<tr><td>Summary</td><td colspan="2">User manages their stored investment notes</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User accesses the application</td></tr>
<tr><td>2</td><td>User clicks on Manage investment note button</td></tr>
<tr><td>3</td><td>The application loads browser local storage and then updates the UI to show a list of all stored investment notes</td></tr>
</table></td></tr>
</table>

Table A.7: Manage investment note use case specification

<table>
<tr><td>Name</td><td colspan="2">Add investment note</td></tr>
<tr><td>Actors</td><td colspan="2">Unconnected Account, Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing the list of stored investment notes</td></tr>
<tr><td>Summary</td><td colspan="2">User adds a new investment note to their list</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User clicks on Add button</td></tr>
<tr><td>2</td><td>The application updates UI and shows a form to add new investment note</td></tr>
<tr><td>3</td><td>User fills necessary information and submits</td></tr>
<tr><td>4</td><td>The application validates the user’s input information, adds a new investment note to browser local storage, and then updates the UI to show the list</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>4a</td><td>The application validates the user’s input information as incorrect and then prompts user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">A new investment note is added to browser local storage</td></tr>
</table>

Table A.8: Add investment note use case specification

<table>
<tr><td>Name</td><td colspan="2">Download investment note</td></tr>
<tr><td>Actors</td><td colspan="2">Unconnected Account, Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing the list of stored investment notes</td></tr>
<tr><td>Summary</td><td colspan="2">User downloads an investment note to their device</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User selects an investment note from the list</td></tr>
<tr><td>2</td><td>User clicks on Download button</td></tr>
<tr><td>3</td><td>The application downloads the file to the user’s device</td></tr>
</table></td></tr>
</table>

Table A.9: Download investment note use case specification

<table>
<tr><td>Name</td><td colspan="2">Update investment note</td></tr>
<tr><td>Actors</td><td colspan="2">Unconnected Account, Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing the list of stored investment notes</td></tr>
<tr><td>Summary</td><td colspan="2">User updates information for a specific investment note</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User selects an investment note from the list</td></tr>
<tr><td>2</td><td>User clicks on Update button</td></tr>
<tr><td>3</td><td>The application shows the form for user to edit information of the selected investment note</td></tr>
<tr><td>4</td><td>User fills necessary information and submits</td></tr>
<tr><td>5</td><td>The application validates the user’s input information, updates the corresponding investment note, and then updates the UI</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>5a</td><td>The application validates the user’s input information as incorrect, and then prompts user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">The information of a specific investment note is updated</td></tr>
</table>

Table A.10: Update investment note use case specification

<table>
<tr><td>Name</td><td colspan="2">Delete investment note</td></tr>
<tr><td>Actors</td><td colspan="2">Unconnected Account, Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing the list of stored investment notes</td></tr>
<tr><td>Summary</td><td colspan="2">User deletes an investment note from the list</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User selects an investment note from the list</td></tr>
<tr><td>2</td><td>User clicks on Delete button</td></tr>
<tr><td>3</td><td>The application warns user about the deletion of the selected investment note</td></tr>
<tr><td>4</td><td>User accepts deleting the selected investment note</td></tr>
<tr><td>5</td><td>The application deletes the selected investment note from browser local storage</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>4a</td><td>User denies deleting the selected investment note</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">An investment note is deleted from browser local storage</td></tr>
</table>

Table A.11: Delete investment note use case specification

<table>
<tr><td>Name</td><td colspan="2">View all DAOs</td></tr>
<tr><td>Actors</td><td colspan="2">Unconnected, Connected Account, Committee Account</td></tr>
<tr><td>Summary</td><td colspan="2">User views all DAOs in the system</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User accesses the application</td></tr>
<tr><td>2</td><td>The application gets data of all DAOs</td></tr>
<tr><td>3</td><td>The application renders a list of all DAOs to the UI</td></tr>
</table></td></tr>
</table>

Table A.12: View all DAOs use case specification

<table>
<tr><td>Name</td><td colspan="2">Search for DAOs</td></tr>
<tr><td>Actors</td><td colspan="2">Unconnected, Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing the list of DAOs in the system</td></tr>
<tr><td>Summary</td><td colspan="2">User searches for DAOs by their input keywords</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User inputs keywords for searching</td></tr>
<tr><td>2</td><td>The application handles the user’s keywords to find DAOs matched with the keywords</td></tr>
<tr><td>3</td><td>The application updates the UI to show the list of suitable DAOs</td></tr>
</table></td></tr>
</table>

Table A.13: Search for DAOs use case specification

<table>
<tr><td>Name</td><td colspan="2">Create DAO</td></tr>
<tr><td>Actors</td><td colspan="2">Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing the list of DAOs in the system</td></tr>
<tr><td>Summary</td><td colspan="2">User creates a new DAO</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User clicks on Create DAO button</td></tr>
<tr><td>2</td><td>The application shows the configuration form and description form for creating DAO</td></tr>
<tr><td>3</td><td>User fills and submits the description form to retrieve the IPFS hash of the description</td></tr>
<tr><td>4</td><td>The application automatically fills the IPFS hash into the description hash field in the configuration form</td></tr>
<tr><td>5</td><td>User fills all necessary fields in the configuration form</td></tr>
<tr><td>6</td><td>User clicks on Create button</td></tr>
<tr><td>7</td><td>The application validates the user’s input information and prompts to accept sending the transaction</td></tr>
<tr><td>8</td><td>User accepts sending the transaction</td></tr>
<tr><td>9</td><td>The application waits for the transaction to be confirmed and then redirects the user to view the page of the created DAO</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>7a</td><td>The application validates the user’s input information as incorrect and prompts the user</td></tr>
<tr><td>8a</td><td>User denies sending the transaction</td></tr>
<tr><td>9a</td><td>The transaction has failed, and the application client prompts the user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">A new DAO is created</td></tr>
</table>

Table A.14: Create DAO use case specification

Name	View DAO detail	
Actors	Unconnected Account, Connected Account, Committee Account	
Pre-conditions	User is viewing the list of DAOs in the system	
Summary	User views detailed information of a specific DAO	
Main flow	**Step**	**Action**
	1	User selects a DAO from the list
	2	The application gets detailed information of DAO selected by the user
	3	The application updates the UI to show detailed information of the selected DAO

Table A.15: View DAO detail use case specification

Name	View DAO proposals	
Actors	Unconnected Account, Connected Account, Committee Account	
Pre-conditions	User is viewing the detailed information of a DAO	
Summary	User views the list of proposals of a DAO	
Main flow	**Step**	**Action**
	1	User clicks on Proposals button
	2	The application gets the list of proposals
	3	The application updates the UI to show the list of proposals

Table A.16: View DAO proposals use case specification

Name	View proposal detail	
Actors	Unconnected Account, Connected Account, Committee Account	
Pre-conditions	User is viewing the list of proposals	
Summary	User views detailed information of a specific proposal	
Main flow	**Step**	**Action**
	1	User selects proposal from the list
	2	The application gets detailed information of the proposal selected by the user
	3	The application updates the UI to show detailed information of the selected proposal

Table A.17: View proposal detail use case specification

<table>
<tr><td>Name</td><td colspan="2">Create proposal</td></tr>
<tr><td>Actors</td><td colspan="2">Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing the list of proposals</td></tr>
<tr><td>Summary</td><td colspan="2">User creates a new proposal</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User clicks on Create proposal button</td></tr>
<tr><td>2</td><td>The application shows the configuration form and description form for creating proposal</td></tr>
<tr><td>3</td><td>User fills and submits the description form to retrieve the IPFS hash of the description</td></tr>
<tr><td>4</td><td>The application automatically fills the IPFS hash into the description hash field in the configuration form</td></tr>
<tr><td>5</td><td>User fills all necessary fields in the configuration form</td></tr>
<tr><td>6</td><td>User clicks on Create button</td></tr>
<tr><td>7</td><td>The application validates the user’s input information and prompts to accept sending the transaction</td></tr>
<tr><td>8</td><td>User accepts sending the transaction</td></tr>
<tr><td>9</td><td>The application waits for the transaction to be confirmed and then redirects the user to view the page of the created proposal</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>7a</td><td>The application validates the user’s input information as incorrect and prompts the user</td></tr>
<tr><td>8a</td><td>User denies sending the transaction</td></tr>
<tr><td>9a</td><td>The transaction has failed, and the application client prompts the user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">A new proposal is created</td></tr>
</table>

Table A.18: Create proposal use case specification

<table>
<tr><td>Name</td><td colspan="2">Cast vote</td></tr>
<tr><td>Actors</td><td colspan="2">Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing detailed information of a proposal, and that proposal’s voting period is active</td></tr>
<tr><td>Summary</td><td colspan="2">User casts a vote on the proposal</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User chooses an investment note and an option to cast vote</td></tr>
<tr><td>2</td><td>The application prompts the user to accept sending the transaction</td></tr>
<tr><td>3</td><td>User accepts sending the transaction</td></tr>
<tr><td>4</td><td>The application waits for the transaction to be confirmed and then reloads the UI</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>3a</td><td>User denies sending the transaction</td></tr>
<tr><td>4a</td><td>The transaction has failed, and the application client prompts the user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">The proposal gets one more vote</td></tr>
</table>

Table A.19: Cast vote use case specification

<table>
<tr><td>Name</td><td colspan="2">Queue proposal</td></tr>
<tr><td>Actors</td><td colspan="2">Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing detailed information of a proposal, and that proposal is approved after the voting period</td></tr>
<tr><td>Summary</td><td colspan="2">User queues a proposal for execution</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User clicks on Queue button</td></tr>
<tr><td>2</td><td>The application prompts the user to accept sending the transaction</td></tr>
<tr><td>3</td><td>User accepts sending the transaction</td></tr>
<tr><td>4</td><td>The application waits for the transaction to be confirmed and then reloads the UI</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>3a</td><td>User denies sending the transaction</td></tr>
<tr><td>4a</td><td>The transaction has failed, and the application client prompts the user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">The proposal is queued</td></tr>
</table>

Table A.20: Queue proposal use case specification

<table>
<tr><td>Name</td><td colspan="2">Execute proposal</td></tr>
<tr><td>Actors</td><td colspan="2">Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing detailed information of a proposal, and that proposal is not executed yet after being queued</td></tr>
<tr><td>Summary</td><td colspan="2">User executes a proposal</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User clicks on Execute button</td></tr>
<tr><td>2</td><td>The application prompts the user to accept sending the transaction</td></tr>
<tr><td>3</td><td>User accepts sending the transaction</td></tr>
<tr><td>4</td><td>The application waits for the transaction to be confirmed and then reloads the UI</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>3a</td><td>User denies sending the transaction</td></tr>
<tr><td>4a</td><td>The transaction has failed, and the application client prompts the user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">The proposal is executed</td></tr>
</table>

Table A.21: Execute proposal use case specification

<table>
<tr><td>Name</td><td colspan="2">Cancel proposal</td></tr>
<tr><td>Actors</td><td colspan="2">Connected Account, Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing detailed information of a proposal, and that proposal is not executed yet after being queued</td></tr>
<tr><td>Summary</td><td colspan="2">User cancels a proposal</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User clicks on Cancel button</td></tr>
<tr><td>2</td><td>The application prompts the user to accept sending the transaction</td></tr>
<tr><td>3</td><td>User accepts sending the transaction</td></tr>
<tr><td>4</td><td>The application waits for the transaction to be confirmed and then reloads the UI</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>3a</td><td>User denies sending the transaction</td></tr>
<tr><td>4a</td><td>The transaction has failed, and the application client prompts the user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">The proposal is canceled</td></tr>
</table>

Table A.22: Cancel proposal use case specification

<table>
<tr><td>Name</td><td colspan="2">View list of distributed keys</td></tr>
<tr><td>Actors</td><td colspan="2">Committee Account</td></tr>
<tr><td>Summary</td><td colspan="2">User views list of all generated distributed keys</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User accesses the application</td></tr>
<tr><td>2</td><td>User clicks on Distributed key button</td></tr>
<tr><td>3</td><td>The application gets data of all generated distributed keys and then updates the UI to show the list</td></tr>
</table></td></tr>
</table>

Table A.23: View list of distributed keys use case specification

<table>
<tr><td>Name</td><td colspan="2">Launch funding round</td></tr>
<tr><td>Actors</td><td colspan="2">Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing the list of generated distributed keys</td></tr>
<tr><td>Summary</td><td colspan="2">User uses a distributed key to launch a new funding round</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User clicks on Launch funding round button</td></tr>
<tr><td>2</td><td>The application shows a form for launching a funding round</td></tr>
<tr><td>3</td><td>User fills necessary information and submits</td></tr>
<tr><td>4</td><td>The application validates the user’s input information and prompts to accept sending the transaction</td></tr>
<tr><td>5</td><td>User accepts sending the transaction</td></tr>
<tr><td>6</td><td>The application waits for the transaction to be confirmed, and then updates UI</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>4a</td><td>The application validates the user’s input information as incorrect</td></tr>
<tr><td>5a</td><td>User denies sending the transaction</td></tr>
<tr><td>6a</td><td>The transaction has failed, and the application client prompts the user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">A new funding round is launched</td></tr>
</table>

Table A.24: Launch funding round use case specification

Name	Generate new distributed key	
Actors	Committee Account	
Pre-conditions	User is viewing the list of generated distributed keys	
Summary	User generates a new distributed key	
Main flow	**Step**	**Action**
	1	User clicks on Generate new distributed key button
	2	The application shows a form for generating a distributed key
	3	User fills necessary information and submits
	4	The application validates the user's input information and prompts to accept sending the transaction
	5	User accepts sending the transaction
	6	The application waits for the transaction to be confirmed, and then updates UI
Alternative Flow	**Step**	**Action**
	4a	The application validates the user's input information as incorrect
	5a	User denies sending the transaction
	6a	The transaction has failed, and the application client prompts the user
Post-conditions	A new distributed key is generated	

Table A.25: Generate new distributed key use case specification

Name	View distributed key detail	
Actors	Committee Account	
Pre-conditions	User is viewing the list of generated distributed keys	
Summary	User views detail information of a specific distributed key	
Main flow	**Step**	**Action**
	1	User selects a distributed key from the list
	2	User clicks on Detail button
	3	The application gets detailed information and stored distributed key data in browser local storage of a distributed key selected by user. And then updates the UI to show detailed information

Table A.26: View distributed key detail use case specification

Name	Submit public key contribution	
Actors	Committee Account	
Pre-conditions	User is viewing detailed information of a distributed key, and that key is in public key contribution phase	
Summary	User submits their contribution to establish the public key	
Main flow	**Step**	**Action**
	1	User clicks on Submit button
	2	The application finds in browser local storage to get distributed key data, and then prompts the user to accept sending the transaction
	3	User accepts sending the transaction
	4	The application waits for the transaction to be confirmed, and then updates the UI
Alternative Flow	**Step**	**Action**
	2a	The application does not find stored distributed key data in browser local storage
	3a	User denies sending the transaction
	4a	The transaction has failed, and the application prompts user
Post-conditions	A new public key contribution is submitted to the distributed key	

Table A.27: Submit public key contribution use case specification

Name	Download distributed key data	
Actors	Committee Account	
Pre-conditions	User is viewing detailed information of a distributed key	
Summary	User downloads distributed key data to their device	
Main flow	**Step**	**Action**
	1	User clicks on Download button
	2	The application downloads the file to the user's device

Table A.28: Download distributed key data use case specification

Name	Update distributed key data	
Actors	Committee Account	
Pre-conditions	User is viewing detailed information of a distributed key	
Summary	User updates distributed key data	
Main flow	**Step**	**Action**
	1	User edits information of distributed key data
	2	User clicks on Update button
	3	The application prompts the user that new data will replace old data
	4	User accepts the replacement
	5	The application stores new data to replace old data
Alternative Flow	**Step**	**Action**
	4a	User denies the replacement
Post-conditions	Distributed key data is updated	

Table A.29: Update distributed key data use case specification

Name	Delete distributed key data	
Actors	Committee Account	
Pre-conditions	User is viewing detailed information of a distributed key	
Summary	User deletes distributed key data	
Main flow	**Step**	**Action**
	1	User clicks on Delete button
	2	The application warns the user that this data should be backed up first
	3	User accepts the deletion
	4	The application deletes distributed key data stored in browser local storage
Alternative Flow	**Step**	**Action**
	3a	User denies the replacement
Post-conditions	Distributed key data is deleted	

Table A.30: Delete distributed key data use case specification

Name	View list of distributed key requests	
Actors	Committee Account	
Summary	User views a list of distributed key requests	
Main flow	**Step**	**Action**
	1	User accesses the application
	2	User clicks on Distributed key request button
	3	The application gets data of all distributed key requests, and then updates the UI to show the list

Table A.31: View list of distributed key requests use case specification

Name	View distributed key request detail	
Actors	Committee Account	
Pre-conditions	User views a list of distributed key requests	
Summary	User views detailed information of a specific distributed key request	
Main flow	**Step**	**Action**
	1	User selects a distributed key request from the list
	2	User clicks on Detail button
	3	The application gets detailed information of a distributed key request and stored distributed key data in browser local storage, and then updates the UI to show information

Table A.32: View distributed key request detail use case specification

<table>
<tr><td>Name</td><td colspan="2">Submit result contribution</td></tr>
<tr><td>Actors</td><td colspan="2">Committee Account</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">User is viewing detailed information of a distributed key</td></tr>
<tr><td>Summary</td><td colspan="2">Users submit their contribution to obtain the final result</td></tr>
<tr><td>Main flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>User clicks on Submit button</td></tr>
<tr><td>2</td><td>The application finds in browser local storage to get distributed key data, and then prompts the user to accept sending the transaction</td></tr>
<tr><td>3</td><td>User accepts sending the transaction</td></tr>
<tr><td>4</td><td>The application waits the transaction to be confirmed and then updates the UI</td></tr>
</table></td></tr>
<tr><td>Alternative Flow</td><td colspan="2"><table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>2a</td><td>The application does not find stored distributed key data in browser local storage</td></tr>
<tr><td>3a</td><td>User denies sending the transaction</td></tr>
<tr><td>4a</td><td>The transaction has failed, and then the application prompts the user</td></tr>
</table></td></tr>
<tr><td>Post-conditions</td><td colspan="2">A new result contribution is submitted to the distributed key request</td></tr>
</table>

Table A.33: Submit result contribution use case specification

B. CLASS STORAGE VARIABLE DESCRIPTION

The detailed description of main contracts' storage variable diagrams is presented in the following tables.

Name	Visibility	Data Type	Description
owner	public	address	Address of the FundManager contract
distributedKeyCounter	public	uint256	Count the number of generated distributed keys
distributedKeys	public	mapping (uint256 => IDKG.DistributedKey)	Mapping from distributed key ID to distributed key data
tallyTrackers	public	mapping (bytes32 => IDKG.TallyTracker)	Mapping from distributed key request ID to tally tracker data
round2Verifiers	public	IVerifier	The Verifier contract for the round 2 contribution
fundingVerifiers	public	mapping (uint256 => IVerifier)	Mapping from dimension to corresponding contract Verifier for the investment process
votingVerifiers	public	mapping (uint256 => IVerifier)	Mapping from dimension to corresponding contract Verifier for the vote casting process

tallyContribution-Verifiers	public	mapping (uint256 => IVerifier)	Mapping from dimension to corresponding contract Verifier for the tally contribution
resultVerfiers	public	mapping (uint256 => IVerifier)	Mapping from dimension to corresponding contract Verifier for the final result contribution

Table B.1: The DKG class storage variable description

Name	**Visibility**	**Data Type**	**Description**
founder	public	address	Address of the wallet account created the FundManager contract
numberOfCommittees	public	uint8	The number of committee members
threshold	public	uint8	The threshold of committee members needed to participate in tally contribution to obtain the final result
fundingRoundInProgress	public	bool	Indicate whether a funding round is launching or not
daoManager	public	address	Address of the DAOManager contract

config	public	IFundManager.FundingRoundConfig	Config parameters for the funding round
isCommittee	public	mapping (address => bool)	Mapping whether a wallet address is a committee member or not
requests	public	mapping (bytes32 => IDKGRequest.Request)	Mapping from distributed key request ID to request data
fundingRounds	public	mapping (uint256 => IFundManager.FundingRound)	Mapping from funding round ID to funding round data
fundingRoundCounter	public	uint256	Count the number of launched funding rounds
fundingRoundQueue	public	Queue	The Queue contract is used to store DAO in the queue, waiting for the funding round
dkgContract	public	IDKG	The DKG contract is used to manage distributed keys

Table B.2: The FundManager class storage variable description

Name	**Visibility**	**Data Type**	**Description**
fundManager	public	IFundManager	The FundManager contract
dkg	public	IDKG	The DKG contract
daoCounter	public	uint256	Count the number of created DAOs

distributedKeyID	public	uint256	ID of the distributed key used in governance process
daos	public	mapping (uint256 => address)	Mapping from index of DAO to address of DAO
deposits	public	mapping (address => uint256)	Mapping from the DAO creator's wallet address to the amount they deposited
requiredDeposit	private	uint256	The required value deposit to create DAO
admin	private	address	Address of admin of the DAOManager contract

Table B.3: The DAOManager class storage variable description

Name	**Visibility**	**Data Type**	**Description**
descriptionHash	public	bytes32	IPFS hash of the DAO's information, converted from Base58 string to 32-byte string
VOTE_OPTIONS	public	uint256	Number of options in the vote casting process
config	public	IDAO.Config	Config parameters for the governance process
proposalCounter	public	uint256	Count the number of the proposed proposal

proposalIDs	public	mapping (uint256 => uint256)	Mapping from index of proposal to ID of proposal
proposals	public	mapping (uint256 => IDAO.Proposal)	Mapping from ID of proposal to proposal data
actions	public	mapping (uint256 => IDAO.Action[])	Mapping from ID of proposal to the actions to be executed when the proposal is approved
descriptions	public	mapping (uint256 => bytes32)	Mapping from ID of proposal to IPFS hash of the proposal's information, converted from Base58 string to 32-byte string
requests	public	mapping (bytes32 => IDKGRequest.Request)	Mapping from distributed key request ID to request data
fundManager	private	FundManager	The FundManager contract
dkg	private	IDKG	The DKG contract
distributedKeyID	private	uint256	ID of the distributed key used in governance process
nullifierHashes	private	mapping (uint256 => mapping (uint256 => bool))	The mapping is used to store which nullifier hash is used to cast vote on proposal

queuedTransactions	public	mapping(bytes32 => bool)	Mapping indicates what queued transaction is executed

Table B.4: The DAO class storage variable description

C. DATABASE DESCRIPTION

The detailed descriptions for fields in database collections are presented in the following tables.

In the EventRegistry collection, each document represents a specific type of event that the application server will listen for. These events are defined by the contract name, contract address, event signature hash, and the chain ID. To uniquely identify each event type, the values of these attributes are combined and hashed to create an event registry ID. This event registry ID serves as a unique identifier for each event type in the system.

Field	Data Type	Description
eventRegistryID	String	A unique string to identify the event registry, that is a hash value of eventSignatureHash, chainID, contractName and contractAddress
eventSignatureHash	String	Hash of event signature
chainID	String	Network's ID
contractName	String	Name of the target contract
contractAddress	String	Address of the target contract

Table C.1: EventRegistry collection description

For each type of event registered in the EventRegistry collection, the application server will actively listen to these event types and record the events into the Event collection. Each event is defined by its transaction hash, block number, and the data emitted along with the event (topics). These details are captured and stored in the Event collection for further processing and querying.

Field	Data Type	Description
eventRegistryID	String	A unique string to identify the event registry, that is a hash value of eventSignatureHash, chainID, contractName and contractAddress

transactionHash	String	Hash of the transaction which the event was emitted
blockNumber	String	Block number which the event was emitted
topics	String[]	The data is emitted along with the event

Table C.2: Event collection description

For every user's investment, an event is emitted, containing the index and commitment value added to the Merkle tree. This specific event type is pre-registered in the EventRegistry collection. When the application server listens to this event, it not only adds the event to the Event collection but also extracts essential information such as the index and commitment. This information is then stored in the MerkleLeaf collection. Subsequently, the application server builds a Merkle tree to facilitate user queries for the path of the leaf they own. This path is crucial for their participation in the governance process, as it is used to prove that they have invested in a certain DAO.

Field	**Data Type**	**Description**
eventRegistryID	String	A unique string to identify the event registry, that is a hash value of eventSignatureHash, chainID, contractName and contractAddress
transactionHash	String	Hash of the transaction which the event was emitted
index	String	Index of the Merkle leaf
commitment	String	Value of the Merkle leaf

Table C.3: MerkleLeaf collection description

D. PERFORMANCE SPECIFICATION OF ZERO-KNOWLEDGE PROOFS

In the algorithm design of the proposed model, five processes require the use of Zero-Knowledge Proofs:

- ZKP for round 2 contribution in the public key contribution process.
- ZKP for the investment process.
- ZKP for the governance process.
- ZKP for tally contribution in the result contribution process.
- ZKP for final result contribution in the result contribution process.

For the prototype demonstration developed in this thesis, the ZKPs are constructed with the following input parameters:

- The number of committee members is 5.
- The threshold of committee members required to proceed with the result contribution process is 3.
- The investment process allows each funding round to have participation from 3 DAOs. In the governance process, there are 3 voting options. Hence, the number of dimensions set up for ZKPs in the investment and governance processes is 3.
- Arithmetic operations are performed with 256-bit numbers.

In the context of this thesis, ZKPs are generated using the Groth16 ZK-SNARK protocol [31]. The complexity of the ZKPs is measured by the number of non-linear constraints. On the client side, to generate ZKPs, "zkey" and "wasm" files are required. The following table shows the specifications of the ZKPs.

ZKP	**Non-linear constraints**	**Zkey file size**	**Wasm file size**
ZKP for round 2 contribution in the public key contribution process	67088	37.3 MB	343.0 KB

ZKP for the investment process	18347	10.0 MB	149.0 KB
ZKP for the governance process	24248	12.7 MB	629.3 KB
ZKP for tally contribution in the result contribution process	42844	23.0 MB	249.5 KB
ZKP for final result contribution in the result contribution process	26382	13.1 MB	161.5 KB

Table D.1: Zero-Knowledge Proof specifications

Zero-Knowledge Proofs are verified on-chain through Verifier contracts. The gas consumption during ZKP verification is primarily influenced by two factors: the number of required public inputs and the gas consumed to invoke the "verify" function of the Verifier contract. The table below presents the quantity of public inputs and the corresponding gas consumption for the "verify" function. It should be noted that the public inputs encompass both the data provided by the user during the transaction and the data read from the contract's storage. However, the costs associated with sending public inputs and reading from contract storage are not included in these statistics.

ZKP	**Public inputs**	**Gas**
ZKP for round 2 contribution in the public key contribution process	30	$\approx 450K$
ZKP for the investment process	19	$\approx 360K$
ZKP for the governance process	18	$\approx 350K$
ZKP for tally contribution in the result contribution process	31	$\approx 457K$
ZKP for final result contribution in the result contribution process	30	$\approx 450K$

Table D.2: Gas consumption for ZKP verification