DAO-FL: Enabling Decentralized Input and Output Verification in Federated Learning with Decentralized Autonomous Organizations

Umer Majeed 1, Sheikh Salman Hassan 2, Zhu Han 2, and Choong Seon Hong 2

1Kyung Hee University
2Affiliation not available

December 7, 2023

Abstract

Federated Learning (FL) has emerged as a decentralized machine learning paradigm that facilitates collaborative training of a global model (GM) across multiple devices while maintaining data privacy. Traditional FL systems suffer from centralized validation of local models and GM updates, compromising transparency and security. In this paper, we propose DAO-FL, a smart contract-based framework that leverages the power of Decentralized Autonomous Organizations (DAOs) to address these challenges. DAO-FL introduces the concept of DAO Membership Tokens (DAOMTs) as a governance tool within a DAO. DAOMTs play a crucial role within the DAO, facilitating members’ enrollment and expulsion. Our framework incorporates a Validation-DAO for decentralized input verification of the FL process, ensuring reliable and transparent validation of local model uploads. Additionally, DAO-FL employs a multi-signatures approach facilitated by an Orchestrator-DAO to achieve partially decentralized GM updates, and thus decentralized output verification of the FL process. We present a comprehensive system architecture, detailed execution workflow, implementation specifications, and qualitative evaluation for DAO-FL. Evaluation under threat models highlights DAO-FL’s out-performance against traditional-FL (FedAvg), effectively countering input and output attacks. DAO-FL excels in scenarios where decentralized input and output verification are crucial, offering enhanced transparency and trust. In conclusion, DAO-FL provides a compelling solution for FL, reinforcing the integrity of the FL ecosystem through decentralized decision-making and validation mechanisms.
DAO-FL: Enabling Decentralized Input and Output Verification in Federated Learning with Decentralized Autonomous Organizations

Umer Majeed, Sheikh Salman Hassan, Student Member, IEEE, Zhu Han Fellow, IEEE, and Choong Seon Hong, Senior Member, IEEE

Abstract—Federated Learning (FL) has emerged as a decentralized machine learning paradigm that facilitates collaborative training of a global model (GM) across multiple devices while maintaining data privacy. Traditional FL systems suffer from centralized validation of local models and GM updates, compromising transparency and security. In this paper, we propose DAO-FL, a smart contract-based framework that leverages the power of Decentralized Autonomous Organizations (DAOs) to address these challenges. DAO-FL introduces the concept of DAO Membership Tokens (DAOMTs) as a governance tool within a DAO. DAOMTs play a crucial role within the DAO, facilitating members’ enrollment and expulsion. Our framework incorporates a Validation-DAO for decentralized input verification of the FL process, ensuring reliable and transparent validation of local model uploads. Additionally, DAO-FL employs a multi-signatures approach facilitated by an Orchestrator-DAO to achieve partially decentralized GM updates, and thus decentralized output verification of the FL process. We present a comprehensive system architecture, detailed execution workflow, implementation specifications, and qualitative evaluation for DAO-FL. Evaluation under threat models highlights DAO-FL's out-performance against traditional-FL (FedAvg), effectively countering input and output attacks. DAO-FL excels in scenarios where decentralized input and output verification are crucial, offering enhanced transparency and trust. In conclusion, DAO-FL provides a compelling solution for FL, reinforcing the integrity of the FL ecosystem through decentralized decision-making and validation mechanisms.

Index Terms—Decentralized autonomous organization, Decentralized input verification, Decentralized output verification, Federated Learning, DAO membership tokens, Non-transferable tokens, Smart contract, Soul-bound tokens, Structured transparency.

I. INTRODUCTION

In the dynamic landscape of Web3 and blockchain technology, several disrupting technologies have emerged, transforming the way we interact and conduct digital transactions. Decentralized autonomous organizations (DAOs) represent innovative organizational structures that operate autonomously through blockchain technology and smart contracts [3], [4], eliminating the need for centralized control. DAOs have the potential to revolutionize traditional hierarchical management paradigms, reducing communication, administration, and collaboration expenses within organizations [5]. Another groundbreaking innovation is Soul-Bound tokens (SBTs) [6], [7], which are non-transferable tokens (NTTs) intrinsically linked to specific addresses, serving as unique digital identities and reputation indicators. SBTs provide enhanced security and authenticity in various applications, including identity verification and exclusive ownership rights. Furthermore, Non-fungible Tokens (NFTs) [8], [9] have emerged as a game-changer in the art and gaming industries. These tokens represent distinct and indivisible digital assets, enabling provable ownership and authenticity for digital art, collectibles, and virtual assets.

Federated learning (FL) [10]–[12] as a distributed artificial intelligence (DAI) technique facilitates the collaborative learning of a highly accurate deep learning model by aggregating local models into a global model (GM) through the FL process. The FL process can be viewed as an information flow within the context of Structured Transparency (ST) [13], where local models serve as inputs and the global model is the output for each global iteration [14]. Input and output verification are two crucial components of structured transparency. Input verification plays a pivotal role in guaranteeing the validation of inputs of information flow, ensuring they align seamlessly with the requirements. Output verification ensures the integrity of the information flow’s output, validating policy compliance and preventing tampering. Decentralized input and output verification are novel concepts that serve as a robust mechanism, distributing these verification processes across multiple entities, and negating the need for reliance on a single entity.

In our previous study [15], we addressed the challenges of FL by introducing FL-Incentivizer, which utilized FL-Tokens to reward trainer devices. However, FL remains a resource-intensive process, often requiring several days of training for the initial deployable GM and continuous updates over months. In FL-Incentivizer, local model submissions were validated by a single central authority, the FL Task Publisher Contract’s owner (FLTPCO), who also aggregated and uploaded the GM. Centralization of the server-side FL process raises concerns, as a single incorrect GM update can jeopardize the accuracy of the latest GM.

To tackle these challenges, in this study, we propose the DAO-FL framework, which integrates DAOs and a multi-signature [16] contract with FL to enable decentralized input and output verification of FL process, that is decentralized validation of local models and global model. By employing
DAOs, we distribute the verification process across multiple participants, ensuring transparency and mitigating the risk of central authority manipulation. The following is a summary of our contributions:

- We introduce DAO Membership Tokens (DAOMTs) which are SBTs, NTTs, NFTs. DAOMTs have specific characteristics such as being burnable, mintable, and limited to a maximum balance of one per address. They are also part of a token collection and serve as a means for governance in systems utilizing DAOs.
- We design decentralized schemes for member enrollment and member expulsion within a DAO, thereby enabling candidates to join a DAO and to kick out malicious or inactive DAO members respectively.
- We present a comprehensive system architecture and detailed execution workflow of DAO-FL, which is a smart-contract-enabled framework for partially decentralized orchestration of FL process and decentralized validation of local model uploads (LMUs). DAO-FL employs two DAOs, Orchestrator-DAO (ODAO) and Validation-DAO (VDAO) for its operations.
- DAO-FL validates and rewards LMUs via the Validation-DAO to ensure decentralized input verification of FL process.
- DAO-FL uses a multi-signatures contract to validate the GM update and partially decentralized orchestration of FL process via the Orchestration-DAO satisfying decentralized output verification.
- We present comprehensive implementation specifications, including the smart contract code. Furthermore, we provide detailed deployment information, an evaluation on threat models, and a qualitative evaluation of DAO-FL.

The remaining sections of this article are structured as follows: Section II provides a comprehensive review of related literature pertaining to our study. In Section III, we explore the relevant preliminaries necessary for understanding our work. The system architecture and execution workflow of DAO-FL is expounded upon in Section IV. Detailed information regarding the implementation specifications, deployment details, evaluation on threat models, and qualitative evaluation of DAO-FL can be found in Section V. Finally, we conclude our paper in Section VI.

II. RELATED WORK

Bluemke et al. in [17] explored the significance of data privacy-enhancing technologies in the realm of AI governance. They highlighted the progress made in balancing privacy and performance during data exchange and analysis, emphasizing the value of structured transparency. Thus, enabling controlled information flow, addressing who, when, and how information should be accessible, and ensuring efficient collaboration while reducing data misuse risks.

Majeed et al. in [15] proposed the ST-BFL framework, utilizing homomorphic encryption, FL-aggregators, FL-verifiers, and a smart contract to satiate components of structured transparency [13] for FL process. Homomorphic encryption ensures input privacy, and FL-verifiers validate the global model for output verification. However, ST-BFL lacks local model validation as it prioritizes input privacy over verification. Additionally, detailed information on authentication and authorization of FL-verifiers, vital for output verification, is missing. In contrast, DAO-FL focuses on DAO-based input and output verification of the FL process.

Majeed et al. proposed FL-Incentivizer in [15], incentivizing device participation in FL with FL-Tokens and enabling ownership rights to a global model via FL-NFT. FL-Incentivizer employs an FLTPCO for local models’ validation and global model updates, ensuring input and output verification centrally. However, this work extends FL-Incentivizer by decentralizing the input and output verification processes through DAOs and a multi-signature contract. Table I compares the structured transparency components of ST-BFL, FL-Incentivizer, and the proposed work “DAO-FL”.

Lunesu et al. in [18] presented a practical application of SBTs for COVID vaccine certification using the decentralized Vaccine System DApp, powered by blockchain. The research explains system components, smart contract, user interface, and database, while also addressing the roles and actions of citizens and administrators within the system. It emphasizes the potential of SBTs in establishing a reliable decentralized society, and self-sovereign identity (SSI). They also discuss associated challenges and privacy concerns. [19] proposed an innovative approach that utilizes SBTs to encode individuals’ affiliations and academic credentials in a decentralized network. The system employs off-chain storage, smart contracts, and cryptographic technologies to enhance privacy and security, and offers a trustworthy environment for stakeholders, providing a robust and confidential alternative to centralized academic credential verification.

Diallo et al. in [20] presented an eGov-DAO system to enhance e-government transaction efficiency, transparency, and security. Through the implementation of a decentralized autonomous organization and smart contracts, the system automates transactions, thereby reducing errors and uncertainty while ensuring accountability and mitigating corruption risks. Although the study offers a comprehensive design and potential advantages, additional research is essential to assess the practical applicability of the system in real-world government operations.

Aitzhan et al. in [16] presented a decentralized energy trading system utilizing multi-signature transactions on the blockchain. Multi-signature ensures transaction security, requiring 2 out of 3 signatures to spend a token and preventing mediators from controlling transactions. It protects against theft by requiring multiple signatures for validity. This approach fosters a secure and trustworthy energy trading system without reliance on trusted third parties, promoting a more decentralized and competitive environment for energy trade.

III. PRELIMINARIES

This section offers an overview of the technologies utilized in the design and implementation of the DAO-FL framework.
A DAO is an internet-native digital equivalent to traditional companies in the physical world. DAOs, in essence, allow members to create and vote on governance decisions that are specifically made by the boards of directors or executives in conventional companies. A DAO operates autonomously following predefined business logic contained in its smart contract to accomplish a collective mission of DAO’s community with token economy-based incentives. “The DAO,” launched in 2016, was the world’s first DAO and raised $150 million in Ether (ETH), making it one of the largest digital crowdfunding projects. Some other popular examples of DAOs are DigixDAO, Aragon, Steemit, etc. DAO has an initial creation phase in which typically E0As send Ethers to the DOA smart contract’s address and DOA tokens are created and assigned to those E0As as proof of DOA’s membership and voting rights. DAOs make it possible to accomplish a broad spectrum of objectives, encompassing activities such as delivering services, generating targeted funds, owning and managing smart assets, coordinating with other autonomous software, and facilitating cooperation among various stakeholders.

B. Structured Transparency

Structured transparency is a framework designed to address the tradeoff between privacy and transparency for information flows. It consists of five components: input privacy, output privacy, input verification, output verification, and flow governance. Input privacy refers to the ability to process hidden information without revealing it to others, while output privacy allows receiving and contributing to information flows without revealing sensitive input. Input verification involves ensuring the integrity of the input, while output verification ensures that the output has not been tampered with. Flow governance refers to the overall management and control of information flows.
the information flow. To satisfy each component, certain requirements must be met. Input privacy requires mechanisms to process information without revealing it, while output privacy necessitates preventing the inference of sensitive input from the output. Input verification requires methods to ensure the integrity and authenticity of the input, and output verification requires techniques to prove that the output has not been tampered with. Flow governance requires effective management and control mechanisms to govern the entire information flow.

C. Multi-signature wallet

A multi-signature (also known as a “multisig”) wallet is a type of digital wallet that enhances security by requiring more than one person to sign off on a transaction before it can be executed [22]. In multi-signature wallets, the execution of transactions is governed by the quorum quotient, which is represented by the m-of-n ratio. This ratio refers to the minimum number of signatories required to sign a transaction, expressed as a fraction of the total number of registered signatories. For instance, a 3-of-5 wallet mandates that at least three out of five designated signers must approve a transaction for it to be processed. This can be useful in cases where multiple parties need to agree on a transaction, or where added security is desired to protect against unauthorized transactions. Multi-signature wallets are commonly used in a variety of contexts, including financial transactions, corporate governance, and even in the management of cryptocurrency exchanges. Multi-signature wallets are commonly implemented using smart contracts to enforce the requirement of multiple signatures for transaction authorization.

IV. PROPOSED FRAMEWORK

This section offers a comprehensive explanation of the proposed system architecture and execution workflow within the DAO-FL framework. The system architecture, depicted in Fig. 1, comprises three blocks: the administrative block, the decentralized block, and the FL-trainer block.

The administrative block consists of pivotal stakeholders in the DAO-FL framework including a regulator, FL-task-publisher (FLTP), ODAO, and VDAO. These entities govern and orchestrate various aspects of the DAO-FL ecosystem. The regulator governs the FL ecosystem, deploys the FLNFTC, and standardizes FLNFTs metadata. Throughout our study, we denote this regulatory entity as Regulator. When an entity adopts the DAO-FL framework to train an FL model, it must deploy specific smart contracts, namely ODAOC, VDAOC, DAOFLC, and MultiSigC, customized exclusively for the specific FL task. This entity is referred to as the FL-task-publisher (FLTP). The ODAO, a DAO overseeing the FL process, comprises multiple members (ODAOM). These ODAO members (ODAOMs) are responsible for approving proposals from the FLTP and possess the ability to aggregate local models. Similarly, the VDAO, as a decentralized entity, verifies the local models submitted by FL-Trainers by utilizing its VDAO members (VDAOMs), where each VDAOM has the capability to validate local models relevant to the given FL task.

The decentralized block consists of essential components: FL-NFT contract (FLNFTC), ODAO contract (ODAOC), ODAO Membership Token contract (ODAOMTC), Validation-DAO contract (VDAOC), Validation-DAO Membership Token contract (VDAOMTC), DAO-FL contract (DAOFLC), Multi-Signature contract (MultiSigC), FL Token contract (FLTo-
The FLNFT, derived from ERC-721 standard and deployed by the regulator, enables the tokenization of FLTs GMs. ODAOC manages membership operations within the DAO, while ODAOMTC mints ODAO Membership Tokens (ODAOMTs) for ODAO members (ODAOMs). Similarly, VDAOC handles member-related operations in the VDAO, and VDAOMTC generates VDAO Membership Tokens (VDAOMTs) for VDAO members. A comprehensive explanation of DAO Membership Tokens (DAOMTs) is provided in Section IV-A. Both ODAOMTC and VDAOMTC are customization of ERC-721 standard. It is worth noting that the ODAOMTC and VDAOMTC are deployed upon the deployment of the ODAO and VDAO respectively. The DAOFLC orchestrates the FL process for a given FL task, supported by MultiSigC for decentralized execution. The MultiSigC, in turn, facilitates the decentralized execution of FL operations within the DAOFLC by collecting multiple signatures from ODAO, FLTokenC, deployed by DAOFLC, derived from ERC-20, manages FL-Tokens specific to each FL task. IPFS serves as a decentralized file storage system for metadata, local models, and global models.

The FL-Trainers block consists of multiple FL learners, with each FL-Trainer representing a participating device or client in the FL process. We denote the FL-Trainer for the $i^{th}$ client in the $t+1^{th}$ generation interval of FL task as FLTrainer$_{i,t+1}$. The FL-Trainer retrieves and downloads the $GM_{t+1}$ and generates its local model upload $LMU_{i,t+1}$ utilizing its respective local dataset $D_{i,t+1}$.

Besides the previously mentioned entities, the system architecture also includes two crucial components: FLNFTs and FL-Tokens. Each FL-NFT, denoted as $FLNFT$, is an ERC-721 compliant dynamic Non-Fungible Token (NFT) associated with an FL task. It possesses a distinct numeric identity, referred to as $tokenURI$ that links to the metadata of the current GM for the FL task [13]. Additionally, the FLNFT includes the $GMCID$ property, which represents the IPFS Content Identifier (CID) of the most recent GM. Crucially, the FL-NFT contains the address of the corresponding DAOFLC, known as $OrchestratorAddress$. The $tokenURI$, $GMCID$, and $OrchestratorAddress$ for each FL-NFT are distinctive. The FLTP acts as the rightful owner of the FLNFT, facilitating the benefits of GM commercialization and tokenization [15]. Furthermore, FL-Tokens, symbolized as $FLToken$, conform to the ERC-20 standard and are awarded to FL-Trainers within the FL process. These FL-Tokens, minted within the same FLTokenC, are interchangeable, representing fungibility [15].

The aforementioned entities constitute the core components of the DAO-FL framework. An overview of subsequent subsections is presented as follows. In Section IV-A, we introduce the novel concept of DAO Membership Tokens (DAOMTs). Section IV-B proposes a member enrollment scheme for adding new members to a DAO, while Section IV-C presents a member expulsion scheme to address inactive or malicious members. Furthermore, Section IV-D outlines a mechanism for transferring ODAOOC or VDAOOC to a new proprietor. In Section IV-E, a scheme is proposed for partially decentralized FL process orchestration in the DAOFLC using a Multi-Signature Contract (MultiSigC). Additionally, Section IV-F details a comprehensive execution workflow for the DAO-FL framework, orchestrating the FL process from initial setup to completing a full global iteration. Lastly, Section IV-G delves into GM commercialization, involving the transfer of FL-NFT and contracts ownership to the new proprietor.

### A. DAO Membership Tokens (DAOMTs)

In this subsection, we introduce the concept of DAO Membership Tokens (DAOMTs). DAOs are decentralized organizations that operate autonomously on a blockchain, governed by their members through a voting-based decision-making process. DAOMTs are a specific type of token designed to represent the membership of entities within a DAO. They are classified as NFTs and SBTs [6], meaning they cannot be traded or transferred on a marketplace. Additionally, DAOMTs are categorized as NFTs, with each token being unique. These tokens can be minted or burnt, denoting controlled creation and destruction, respectively. Typically, members are limited to holding one token per address, thereby restricting the maximum balance to one token per address. DAOMTs can be grouped together with other tokens to represent various levels or types of membership, forming a collection. They can be utilized for the governance of DAO-based systems, granting members the right to vote on proposals and participate in decision-making processes regarding the organization’s direction and operation. Ultimately, DAOMTs contribute to a more democratic and decentralized approach to decision-making within a DAO.

### B. Membership Enrollment in ODAO and VDAO

The process of becoming a member of ODAO or VDAO follows a similar procedure. Hence, in this section, we will describe the steps for joining a DAO through a DAO contract (DAOC), which is inherited by both ODAO and VDAO. After the creation of the DAO, it is essential to have existing members. Let us denote the existing member within the DAO as $DAO_M_i \in DAO$. The simplified sequential outline for joining a DAO is outlined below:

1. **Step 1:** When a new candidate seeks to join the DAO, a current member of the DAO, denoted as $DAO_M_p$, will initiate a “proposeJoin” transaction to the DAOC. This transaction includes the candidate’s address as an argument, effectively proposing its inclusion into the DAO.

2. **Step 2:** To process the “proposeJoin” transaction, the DAOC first validates that the submitter, $DAO_M_p$, possesses a DAOOMT, thus implementing a safeguard mechanism against potential spam transactions.

3. **Step 3:** If the candidate is not a current member of DAO and no existing “Join Proposal” exists for it, a new “Join Proposal” (joinProposal) is initiated. The joinProposal includes the candidate’s address and is proposed by $DAO_M_p$. A boolean flag called “open”
Algorithm 1: Membership Enrollment via DAOC

**Caller:** DAOM_i

1. procedure proposeJoin(address candidate)
2.   Ensure DAOM_i holds a DAOMT
3.   if candidate \notin DAO and JoinProposals[candidate].open == false then
4.     Create new joinproposal
5.     Set JP.proposer = DAOM_i
6.     Set JP.candidate = candidate
7.     Set JP.open = true
8.     Set JP.approvalvotes = JP.denialvotes = 0
9.     Set JP.voters = empty AddressSet
10.   Add JP to JoinProposals
11. end if
12. end procedure

**Caller:** DAOM_i

13. procedure voteJoin(address candidate, bool vote)
14.   Ensure DAOM_i holds a DAOMT
15.   Set JP = JoinProposals[candidate]
16.   if JP.open == true and DAOM_i \notin JP.voters then
17.     if vote=true then
18.       Add Approval vote for DAOM_i
19.     else
20.       Add Deny vote for DAOM_i
21.     end if
22.   end if
23.   Count JP.approvalvotes and JP.denialvotes
24.   quorum = 60% \times n(DAOMT)
25.   if JP.denialvotes > quorum then
26.     Mint DAOMT for candidate
27.     Set JP.open = false
28.   else if JP.denialvotes > quorum then
29.     Set JP.open = false
30.   end if
31. end if
32. end procedure

is set to true to indicate that joinproposal is currently being processed and has not been accepted or rejected. The approvalvotes and denialvotes fields of the joinproposal are initialized to 0, indicating no approval or denial votes have been cast yet. The set of voters for the joinproposal is initially empty, indicating no DAOM_i \in DAO have voted for the joinproposal yet.

- Step 4: Subsequently, the joinproposal is then stored in a mapping data structure called the JoinProposals with candidate as the index to associate the candidate with their corresponding joinproposal.

Steps 1-4 are combined in the proposeJoin procedure presented in Algorithm 1. Following that, the current DAO members proceed with the process of voting to accept or reject the joinproposal. The voting procedure consists of the following steps:

- Step 5: When a DAO intends to vote on a joinproposal, they will initiate a “voteJoin” transaction within DAOC, providing the candidate’s address and a boolean variable, denoted as “vote”, representing their voting decision. The value “true” signifies the approval of DAOM_i for the joinproposal, while “false” indicates disapproval.

- Step 6: To prevent spam transactions, the DAOC will first verify that the sender DAOM_i of the “voteJoin” transaction possesses a valid DAOMT.

- Step 7: If an open joinproposal exists for candidate and DAOM_i has not yet voted on it, their vote is added to the list of votes against the joinproposal. The total number of approval and denial votes are tallied. The quorum is defined as 60% of the total supply of DAOMTC. If the total number of approval votes exceeds the quorum, the DAO proceeds to mint a DAOMT for the candidate through the DAOMTC. Subsequently, the joinproposal is closed by setting the “open” flag to false. However, if the total number of denial votes exceeds the quorum, the join proposal is rejected by setting the “open” flag to false.

Steps 5-7 are consolidated in the procedure voteJoin as depicted in Algorithm 1. Fig. 2 visually illustrates the process.
of joining a DAO.

### C. Member Expulsion in ODAO and VDAO

The presence of non-active or malicious members in a DAO raises concerns and calls for their expulsion. Non-active members fail to actively participate in the orchestration of the FL process, while malicious members engage in endorsing incorrect or inaccurate updates. The procedure for removing members from both ODAO and VDAO is consistent, and a kick-out mechanism is introduced to address these non-active or malicious individuals. The simplified kick-out mechanism encompasses the following sequential steps:

- **Step 1:** When a DAO member, identified as $DAOM_p$, determines that another member (referred as candidate) should be expelled, $DAOM_p$ initiates the kick-out process by submitting a “proposeKick” transaction to the DAOC. This transaction includes the address of the targeted candidate as an argument.
- **Step 2:** $DAOC$ verifies if the submitter ($DAOM_p$) of the “proposeKick” transaction holds a $DAOMT$ to prevent spam transactions.
- **Step 3:** If the candidate is an active member of the DAO and there is no existing “Kick Proposal” in progress for the candidate, a new “Kick Proposal” ($kickproposal$)

**Algorithm 2**: Member Expulsion via $DAOC$

**Caller:** $DAOM_p$

1. procedure proposeKick(address candidate)
2. Ensure $DAOM_p$ holds a $DAOMT$
3. if candidate ∉ DAO and $KickProposals[candidate].open == false$ then
4. Create new kickproposal $kickproposal=KP$
5. Set $KP.proposer = DAOM_p$
6. Set $KP.candidate = candidate$
7. Set $KP.open = true$
8. Set $KP.approvalvotes = KP.denialvotes = 0$
9. Set $KP.voters = empty AddressSet$
10. Add $KP$ to $KickProposals$
11. end if
12. end procedure

**Caller:** $DAOM_i$

1. procedure voteKick(address candidate, bool vote)
2. Ensure $DAOM_i$ holds a $DAOMT$
3. Ensure candidate holds a $DAOMT$
4. Set $KP = KickProposals[candidate]$  
5. if $KP.open == true$ and $DAOM_i ∉ KP.voters$ then
6. if vote=true then
7. Add Approval vote for $DAOM_i$
8. else
9. Add Deny vote for $DAOM_i$
10. end if
11. Count $KP.approvalvotes$ and $KP.denialvotes$
12. $quorum = 60% * n(DAOMT)$
13. if $KP.approvalvotes > quorum$ then
14. Burn $DAOMT$ owned by candidate
15. Set $KP.open = false$
16. else if $KP.denialvotes > quorum$ then
17. Set $KP.open = false$
18. end if
19. end if
20. end procedure

is initiated. The candidate is specified as the target of the kickproposal, and $DAOM_i$ assumes the role of the proposer. The kickproposal is marked as “open” to indicate its ongoing status, awaiting acceptance or rejection. Initially, the kickproposal has no approval or denial votes, so both approvalvotes and denialvotes are set to zero. The set of voters for the kickproposal is empty, indicating that no DAO members ($DAOM_i ∈ DAO$) have cast their votes in support or against the kickproposal at this stage.

- **Step 4:** The kickproposal is added to a mapping structure called KickProposals, with the candidate serving as the index.
Steps 1-4 are consolidated into the `proposeKick` procedure outlined in Algorithm 2. The voting process, executed by existing DAO members for `kickproposal`, involves the following steps:

- **Step 5:** In the DAO’s kick proposal voting process, a DAO\(_i\) can cast their votes through a transaction called “voteKick” to DAOC. It includes the candidate’s address and a boolean variable, “vote”, indicating approval (true) or disapproval (false), as arguments. This allows each member to participate and express their stance on the kick proposal actively.
- **Step 6:** The DAOC validates “voteKick” transactions to prevent spam and ensure legitimacy. It verifies that both the submitter (DAOM\(_i\)) and candidate hold a DAOMT token as proof of membership.
- **Step 7:** If the `kickproposal` is open for a specific candidate and DAO\(_i\) has not yet voted, their vote is added against the `kickproposal`. The total approval and denial votes are counted. If the approval votes exceed the quorum, DAOC burns the DAOMT owned by the candidate, and the proposal is closed by setting a “open” flag to false. If the denial votes surpass the quorum, the proposal is rejected by setting a “open” flag to false.

Steps 5-7, for a kick proposal, are summarized in procedure `voteKick` in Algorithm 2. The sequential flow for kicking out a DAO’s member is depicted in Fig. 3.

### D. Transferring ODAO and VDAO

The FLTP possesses ownership of the GM, which is authenticated through ownership of the corresponding FL-NFT in the FLNFTC. Additionally, the FLTP holds ownership of the ODAO and VDAO. When transferring ownership of the FLNFT to a successor proprietor, the ownership of ODAO and VDAO must also be transferred accordingly. The steps for ownership transfer of the DAOC, the parent contract of ODAO and VDAO, are summarized in the procedure `transferOwnership` of Algorithm 3 and are as follows:

- **Step 1:** The current owner (FLTP) initiates a “transfer ownership” transaction to DAOC with the address of the new owner (`newOwner`) as an argument.
- **Step 2:** The DAOC verifies that the new owner `newOwner` is different from the previous owner, and proceeds to transfer ownership of DAOC to `newOwner`. If `newOwner` is not already a member of the DAO, a DAOMT is minted for `newOwner`, while the DAOMT owned by `oldOwner` is burned, ensuring scarcity of DAOMT.

### E. Partially Decentralized Orchestration of FL process in DAOFLC through Multi-Signature Contract

The implementation of a multi-signature wallet commonly involves a Multi-Signature Contract (MultiSigC). This contract collects the required signatures or votes from designated individuals on a transaction. Once the accumulated votes surpass the predetermined quorum, the MultiSigC executes the transaction within the target contract. In the DAO-FL framework, the MultiSigC aggregates votes from ODAOMs to enable decentralized approval for transaction execution within the DAOFLC, contributing to the orchestration of the FL process. However, it is crucial to note that the FLTP is solely responsible for executing the approved proposals, making the overall orchestration process partially decentralized. The sequential process unfolds as illustrated in Fig. 4 as follows:

- **Step 1:** The FLTP generates a transaction called “propose” (or “proposecreateFLNFT” or “proposeUpdateGM”) and submits it to the MultiSigC with specific arguments.

This transaction encompasses various proposals such as “createFLNFT”, “Initiate_LMU”, “Cease_LMUs”, “setLMUVDRF”, or “UpdateGM”. A comprehensive explanation of these transactions is provided in Section IV-F. The MultiSigC first verifies the submitter of the transaction is MultiSigC’s owner. Subsequently, the MultiSigC performs a rigorous validation of the “propose” transaction, considering factors such as associated arguments, proposal nature, and current MultiSigC state. If the validation succeeds, a new “Proposal” is created with a unique identifier (`proposalID`) and set to the “Open” state. The proposal’s selector is configured using the corresponding function signature within the DAOFLC. The FLTP then communicates off-chain to persuade ODAOMs for approval. This process is encapsulated in the procedure `propose` described in Algorithm 2.

- **Step 2:** The ODAOMs first validate the proposal off-chain considering its properties, proposal nature, MultiSigC and DAOFLC states. If valid, an ODAO (ODAOM\(_i\)) initiates an “approve” transaction towards the MultiSigC, including the `proposalID` as an argument. The MultiSigC ensures that the transaction is indeed submitted by an ODAOM, the proposal is open, and `ODAOM\(_i\)` has not previously voted on the proposal. The MultiSigC performs comprehensive validation of the transaction considering relevant arguments, proposal nature, and MultiSigC state. If valid, an approval vote is recorded, and if the cumulative approvals exceed the quorum (60% of ODAOMTC supply), the proposal state is updated to “Executable.” This process is outlined in procedure `approve` in Algorithm 4.

- **Step 3:** After receiving necessary approvals, the FLTP executes the approved proposal by initiating an “execute” transaction towards the MultiSigC with the unique
Algorithm 4: Multi-Signature Contract MultiSigC

**Caller:** FLTP  
**Modifier:** onlyOwner()  
**Input:** [selector], [tokenURI], [GMCID], [t + 1]

1. procedure propose
2.  Require: Caller==MultiSigC.owner()
3.  Validate propose
4.  if propose is valid then
5.    proposal = Create new Proposal with proposalID
6.    Set proposal.state = Open, Set proposal.selector
7.  end if
8. end procedure

**Caller:** ODAOM,  
**Modifier:** onlyOwner()
1. procedure approve(uint proposalID)
2.  Ensure ODAOM, holds a ODAOMT
3.  proposal = Proposal[proposalID]
4.  if proposal.state == Open and ODAOM, ! proposal.approvals then
5.    Add ODAOM, to proposal.approvals
6.    numApprovals = proposal.approvals.length()
7.    curQuorum = 60% * n(ODAOMT)
8.  end if
9.  if numApprovals > curQuorum then
10.  Set proposal.state = Executable
11. end if
12. end procedure

**Caller:** FLTP  
**Modifier:** onlyOwner()
1. procedure execute(uint proposalID)
2.  Require: Caller==MultiSigC.owner()
3.  state = Proposal[proposalID].state
4.  if state == Executable then
5.    selector = Proposal[proposalID].selector
6.    argumentData = Proposal[proposalID].argumentData
7.    if Call DAOFLC.selector with argumentData then
8.    Set proposal.state = Executed
9.  end if
10. end if
11. end procedure

1. procedure closeProposal(uint proposalID)
2.  Require: Caller==MultiSigC.owner()
3.  state = Proposal[proposalID].state
4.  if state == Open or state == Executable then
5.    Set Proposal[proposalID].state = Closed
6.  end if
7. end procedure

**Algorithm 4** : Multi-Signature Contract MultiSigC

**ProposalID.** The MultiSigC verifies the proposal’s executability based on the proposal state (propose.state) and MultiSigC state. If conditions are met, the MultiSigC executes the proposal within the DAOFLC and updates its state accordingly. This process is captured in the procedure `execute` defined in Algorithm 4. Following the execution, the FLTP proposes the next “propose” transaction, in alignment with the FL process, to facilitate ongoing DAO-FL operations.

In cases where proposals lack sufficient approvals from ODAOs due to inaccuracies in the values of tokenURI and GMCID, the FLTP can create alternative proposals with accurate values. To close inaccurate proposals, the FLTP submits a “closeProposal” transaction using the respective proposalID as an argument. This process discards the inaccurate proposal and allows for subsequent accurate proposals. The steps for processing a “closeProposal” transaction are condensed in procedure `closeProposal` in Algorithm 4.

**F. Execution Workflow of DAO-FL framework**

In this subsection, we explore the execution workflow of the DAO-FL framework for a complete global iteration (GI) t, as depicted in Fig. 5. The following is a concise outline of the sequential flow:
Fig. 5. DAO-FL: Simplified execution workflow.

- Step 1: The FLNFTC is deployed for the FL ecosystem by the Regulator. The deployment transaction includes three arguments: “Federated Learning NFT” as the name, “FLNFT” as the symbol, and a base URI used in the TokenURI of FLNFTs. The ownership of FLNFTC is then transferred to the Regulator. The **FLNFTC_Constructor** procedure in Algorithm 5 summarizes this step.

- Step 2: For this particular FL task, FLTP deploys the ODAOC and specifies two candidate ODAOMs (ODAOM $i$) as arguments. The deployment transaction also includes a base_URI parameter, which serves as the base_URI in the TokenURI of ODAOMTs. The procedure **ODAOC_Constructor**, defined in Algorithm 7, is initiated for the deployment of ODAOC, and the ownership is transferred to FLTP. Subsequently, ODAOC deploys ODAOMTC with the name, symbol, and base_URI of ODAOMTs as arguments. The ownership of ODAOMTC is transferred to ODAOC. ODAOC then proceeds to mint ODAOMTs for FLTP and the two specified members, following the procedures **ODAOMTC_Constructor** and **mint** outlined in Algorithm 8. Once ODAOC is deployed, the ODAOMs gain the ability to perform membership enrollment and expulsion operations within the ODAOC, as defined in Section IV-B and Section IV-C, respectively.
Algorithm 5: FL-NFT Contract FLNFTC

 Owner: Regulator
 Deployer: Regulator

 Input: "Federated Learning NFT", "FLNFT", base_URI

 3: procedure FLNFTCConstructor(name, symbol, base_URI)
 4: Assign FLNFTC.owner ← Regulator.address
 5: Assign FLNFTC.name ← name
 6: Assign FLNFTC.symbol ← symbol
 7: Assign FLNFTC.base_URI ← base_URI
 8: end procedure

 Executor: DAOFLC

 1: procedure craftFLNFT(GMCID, tokenURI)
 2: FLNFTID = Mint FLNFT transferred to FLTP
 3: Assign FLNFT.tokenURI ← tokenURI
 4: Assign FLNFT.GMCID ← GMCID
 5: Assign FLNFT.OriginatingAddress ← DAOFLC.address
 6: end procedure

 Algorithm 6: Federated Learning Task Publisher

 1: procedure Generate_FLNFT
 2: Create GMt, \( t = 0 \)
 3: GMCID ← Store GMt on IPFS
 4: Create FLNFT_Metadata, for GMt
 5: tokenURI ← Store FLNFT_Metadata, on IPFS
 6: Call MultiSigC:proposecreateFLNFT(GMCID, tokenURI)
 7: end procedure

 1: procedure Initiate_LMUploads
 2: Call MultiSigC:propose (selector, \( t + 1 \)) \( \triangleright \) selector for proposal "Initiate_LMUs"
 3: end procedure

 1: procedure Halt_LMUploads
 2: Call MultiSigC:propose (selector, \( t + 1 \)) \( \triangleright \) selector for proposal "Cease_LMUs"
 3: end procedure

 1: procedure Configure_LMUVIDRF
 2: Call MultiSigC:propose (selector, \( t + 1 \)) \( \triangleright \) selector for proposal "setLMUVIDRF"
 3: end procedure

 1: procedure Aggregate_LMUs
 2: Create GMt+1 using [2]
 3: GMCID ← Store GMt+1 on IPFS
 4: Create FLNFT_Metadatat+1 for GMt+1
 5: tokenURI ← Store FLNFT_Metadatat+1 on IPFS
 6: Call MultiSigC:proposeUpdateGM(t+1, GMCID, tokenURI)
 7: end procedure

 Algorithm 7: Orchestration-DAO Contract ODAOC

 Owner: FLTP
 Deployer: FLTP

 1: procedure ODAOC_Constructor(address member1, address member2, base_URI)
 2: Set ODAOC.owner = FLTP.address
 3: Deploy ODAOMTC("Orchestrator-DAOMT","ODAOMT", base_URI)
 4: Call ODAOMTC.mint(FLTP)
 5: Call ODAOMTC.mint(member1)
 6: Call ODAOMTC.mint(member2)
 7: end procedure

 Algorithm 8: ODAO Membership Token Contract ODAOMTC

 Owner: ODAOC
 Deployer: ODAOC

 1: procedure ODAOMTC_Constructor
 2: Set ODAOMTC.owner = ODAOC.address
 3: Set ODAOMTC.name = name
 4: Set ODAOMTC.symbol = symbol
 5: Set ODAOMTC.base_URI = base_URI
 6: end procedure

 Caller: ODAOC
 Modifier: onlyOwner()

 1: procedure mint(address recipient)
 2: if candidate \( \notin \) ODAO then
 3: Mint ODAOMT for recipient
 4: end if
 5: end procedure

 Algorithm 9: Validation-DAO Contract VDAO

 Owner: FLTP
 Deployer: FLTP

 1: procedure VDAO_Constructor(address member1, address member2, base_URI)
 2: Set VDAO.owner = FLTP.address
 3: Deploy VDAOMTC("Validation-DAOOMT","VDAOMT", base_URI)
 4: Call VDAOMTC.mint(FLTP)
 5: Call VDAOMTC.mint(member1)
 6: Call VDAOMTC.mint(member2)
 7: end procedure

- Step 3: For this specific FL task, FLTP deploys the corresponding VDAO and adds two entities as potential VDAO members (VDAOMT). A base_URI is provided in the deployment transaction for the TokenURI of VDAOMTs. The VDAO’s deployment is initiated using the procedure VDAO_CONSTRUCTOR outlined in Algorithm 9, and ownership is transferred to FLTP. Subsequently, VDAO deploys VDAOMTC with the name and symbol of VDAOMTs and the base_URI as arguments. Ownership of VDAOMTC is transferred to VDAO, which then mints VDAOMTs for FLTP and the two specified members as outlined in the procedures VDAOOMTC_CONSTRUCTOR and mint of Algorithm 10. Upon deploying VDAO, VDAOOMTs acquire the capability to execute operations outlined in Section IV-B and Section IV-C within VDAO.

- Step 4: FLTP deploys DAOFLC by providing the addresses of FLNFTC, ODAO, and VDAO as arguments in the deployment transaction. Ownership of DAOFLC is transferred to FLTP. Subsequently, DAOFLC deploys FLTokenC with a specific name and symbol for FLTokens. FLTokenC transfers its ownership to
Algorithm 10: VDAO Membership Token Contract

\[
\text{VDAOOMTC} \quad \text{Owner: } \text{VDAOOC} \quad \text{Deployer: } \text{VDAOOC} \\
\text{Input: } \_\text{name, } \_\text{symbol, base\_URI} \\
1: \text{procedure } \text{VDAOOMTC}\_\text{Constructor} \\
2: \hspace{1em} \text{Set } \text{VDAOOMTC}\.\text{owner} = \text{ODAOC}\_\text{address} \\
3: \hspace{1em} \text{Set } \text{VDAOOMTC}\.\text{name} = \_\text{name} \\
4: \hspace{1em} \text{Set } \text{VDAOOMTC}\.\text{symbol} = \_\text{symbol} \\
5: \hspace{1em} \text{Set } \text{VDAOOMTC}\.\text{base\_URI} = \_\text{base\_URI} \\
6: \text{end procedure} \\
\text{Caller: } \text{VDAOOC} \quad \text{Modifier: } \text{onlyOwner()} \\
1: \text{procedure } \text{mint}(\text{address } \text{recipient}) \\
2: \hspace{1em} \text{if candidate } \notin \text{VDAO} \text{ then} \\
3: \hspace{2em} \text{Mint } \text{VDAOOMTC} \text{ for } \text{recipient} \\
4: \text{end if} \\
5: \text{end procedure}
\]

Algorithm 11: DAO FL Contract DAOFLC

\[
\text{DaoFLC} \quad \text{Owner: } \text{FLTP} \quad \text{Deployer: } \text{FLTP} \\
1: \text{procedure } \text{DAOFLC}\_\text{Construct}(\text{FLNFT}\_\text{address}, \text{ODAOC}\_\text{address}, \text{VDAOOC}\_\text{address}) \\
2: \hspace{1em} \text{Set } \text{DAOFLC}\.\text{owner} = \text{FLTP}\_\text{address} \\
3: \hspace{1em} \text{Deploy } \text{FLTokenC} (\text{"Federated Learning Token", } \text{"FLToken"}) \\
4: \text{end procedure} \\
\text{Caller: } \text{FLTP} \quad \text{Modifier: } \text{onlyOwner()} \\
1: \text{procedure } \text{setMultiSigCAddr}(\text{MultiSigC}\_\text{address}) \\
2: \hspace{1em} \text{Set } \text{DAOFLC}\.\text{MultiSigCAddr} = \text{MultiSigC}\_\text{address} \\
3: \hspace{1em} \text{end procedure} \\
\text{Caller: } \text{MultiSigC} \quad \text{Modifier: } \text{onlyMultiSigC()} \\
1: \text{procedure } \text{createFLNFT}(\text{tokenURI}, \text{GMCID}) \\
2: \hspace{1em} \text{FLNFTID } = \text{call } \text{FLNFT}\_\text{CraftFLNFT} (\text{tokenURI}, \text{GMCID}) \\
3: \hspace{1em} \text{Set } \text{DAOFLC}\.\text{FLNFTID} = \text{FLNFTID} \\
4: \hspace{1em} \text{Set } \text{DAOFLC}\.\text{GMCID} = \text{GMCID} \\
5: \hspace{1em} \text{end procedure} \\
\text{Caller: } \text{FLTrainer}_{i,t+1} \\
1: \text{procedure } \text{uploadLM}(\text{LMCID}, \text{LMURI}, t+1) \\
2: \hspace{1em} \text{if } \text{DAOFLC}\.\text{LMactive}F == \text{false} \text{ then} \\
3: \hspace{2em} \text{Set } \text{DAOFLC}\.\text{LMactive}F = \text{true} \\
4: \hspace{2em} \text{Emmit } \text{DAOFLC}\.\text{LMInitiated}(t+1) \\
5: \hspace{1em} \text{end if} \\
6: \text{end procedure} \\
\text{Input: } \text{LMCID}, \text{LMURI}, t+1, \text{FLTrainer}_{i,t+1}\_\text{address} \\
1: \text{procedure } \text{Record}\_\text{LM}(\text{LMCID}, \text{LMURI}, t+1, \text{FLTrainer}_{i,t+1}\_\text{address}) \\
2: \hspace{1em} \text{Set } \text{LM}\.\text{status} = \text{Submitted} \\
3: \hspace{1em} \text{Set } \text{LM}\.\text{LMCID} = \text{LMCID} \\
4: \hspace{1em} \text{Set } \text{LM}\.\text{LMURI} = \text{LMURI} \\
5: \hspace{1em} \text{Set } \text{LM}\.\text{approval}\_\text{votes} = \text{LM}\.\text{deny}\_\text{votes} = 0 \\
6: \hspace{1em} \text{Set } \text{LM}\.\text{voters} = \text{empty AddressSet} \\
7: \text{end procedure}
\]

DAOFLC. This process is summarized in the procedures \text{DAOFLC}\_\text{Constructor} of Algorithm 11 and \text{FLTokenC}\_\text{Constructor} of Algorithm 16.

- Step 5: The FLTP deploys the MultiSigC, and its ownership is transferred to the FLTP, as indicated in the procedure \text{MultiSigC}\_\text{Constructor} of Algorithm 13.

- Step 6: The FLTP submits the transaction “setMultiSigCAddr” to DAOFLC with the address of MultiSigC as an argument. The procedure \text{setMultiSigCAddr} in Algorithm 11 summarizes this step. After this transaction, MultiSigC will be able to execute transactions in DAOFLC.

- Step 7: In the \text{Generate\_FLNFT} procedure in Algorithm 6, the FLTP constructs the “preliminary GM parameters” for the FL task and stores it on IPFS, which yields a CID referred as GMCID. These parameters serve as GM for \(t = 0\). Additionally, FLTP uploads relevant files, including instructions for FL tasks, LMUs, reward criteria, and any tailored information, to IPFS. All of these details, including the addresses of associated contracts, are encompassed within a JSON-encoded meta-data identified as \text{FLNFT\_Metadata}. The \text{FLNFT\_Metadata} is uploaded to IPFS, resulting in a CID called \text{tokenURI}. Afterward, the FLTP initiates procedure \text{proposecreateFLNFT} in Algorithm 2 with arguments e.g. \text{tokenURI} and GMCID. This will initiate the multi-signature process as detailed in Section IV-E for proposal “createFLNFT”. During the execution of proposal “createFLNFT” as illustrated in procedure \text{createFLNFT} of Algorithm 11, the DAOFLC mints the FLNFT on FLNFTC for FLTP using procedure \text{craftFLNFT} in Algorithm 5. The corresponding properties including the \text{Orchestrator\_Address} of FLNFT are also set.

- Step 8: The FLTP triggers the procedure \text{Initiate\_LM\_uploads} in Algorithm 6 to commence the LMUs on the DAOFLC. FLTP initiates the \text{propose} procedure in Algorithm 2 with parameters like \text{selector} and \(t + 1\), where \text{selector} is derived from the Keccak-256 hash of the “Initiate\_LMUs” function signature in the DAOFLC. This starts the multi-signature process outlined in Section IV-E for the proposal “Initiate\_LMUs”. During its execution, the \text{Initiate\_LMUs} procedure in Algorithm 11 checks the status of the \text{DAOFLC}\_\text{LMactive}F flag. A true value indicates that LMUs are accepted, while a false signifies LMU closure. If \text{DAOFLC}\_\text{LMactive}F is false, the procedure updates it to true and emits the \text{LMUsInitiated}(t + 1) event, indicating the initiation of LMUs for GI \(t + 1\). FL-Trainers monitor this event to submit their LM updates.

- Step 9: \text{FLTrainer}_{i,t+1} concurrently initiate procedure \text{SEND\_LM} in Algorithm 14 to commence their local private dataset \(D_{i,t+1}\). \text{FLTrainer}_{i,t+1} retrieves the latest GM CID from \text{DAOFLC}\_\text{GMCID} and downloads the corresponding GM (\text{GM}_i) from IPFS. Utilizing their local private dataset \(D_{i,t+1}\), \text{FLTrainer}_{i,t+1} compute local model \(L_{M_{i,t+1}}\) as \([10, 15]\):

\[
\mathbf{w}_{i,t+1} \leftarrow \mathbf{w}_t - \eta_t g_t, \quad \forall i.
\]
Algorithm 12: DAOFLC - Continued

1: Caller: MultiSigC  Modifier: onlyMultiSigC
2: procedure SendLMU(LMU\_activeF, LMU\_stoppedF, LMUs\_ceasedF)
3: if LMU\_activeF == true then
4: Set LMUs\_activeF = false and LMUs\_stoppedF = true
5: Emit LMUs\_stoppedF LMUs\_ceasedF
6: end if
7: end procedure

Algorithm 13: Multi-Signature Contract MultiSigC
Owner: FLTP  Deployer: FLTP
Input: DAOFLC\_address, DAOAC\_address
1: procedure MultiSigC\_construct()
2: Set MultiSigC\_owner = FLTP\_address
3: end procedure

Algorithm 14: FL-Train FLTrainer\_i+1
1: procedure SendLMU
2: Get DAOFLC\_GMCID
3: Download GMCID \_IPFS using DAOFLC\_GMCID
4: Generate LMU\_t+1 using [\]
5: LMCI\_D = Store LMU\_t+1 on IPFS
6: Create LMURI for LMU\_t+1
7: LMURI = Store LMUR\_t+1 on IPFS
8: Call DAOFLC\_uploadLM(LMUCID\_1, LMURI, t+1)
9: end procedure

Algorithm [11] is instigated by FLTrainer\_i+1. DAOFLCAuthenticateLMU function validates the LMU\_t+1, potentially rejecting it if the LMUs limit is reached. If valid, LMU\_t+1 is appended to the LMUs for GI t + 1 and associated with the FLTrainer\_i+1 via procedure FLTP\_RecordLMU in Algorithm [11] LMU properties, such as approval and deny votes, are set to 0, and LMU\_t+1 status is marked as “Uploaded”.

- Step 11: The FLTP commences the HaltLMUploads procedure in Algorithm [5] to cease LMUs on the DAOFLC. This procedure instigates the propose procedure in Algorithm [4] with arguments like selector and t + 1, where selector represents the selector for the DAOFLC’s “Cease LMUs”. This triggers the multi-signature process as detailed in Section [IV-E] for the “Cease LMUs” proposal. The execution of this proposal activates the Cease LMUs procedure in Algorithm [12] If LMU\_activeF is true, it is changed to false, emitting the LMUs\_ceased(t+1) event. The LMUC flag is set to true, indicating the cessation of LMUs for GI t + 1, and FL-Trainers halt LM uploads.

- Step 12: After LMUs are ceased for t + 1, VDAOIs in VDAO concurrently initiate the procedure ReviewLMUploads (Algorithm [15]). In this procedure, each VDAOI downloads the LMU uploaders’ addresses using the function DAOFLC\_FetchLMU(t+1). For each FLTrainer\_i+1 in the fetched list, the DAOFLC downloads the corresponding LMU (LMU\_t+1) using DAOFLC\_FetchLMU(t+1, FLTrainer\_i+1). The VDAOIs checks LMU\_t+1 and casts an approval or denial vote by invoking procedure DAOFLC\_voteLMU with a boolean vote argument. True signifies approval, while false indicates disapproval for LMU\_t+1. The total approval and denial votes are counted, and the quorum is determined. If the total approval votes exceed the quorum, the procedure FLTP\_IssueLMToken is utilized to issue a FL-Token for FLTrainer\_i+1, and the LM status is set to “Rewarded”. However, if the total denial votes exceed the quorum, the LM status is set to “Denied”.

- Step 13: The FLTP initiates the procedure ConfigureLMU\_VDRC in Algorithm [11] to signify the verification, denial, and reward status of the LMUs. Using the selector of the “setLMUVDRC” function within the DAOFLC and t + 1 as arguments, the FLTP triggers the multi-signature process as outlined.
Algorithm 15 : VDAO member VDAOMi

1: procedure Review_LMUploads
2:   foreach FLTraineri,t+1 in DAOFLC.Fetch_LMUs(t + 1) do
3:     LMU,t+1 = Call DAOFLC.Fetch_LMU(t + 1,
4:        FLTraineri,t+1.address)
5:     Call DAOFLC.voteLMU(FLTraineri,t+1.address, t + 1, vote)
6: end procedure

Algorithm 16 : FL Token Contract FLTokenC

Owner: DAOFLC  Deployer: DAOFLC

1: procedure FLTokenC.Constructor(name, symbol)
2:   Set FLTokenC.owner = DAOFLC.address
3:   Set FLTokenC.name = name
4:   Set FLTokenC.symbol = symbol
5: end procedure

1: procedure issueFLToken(FLTraineri,t+1.address)
2:   Mint 1 * 10^18 FLToken for FLTraineri,t+1
3: end procedure

in Section [4] for the “setLMUVDRF” proposal by invoking procedure propose in Algorithm 4. As part of executing this proposal, the procedure setLMUVDRF in Algorithm 12 is activated, setting the flag LMUVDRF(t + 1) for GI t + 1 implying the verification, denial, and reward status of the respective LMUs.

• Step 14: The FLTP initiates the Aggregate_LMUs procedure in Algorithm 6. The approved and rewarded LMUs from previous steps are denoted as LMUt + 1. The FLTP computes GMt+1 using federated averaging (FedAvg) as

\[ w_{t+1} = \sum_{i \in LMU_{t+1}} \frac{n_i}{\eta} w_{i,t+1} \]  

where \( w_{i,t+1} \) is local parameter, \( w_{i,t+1} \) is global parameter, \( n_i = |D_i| \), and \( \eta = \| D_i \| \) and stores it on IPFS, yielding in CID GMCID. The updated meta-data, encoded in JSON format, denoted as FLNFTMeta data_{t+1}, is created and stored on IPFS, resulting in CID tokenURI. The FLTP then proposes the “UpdateGM” using the procedure proposeUpdateGM (propose) in Algorithm 4 with arguments such as \( t + 1, GMCID, \) and tokenURI. This triggers the multi-signature process outlined in Section [4] for the proposal. During this process, ODAOMs aggregate LMU_{t+1} following predefined guidelines and approve the proposal to certify its authenticity and accuracy. During the execution of the proposal, the UpdateGM procedure in Algorithm 12 is called. This procedure sets the GMCID and tokenURI of the FLNFT by invoking the FLNFTC.assignGMCID and FLNFTC.assignTokenURI procedures (Algorithm 5) respectively [15]. Only the registered OrchestratorAddress can execute these procedures. The FLNFTC.assignGMCID verifies the submitted GMCID using the FLNFTC.Verify_GMCID function, ensuring unique GMCIDs across all FL-NFTs. Similarly, the FLNFTC.assignTokenURI verifies the submitted tokenURI using the FLNFTC.Verify_TokenURI function, ensuring unique “tokenURIs” for all FL-NFTs. The DAOFLC emits the event DAOFLC.GMupdated, and the GIC[t + 1] is flagged by DAOFLC to indicate the completion of GI t + 1.

Step 1 of the above execution workflow is performed once by the Regulator to establish the FL marketplace ecosystem. For each FL task, Steps 2-7 are repeated for each FL task to prepare the FL decentralized orchestrating space using the DAO-FL framework. Steps 8-14 are repeated for each GI t + 1 within an FL task.

G. Commercializing GM and Transferring ownership

The GM is tokenized for efficient orchestration of FL process as well as to commercialize it via platforms such as OpenSea. This trading involves transferring the FL-NFT of GM to the buyer. However, in DAO-FL, the owner of FL-NFT i.e. FLTP also owns multiple contracts such as DAOFLC, MultiSigC, DAOAC, and VDAO. The process of transferring the FL-NFT to a new propietyang begins with the current owner, referred to as FLTP, initiating the FLNFT_Transfer procedure as outlined in Algorithm 17. This procedure involves the transfer of the FL-NFT to the designated Subsequent recipient. Afterward, the ownership of DAOFLC, MultiSigC, DAOAC, and VDAO is also transferred to the new owner.

V. IMPLEMENTATION, DEPLOYMENT, AND EVALUATION

In this section, we present the implementation, deployment, and evaluation aspects of the DAO-FL framework.

A. Implementation and Deployment

The smart contracts for the DAO-FL framework were developed using the Solidity programming language [23]. To visualize the inheritance hierarchy of these contracts, we utilized the Surya tool [24]. To enable membership in ODAO and VDAO, we required a token standard known as Non-Transferable-Token (NTT), such as EIP-4671 [25]. However, as NTT tokens were still in the early stages of development and might not meet our specific requirements, we created a custom smart contract called “DAOOMTC” to implement DAOOMTs.

The inheritance graph of DAOOMTC, illustrated in Fig. 6 demonstrates that DAOOMTC is inherited from customized OpenZeppelin [26] “Ownable” contract [27] and “ERC165” contract. Additionally, DAOOMTC implements the IERC721Metadata interface. Since DAOOMTs are NTT, certain functions of the IERC721 interface are not applicable
The smart contracts underwent compilation using the Hardhat [30]. Following this, the deployment of the smart contracts took place on the Sepolia testnet [31] utilizing JavaScript and Hardhat. To ensure transparency, the deployed smart contracts on the Sepolia network were verified using the ETHERSCAN_API_KEY [32]. The gas utilized, gas price, and transaction fee (in ethers) for deploying smart contracts are illustrated in Fig. 6. It should be noted that the gas used for ODAOMTC, VDAOMTC, and FLTokenC is encompassed within the gas used for ODAO, VDAO, and DAOFLC, respectively. For FLNFTC, the gas price was approximately 0.15 Gwei, which was comparatively high, possibly due to network congestion during its deployment. As a result, the elevated gas price led to a transaction fee of 0.00032 ETH. Consequently, the gas price and transaction fee for FLNFTC are not depicted in Fig. 6.

The Etherscan links of key entities (Regulator, FLTP, and FLTrainer1,1) and smart contracts deployed on the Sepolia network are presented in Table II. By examining these addresses on Etherscan Explorer, users can gain access to comprehensive information including event logs, internal and external transaction logs, and verified contract codes [15].

Given the broader focus of our paper on establishing a decentralized ecosystem for input and output verification of FL process through multi-signature wallets and DAOs, we utilized the MNIST dataset for training the local and global models. Consequently, we will omit specific details related to model configuration, accuracy information, and data allocation in this context. Due to space constraints, some repetitive transactions required to reach quorum have been omitted in some onward figures for brevity.

As the procedures for member enrollment and expulsion are the same for ODAO and VDAO, we present the implementation results for ODAO. Fig. 9 illustrates the transaction list for a “Join Proposal” (JP), including the “proposeJoin” transaction initiated by ODAO_MT and the “voteJoin” transactions by ODAOM. It also captures the relevant events emitted by ODAO, such as JPsupport, JPdenialVote, and JPapprovalVote. Additionally, Fig. 10 showcases the minting of ODAO_M upon reaching the quorum, accompanied by the events “JPproved” emitted by ODAO to indicate JP approval and the “Transfer” event indicating the transfer...
Table II: Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value on Sepolia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regulator.etherscan</td>
<td><a href="https://sepolia.etherscan.io/address/0x8fa37ecf3d89361e60e7e6adf55485ae62cd2b2">https://sepolia.etherscan.io/address/0x8fa37ecf3d89361e60e7e6adf55485ae62cd2b2</a></td>
</tr>
<tr>
<td>FLIP.etherscan</td>
<td><a href="https://sepolia.etherscan.io/address/0x33d3609e5a897ce62f0665c669a61c1f75f47f22">https://sepolia.etherscan.io/address/0x33d3609e5a897ce62f0665c669a61c1f75f47f22</a></td>
</tr>
<tr>
<td>FLNFTC.etherscan</td>
<td><a href="https://sepolia.etherscan.io/address/0x3a1d8ba4769f0d5f93e4e5f6d7a6d9b7f4f973c7">https://sepolia.etherscan.io/address/0x3a1d8ba4769f0d5f93e4e5f6d7a6d9b7f4f973c7</a></td>
</tr>
<tr>
<td>ODAOC.etherscan</td>
<td><a href="https://sepolia.etherscan.io/address/0x002f304c1f3480b0a6f8472f268f8e82e6f11">https://sepolia.etherscan.io/address/0x002f304c1f3480b0a6f8472f268f8e82e6f11</a></td>
</tr>
<tr>
<td>ODAOMTC.etherscan</td>
<td><a href="https://sepolia.etherscan.io/address/0x6df3e610ce7dcb72710e1351c4e5815f4e28108">https://sepolia.etherscan.io/address/0x6df3e610ce7dcb72710e1351c4e5815f4e28108</a></td>
</tr>
<tr>
<td>VDAOC.etherscan</td>
<td><a href="https://sepolia.etherscan.io/address/0x0d9e6904a06b6cc79f9d75f93749b3d754f760b0">https://sepolia.etherscan.io/address/0x0d9e6904a06b6cc79f9d75f93749b3d754f760b0</a></td>
</tr>
<tr>
<td>VDAOMTC.etherscan</td>
<td><a href="https://sepolia.etherscan.io/address/0x3016c9b116659f9d747e4d17d6c5a429c41297f">https://sepolia.etherscan.io/address/0x3016c9b116659f9d747e4d17d6c5a429c41297f</a></td>
</tr>
<tr>
<td>ODAOFL.etherscan</td>
<td><a href="https://sepolia.etherscan.io/address/0x1395a10a82f8543cf5f8e91f7a1d97673c13c52d">https://sepolia.etherscan.io/address/0x1395a10a82f8543cf5f8e91f7a1d97673c13c52d</a></td>
</tr>
<tr>
<td>MultiSigC.etherscan</td>
<td><a href="https://sepolia.etherscan.io/address/0x7001b7f257edf4d97077e0f090c28b0e8528629">https://sepolia.etherscan.io/address/0x7001b7f257edf4d97077e0f090c28b0e8528629</a></td>
</tr>
<tr>
<td>FLTrainer.etherscan</td>
<td><a href="https://sepolia.etherscan.io/address/0xff0e2447422da30927079d75dd985cf0cd21e1">https://sepolia.etherscan.io/address/0xff0e2447422da30927079d75dd985cf0cd21e1</a></td>
</tr>
</tbody>
</table>

Fig. 8. Gas Used, Gas Price, and transaction fee (in ETH) for the deployment of smart contracts.

Fig. 9. Transaction sequence (DAOC.proposeJoin and DAOC.voteJoin) and emitted events for a “Join Proposal” on ODAOC [https://sepolia.etherscan.io/address/0xf002f304cb1c34b40d359d59472f268f8e82e6f1], [Block 3815817-3815822].

Fig. 10. Minting of ODAOMT after reaching the quorum of approval votes for “Join Proposal” and corresponding events emitted on ODAOC [https://sepolia.etherscan.io/address/0x7001b7f257edf4d97077e0f090c28b0e8528629].

![Transaction details]

of ODAOMT to the candidate, emitted by ODAOMTC. Similarly, Fig. [1] presents the transaction list for a “Kick Proposal” (KP), which includes the “proposeKick” transaction initiated by ODAOM and the “voteKick” transactions by ODAOMs. The associated events emitted by ODAOC, such as KPSubmitted, KPDeniedVote, and KPApprovalVote, are also captured. Furthermore, Fig. [2] showcases the burning of ODAOMT upon reaching the quorum, along with the events “KPApproved” emitted by ODAOC to indicate KP approval and the “Transfer” event signifying the burning of ODAOMT.
Fig. 11. Transaction sequence (DAOC.proposeKick and DAOC.voteKick) and emitted events for a “Kick Proposal” on ODAOC (https://sepolia.etherscan.io/address/0xf002f304Cb1C34b40d593474f72f68FC882e61f), [Block 3815823-3815828].

Fig. 12. Burning of ODAOMT after reaching the quorum of approval votes for “Kick Proposal” and corresponding events emitted on ODAOC (https://sepolia.etherscan.io/tx/0x7de873fc9bdfb1fca45ad560430eff5ee4778e821fd1e8d981c12a6f1c099da3).

Fig. 13. Transaction sequence for the creation and execution of the “createFLNFT” proposal on MultiSigC, along with emitted events (https://sepolia.etherscan.io/address/0x7001b7f257EEDF4b970509916BD0ec0), Block [3829542-3829547].

Onwards in this section, we present the implementation of the DAO-FL framework, following the steps outlined in Section IV-F. Fig. 13 depicts the creation of a “createFLNFT” proposal by FLTP using the procedure FLTP.Generate_FLNFT through the transaction “proposecreateFLNFT” on MultiSigC. It also includes one of the “approve” transactions by ODAOMs and the subsequent execution of the “createFLNFT” proposal by FLTP upon reaching quorum. The corresponding events emitted by MultiSigC, such as createFLNFTpCreated, ProposalApprovedSubmitted, ProposalExecutable, and ProposalExecuted, are also shown. Fig. 14 demonstrates the minting of FLNFT following the execution of the “createFLNFT” proposal. The events emitted by FLNFTC, including OrchestratorAddressSet, GMCIDset, and TokenURIset, are displayed. Additionally, the event FLNFTcreated emitted by DAOFLC is depicted. Fig. 15 illustrates the creation and execution of the “Cease_LMUs” proposal by FLTP, following its approval by ODAOMs. The figure also includes the emitted events, such as ProposalCreated and ProposalExecuted by MultiSigC, and LMUsInitiated by DAOFLC. After listening to the LMUsInitiated event, FLTrainers_{i+1} uploads LMs through the “uploadLM” transaction on DAOFLC, as depicted in Fig. 16. The event “LMuploaded” emitted by DAOFLC during a transaction is also shown.

The illustration of the creation and execution of the “Cease_LMUs” proposal will be omitted. However, after its execution, VDAOMs engage in the crucial task of input verification for the FL process. This is achieved through the initiation of “voteLMU” transactions, as illustrated in Fig. 17. The events LMUvoted, LMURewarded, and LMUdenied are emitted by DAOFLC which signifies the validation process of LMUs. Furthermore, the successful validation results in the minting of FLTokens, as indicated by the “Transfer” event emitted by FLTokenC for a FLTrainer_{i+1}.

We will omit the illustration of the execution of “setLMUADRF” proposal. However, after the execution of proposal “setLMUADRF”, FLTP submits proposal “UpdateGM” to MultiSigC as shown in Fig 18 where event UpdateGMpCreated is emitted. The proposal goes through the approval process by ODAOMs as decentralized output verification of
the FL process and is finally executed. The events emitted are ProposalExecuted by MultiSigC, GUpated by DOAFLC, and GMCIDset and TokenURIset by FLNFTC which shows that FLNFT has been updated.

B. Evaluation on Threat Models

In the context of information flows, vulnerabilities can arise at the input or output stages. Input vulnerabilities involve discrepancies between submitted inputs and prescribed policies. For the FL process, this could manifest as submitting inaccurate or malicious local models, and potentially impacting the entity (FL server) responsible for input acceptance. Output vulnerabilities, on the other hand, pertain to non-compliance of the output produced with information flow policies or post-production tampering. In the FL process, this translates to scenarios like global aggregation attacks or global model tampering, jeopardizing the integrity of the corresponding global model.

Fig. 19 depicts accuracy trends subject to input, output, and input & output attacks on MNIST and Fashion-MNIST datasets ($E = 10$ local epochs, $N = 10$ FL-Trainees per global epoch). Fig. 19(a, c, d, f) underscore DAO-FL’s robustness against malicious local models, which were rejected by Validation-DAO through decentralized input verification,
Fig. 18. Creation and execution of proposal “UpdateGM” after decentralized output verification by ODAOMs (https://sepolia.etherscan.io/address/0x7001b7f257EEDF4b970577c630959099916BD0cc0) and events emitted, Block [3843770-3843775].

thereby preserving accuracy. DAO-FL closely matches attack-free FedAvg accuracy, particularly nearing convergence. The slight accuracy drop in DAO-FL (upon input attack) versus attack-free FedAvg results from the diversity of accurate local models in attack-free FedAvg, while in DAO-FL, global parameters are biased towards approved local models. In contrast, under-attack FedAvg, reliant on a single manipulable server, loses accuracy under input attacks. Fig. 19(b, c, e, f) show DAO-FL strictly maintaining accuracy under output attack. This resilience stems from the Orchestration-DAO’s vigilance through decentralized output verification by rejecting malicious “UpdateGM” proposals and ensuring alignment with established policies, the Orchestration-DAO enforces the FLTP for accurate “UpdateGM” proposals. Conversely, under-attack FedAvg, prone to tampering or aggregation attacks, experiences accuracy deterioration. These illustrations show that DAO-FL outperforms in countering input and output attacks.

Moreover, it’s worth noting that these attacks can severely disrupt the FL process, potentially impeding convergence towards high accuracy or even causing learning failures due to vanishing or exploding gradients as depicted at epoch=10 onward in Fig. 19(c, f). Hence, preventing these attacks is pivotal for the success of the FL process.

C. Qualitative Evaluation and Discussion

Our proposed framework provides a secure management solution for FL process. The involvement of multiple stakeholders, including regulators, FLTP, ODAO, and VDAO, facilitates decentralized governance and decision-making. This enables a more democratic and diverse approach to managing the FL process. DAO-FL framework utilizes smart contracts ODAOC and VDAOC to manage membership in ODAO and VDAO respectively. It leverages minting and burning of membership tokens for enrollment and expulsion procedures. These membership operations are themselves decentralized relying on voting mechanisms to execute “Join Proposals” and “Kick Proposals”.

Using the DAO-FL framework, FL can become more secure through decentralized input verification and decentralized output verification. Additionally, “DAO-FL” demonstrates the creation and execution of proposals, validation of LMUs, and input verification by VDAOs. DAO-FL incorporates input verification through the validation of LMUs by VDAOs. This process enhances the trustworthiness of the federated learning process by allowing participants to verify the quality and integrity of the submitted local models. The level of decentralization in ODAO is directly correlated with the total supply of ODAOMTC. As the total supply of ODAOMTC increases, the decentralization of FLP’s output verification also increases. Similarly, the decentralization in VDAO is directly tied to the total supply of VDAOMTC. An elevated total supply of VDAOMTC fosters increased decentralization in the input verification process of FLP. In scenarios prioritizing high decentralization, especially in prominent federated learning setups, the trade-off of increased time and high cumulative transaction fees to reach the quorum becomes acceptable as the ODAOMTC or VDAOMTC supply increases.

In DAO-FL, the Orchestration-DAO only approves proposals in a decentralized fashion, The actual execution of these proposals still remains under the responsibility of FLTP, resulting in a partially decentralized orchestration of the FL process. To attain full decentralization orchestration, a potential solution involves substituting FLTP with an additional DAO, referred to as the Executer-DAO. Coupled with an appropriate multi-signature contract, this arrangement can facilitate the decentralized execution of approved proposals, thereby achieving a fully decentralized orchestration paradigm.

The innovative principles and technologies embedded in DAO-FL offer a versatile framework that extends beyond its original context. Beyond federated learning, DAO Membership Tokens (DAOMTs) can be universally utilized as proof of membership in diverse DAOs. The proposed decentralized enrollment and expulsion schemes hold relevance across various DAO implementations. The versatility of smart contracts like MultiSigC and DAOFLC is evident, as with thoughtful adaptation of requirements and nomenclature of proposals to be executed, they can be used to enable partially decentralized orchestration for a wide spectrum of information flows. Additionally, the efficacy of the proposed quorum-based decentralized input verification and decentralized output verification...
mechanisms is not confined to the realm of federated learning alone. These mechanisms can be adapted to suit the specific needs of diverse information flows that necessitate decentralized decision-making, ensuring their applicability across a broad array of information flows.

VI. CONCLUSION

In this article, we proposed the DAO-FL framework, a Decentralized Autonomous Organizations based framework for achieving decentralized input and output verification in the federated learning process. We introduced the concept of DAO membership tokens, which are soul-bound, non-transferable, and non-fungible tokens that serve as key governance tools within a DAO. These tokens play a crucial role in member enrollment and expulsion, ensuring a fair and transparent decision-making process. The utilization of an ERC-721-powered Validation-DAO ensures decentralized input verification, while a multi-signature-contract empowered by an ERC-721-based Orchestration-DAO enables decentralized output verification. The comprehensive system design, algorithms, sequence diagrams, and smart contract code presented in this study demonstrate the feasibility and effectiveness of the DAO-FL framework. The DAO-FL framework offers a promising solution for addressing the challenges of centralized input and output verification in federated learning. By leveraging the power of DAOs and introducing decentralized governance mechanisms, DAO-FL promotes transparency, fairness, and security in the collaborative machine-learning process.

APPENDIX A
DEMONSTRATIVE METADATA FOR FL-NFT, ODAOMT, AND VDAOMT

• Explore the FL-NFT’s metadata at https://ipfs.io/ipfs/QmCtmSJZyXi9BQZlfK62zo5wzsQWW4ZpelF9cJ5USQfWE
• Explore the metadata of ODAOMT at https://ipfs.io/ipfs/QmNPqQqiC1dwADZ2FLwtUi2nGr5CdKyzzZNEaroc3ZUS7R
• Explore the metadata of VDAOMT at https://ipfs.io/ipfs/QmRkHTzc3jvFDWVq9DUt1nCNWUAAANy8TyMRMeQhPp3

APPENDIX B
DAOMTC AND DAOUML DIAGRAM

See the UML diagram at https://github.com/DAOFL/DAOFLcode/blob/main/UML/appendixB.pdf

APPENDIX C
ODAOMTC, ODAC, VDAOMT, VDAOC, FLTOKENC, DAOFLC, FLNFT, AND MULTISIG UML DIAGRAM

See the UML diagram at https://github.com/DAOFL/DAOFLcode/blob/main/UML/appendixC.pdf

REFERENCES
Zhu Han (S’01, M’04, SM’09, F’14) received the B.S. degree in electronic engineering from Tsinghua University, in 1997, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Maryland, College Park, in 1999 and 2003, respectively. From 2000 to 2002, he was an R&D Engineer of JDSU, Germantown, Maryland. From 2003 to 2006, he was a Research Associate at the University of Maryland. From 2006 to 2008, he was an assistant professor at Boise State University, Idaho. Currently, he is a John and Rebecca Moores Professor in the Electrical and Computer Engineering Department as well as in the Computer Science Department at the University of Houston, Texas. He is also a Chair professor in National Chiao Tung University, ROC. His research interests include wireless resource allocation and management, wireless communications and networking, game theory, big data analysis, security, and smart grid. Dr. Han received an NSF Career Award in 2010, the Fred W. Ellersick Prize of the IEEE Communication Society in 2011, the EURASIP Best Paper Award for the Journal on Advances in Signal Processing in 2015, IEEE Leonard G. Abraham Prize in the field of Communications Systems (best paper award in IEEE JSAC) in 2016, and several best paper awards in IEEE conferences. Dr. Han was an IEEE Communications Society Distinguished Lecturer from 2015-2018, and is AAAS fellow since 2019 and ACM distinguished Member since 2019. Dr. Han is 1% highly cited researcher since 2017 according to Web of Science.

Choong Seon Hong (S’95-M’97-SM’11) received the B.S. and M.S. degrees in electronic engineering from Kyung Hee University, Seoul, South Korea, in 1983 and 1985, respectively, and the Ph.D. degree from Keio University, Tokyo, Japan, in 1997. In 1988, he joined KT, Gyeonggi-do, South Korea, where he was involved in broadband networks as a member of the Technical Staff. Since 1993, he has been with Keio University. He was with the Telecommunications Network Laboratory, KT, as a Senior Member of Technical Staff and as the Director of the Networking Research Team until 1999. Since 1999, he has been a Professor with the Department of Computer Science and Engineering, Kyung Hee University. His research interests include future Internet, intelligent edge computing, network management, and network security. Dr. Hong is a member of the Association for Computing Machinery (ACM), the Institute of Electronics, Information and Communication Engineers (IEICE), the Information Processing Society of Japan (IPSJ), the Korean Institute of Information Scientists and Engineers (KIISE), the Korean Institute of Communications and Information Sciences (KICS), the Korean Information Processing Society (KIPS), and the Open Standards and ICT Association (OSIA). He has served as the General Chair, the TPC Chair/Member, or an Organizing Committee Member of international conferences, such as the Network Operations and Management Symposium (NOMS), International Symposium on Integrated Network Management (IM), Asia-Pacific Network Operations and Management Symposium (APNOMS), End-to-End Monitoring Techniques and Services (E2EMON), IEEE Consumer Communications and Networking Conference (CCNC), Assurance in Distributed Systems and Networks (ADSN), International Conference on Parallel Processing (ICPP), Data Integration and Mining (DIM), World Conference on Information Security Applications (WISA), Broadband Convergence Network (BeN), Telecommunication Information Networking Architecture (TINA), International Symposium on Applications and the Internet (SAINT), and International Conference on Information Networking (ICOIN). He was an Associate Editor of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and the IEEE JOURNAL OF COMMUNICATIONS AND NETWORKS. He currently serves as an Associate Editor for the International Journal of Network Management.