Anonymity on Byzantine-Resilient Decentralized Computing

Kehao Ma[1], Minghui Xu[1*], Yihao Guo[1], Lukai Cui[1], Shiping Ni[1], Shan Zhang[2], Weibing Wang[3], Haiyong Yang[3], Xiuzhen Cheng[1]
[1]Shandong University
[2]Beihang University
[3]Cloud Inspur Information Technology Co., Ltd.
[*]Corresponding author: Minghui Xu (mhxu@sdu.edu.cn).

***Abstract*—In recent years, decentralized computing has gained popularity in various domains such as decentralized learning, financial services and the Industrial Internet of Things. As identity privacy becomes increasingly important in the era of big data, safeguarding user identity privacy while ensuring the security of decentralized computing systems has become a critical challenge. To address this issue, we propose ADC (Anonymous Decentralized Computing) to achieve anonymity in decentralized computing. In ADC, the entire network of users can vote to trace and revoke malicious nodes. Furthermore, ADC possesses excellent Sybil-resistance and Byzantine fault tolerance, enhancing the security of the system and increasing user trust in the decentralized computing system. To decentralize the system, we propose a practical blockchain-based decentralized group signature scheme called Group Contract. We construct the entire decentralized system based on Group Contract, which does not require the participation of a trusted authority to guarantee the above functions. Finally, we conduct rigorous privacy and security analysis and performance evaluation to demonstrate the security and practicality of ADC for decentralized computing with only a minor additional time overhead.**

***Index Terms*—Anonymity, decentralized computing, Byzantine resilience, privacy-preserving smart contract.**

I. Introduction

With the continuous development of information technology, decentralized computing is gaining increasing attention as an innovative computing paradigm. It is built on the core principles of distribution, sharing, and autonomy, aiming to break free from the centralized control found in traditional computing models. However, nodes participating in decentralized computing may cause problems if node identity is revealed. For example, if a competitor knows that we are participating in decentralized computing, they might attempt to interfere with it, blocking decentralized computing from working properly, reducing the effect of computing and attempting to access our private data. Thus, protecting the identity privacy of nodes in decentralized computing is an issue that we must take into consideration.

To be specific, one of the initial solutions we considered is using virtual identities, such as public key in Bitcoin, to mask real identities. However, this approach is vulnerable to social engineering attacks [2], where attackers can extract clues and information about a user's real identity by analyzing the correlations between the user's virtual identity and the compute instance they have participated in (shown in Fig. 1). LearningChain [3] protects identity privacy by changing virtual identities each time a user participates in a new computing process, or even within different iterations of the same process. Nevertheless, the application of this anonymous scheme will result in the lack of user management, making it impossible to block malicious users from entering the system and trace their activities. Due to the provision of anonymity, it is natural to lose the ability to judge user identity. Therefore, we can see that, like LearningChain, most anonymous schemes face conflicts between their anonymity and the blocking and tracing of malicious users. However, for most decentralized computing systems, the absence of preventive and control measures against malicious users can lead to significant security issues. For instance, in some cases of decentralized computing, only a limited number of male users can be tolerated, as implemented by SPDL [4], which allows for a maximum of 33% malicious users. Under this premise, the system should have the ability to distinguish malicious users, such as implementing identity authentication for users. Additionally, it should be able to trace and revoke malicious users when they engage in wrongdoing. Simultaneously, to ensure that the number of malicious users does not exceed this fault tolerance limit, the system should resist Sybil attacks [5], preventing malicious users from creating a multitude of Byzantine nodes that surpass this limit and rendering the system ineffective. Some solutions make some compromises to address the above issues, such as introducing trusted authority. For example, DAFL [6] is an anonymous scheme for decentralized computing that achieves traceability, but requires trusted authority to trace the real identities of malicious users. Similarly, Biscotti [7] protects against Sybil attacks, but also needs trusted authority to facilitate and guide the computing process. While the approach of introducing trusted authority can solve these issues, in many cases, we prefer peer-to-peer users to complete decentralized computing without relying on a trusted authority. In summary, concerning the current anonymous works, what we observe is that they either require the support of a trusted authority or are unable to guarantee the traceability and blocking of malicious users.

For the challenges outlined above, it is evident that we need a reliable, anonymous, and traceable authentication scheme for decentralized scenarios. When considering traceability and

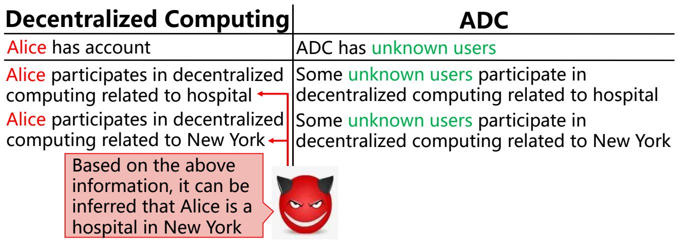
Decentralized Computing
ADC
Alice has account
ADC has unknown users
Alice participates in decentralized computing related to hospital
Some unknown users participate in decentralized computing related to hospital
Alice participates in decentralized computing related to New York
Some unknown users participate in decentralized computing related to New York
Based on the above information, it can be inferred that Alice is a hospital in New York

Fig. 1: Comparison Between Decentralized Computing and ADC

anonymity, group signature technology [22] naturally comes to mind. Nevertheless, group signatures necessitate a group manager, which does not align with the requirements of decentralization. DGSS [26] and DDGSS [27] are decentralized group signature schemes that distribute the permissions of the group manager to other nodes, allowing them to collectively perform the functions of the group manager. However, both schemes suffer from a Single Point of Failure (SPOF), meaning that if one of the nodes fails, the corresponding functions cannot be achieved. This is unacceptable, especially in systems with a large number of nodes like blockchain, where it is not guaranteed that every node will be error-free. Cao *et al.* [28] proposed using smart contracts to replace the role of the group manager. However, this scheme makes the group manager's private key public, enabling any node to trace back or revoke other nodes. To the best of our knowledge, there are no other fully decentralized group signature schemes besides the aforementioned three. Nevertheless, it is evident that these three schemes are not applicable to our desired scenario.

In this paper, we integrate private smart contracts [12] with group signatures [10] to develop a practical decentralized group signature scheme for blockchain, named Group Contract. This scheme offers anonymous identity verification while ensuring traceability and revocability without the need for a trusted authority. Building upon Group Contract, we introduce ADC, an anonymous decentralized computing system. ADC safeguards user identity privacy and enables the tracking and blocking of malicious users, eliminating the necessity for a trusted authority by employing blockchain as group managers. Additionally, ADC achieves Byzantine fault tolerance and Sybil-resistance through a self-designed group membership validation algorithm. This algorithm adopts the Practical Byzantine Fault Tolerant (PBFT) protocol [13] as the backbone, allowing nodes to reach consensus, establish decentralized trust, and maintain comprehensive, immutable, and traceable records of the entire framework.

The contributions of our work could be summarized as follows:

1) We present ADC, an anonymous decentralized computing system that incorporates anonymity and traceability mechanisms. In contrast to centralized approaches, ADC is not reliant on a trusted third party. The system ensures strong Byzantine fault tolerance and Sybil-resistance while preserving anonymity.
2) We propose Group Contract, a decentralized group signature scheme for ADC, which fully combines private smart contract and group signature. This scheme replaces the trusted authority required in the group signature with the entire blockchain, and utilizes private smart contract to prevent from privacy leakage.
3) We conduct a rigorous analysis of the security of ADC and design a series of experiments to demonstrate the security and practicality of ADC.

II. Related Work

A. Decentralized Computing

Decentralized computing refers to a model of computing where the processing power, data storage, and decision-making capabilities are distributed across a network of nodes, rather than being concentrated in a single server.

CoopEdge [30] is a novel blockchain-based decentralized platform used to drive and support cooperative edge computing. Warnat-Herresthal *et al.* [17] proposed swarm learning, where each node can participate in the computing process managed by Ethereum and smart contract. Biscotti [7] uses a commitment scheme to protect data privacy and employs a novel federated proof-based blockchain to improve the security in decentralized computing. SPDL [4] is a decentralized computing system that simultaneously ensures strong security using blockchain, Byzantine Fault Tolerant consensus, and BFT GAR, and preserves data privacy utilizing local gradient computation and differential privacy. TBAC [8] proposes a Tokoin-Based Access Control model utilizing blockchain and Trusted Execution Environment (TEE) technologies to combat the overprivilege attack phenomenon in IoT, offering fine-grained access control and strong auditability. TEMS [9] extends blockchain trust from on-chain to off-chain environments, exemplified by a trustworthy vaccine tracing scheme integrating a TEE-enabled monitoring system and a consistency protocol for real-time, fault-tolerant data transmission.

B. Anonymity in Decentralized Computing

Anonymity plays a crucial role in decentralized computing by safeguarding privacy and enhancing user security. There are some anonymous schemes that are suitable for decentralized computing scenario. DAFL [6] presents an anonymous authentication scheme in a decentralized scenario, achieved by combining directed acyclic graph blockchain and an accumulator. Nevertheless, it overlooks defense against Sybil attacks and depends on a trusted authority for tracing malicious nodes. LearningChain [3] safeguards identity privacy through the continual alteration of virtual identities. However, this method lacks traceability and security. Biscotti [7] employs a novel federated proof-based blockchain to improve the security in decentralized computing. It achieves Byzantine fault tolerance and Sybil-resistance, but lacks traceability and requires a trusted authority.

To address above challenges, this paper proposes an anonymous decentralized computing system, called ADC. ADC provides anonymity for decentralized computing while enabling the tracing and revocation of malicious nodes without relying on trusted authority. Additionally, we also achieve Byzantine fault tolerance and Sybil-resistance, which greatly enhances the security and credibility of ADC.

III. Models and Design Goals

A. System Model

In ADC, there are two types of nodes: users and workers. Users are participants in the entire decentralized computing system, while workers are enablers of this system, assisting in the operation of system functions.

- **Users** collectively execute the group initialization, and subsequently gain entry to the decentralized computing system upon successful registration and acquisition of their respective group member private key (gsk). In the context of a decentralized computing task, each user generates a temporary virtual identity. For instance, user X_i owns the identity (i, A_i), which cannot be linked to their signature unless it is traced.
- **Workers** (as special miners) are responsible for executing private smart contracts in Trusted Execution Environment (TEE) as required by the protocol of private smart contract. In this paper, we take ShadowEth [11] as an example of private smart contract. The privacy of contracts, including code, data, and private keys, is protected by TEE. Workers form a distributed storage network TEE-DS (implemented in ShadowEth [11]) to store private smart contract efficiently. Due to the nature of private smart contract, workers do not have access to the user's private data and code.

B. Threat Model

Regarding system security threats, we would introduce several attack methods that can pose a threat to the security of our system:

- Forgery of Identity: Maliciou user attempts to interfere with the normal operation of the system by forging the identity of users.
- Byzantine Attacks: Byzantine user does not adhere to the designed protocol and engage in arbitrary behavior, attempting to disrupt the normal operation of the system.
- Sybil Attacks: Malicious user tries to create multiple fake identities to gain control or influence over the operation of system.

C. Design Goals

In the context of ADC, it is imperative to safeguard the privacy of node identities while simultaneously enabling the tracing and revocation of malicious nodes to uphold system credibility. Additionally, the system must satisfy certain security prerequisites, such as unforgeability, Byzantine fault tolerance, and Sybil-resistance, to ensure stable operation. Furthermore, it is essential to maintain decentralization and avoid the introduction of trusted authorities. The design goals of Anonymous Decentralized Computing (ADC) are enumerated below for reference:

TABLE I: Summary of Important Notations

Sign	Description
gpk	the group public key
gmsk	the group manager private key
gsk	the group member private key
σ_G	the group signature
(i, A_i)	the group member identity information
GR	the label of group
DL	the label of computing instance
R	a temporary identification of user
(pk_c, sk_c)	signature key pair of Group Contract
P	a temporary identity prove of R
$B_u^{(t)}$,$BC_u^{(t)}$	t-th block and blockchain in DL_u

- Anonymity: User identities may be traced via a Trace function, which is leveraged for identifying malicious users. However, barring the Trace function, the user's authentic identity remains undisclosed.
- Unforgeability: An attacker cannot forge a legitimate user identity to pass verification.
- Byzantine Fault Tolerance: The system can tolerate up to $f < \frac{N}{3}$ Byzantine nodes without affecting its operation.
- Sybil-resistance: The system is impervious to the creation of multiple fake identities by an attacker and precludes their unjust acquisition of control or influence over the network.
- Decentralization: The system maintains a decentralized architecture without relying on any trusted authority.

D. Notations

A summary of important notations is provided in TABLE I. We use subscripts i, v, f and $\mathcal{V}$ to denote the index of users. For example, in the group member identity information (i, A_i), where i is the index of user X_i, and A_i is an identity parameter corresponding to user X_i during the trace phase in the group signature protocol. Subscript k is used to represent the index for group, for example, a group with index k is represented as GR_k. Subscript u is used to represent the index for decentralized computing, for example, decentralized computing with index u is represented as DL_u. The remaining subscripts represent special meanings; for instance, c in pk_c represents the public key of the Group Contract, distinguishing it from a regular public key pk. We use the notation $\langle \mathsf{mes} \rangle_{\sigma_G}$ to represent the information mes and the group signature σ_G on mes. For other notions, we will indicate their meaning at the point of their first appearance in this paper.

IV. Construction of ADC

A. Group Contract

The proposed solution, named "Group Contract", is a decentralized group signature scheme to provide support for group formation and group signature functionality in decentralized computing. This is achieved through the use of private smart contract to assist with group signature. In Group Contract,

```
public InitializeGroup
  Input: $1^\lambda$, $\mathsf{PubKey}_{DS}$
  Output: gpk, $pk_c$, Enc_gmsk and Enc_$sk_c$
  1. $\mathsf{gpk}, \mathsf{gmsk} \leftarrow \mathsf{GroupSig.Setup}(1^\lambda)$
  2. generate contract signature key pair $(pk_c, sk_c)$
  3. $\mathsf{Enc\_gmsk} \leftarrow \mathsf{Encrypt}(\mathsf{PubKey}_{DS}, \mathsf{gmsk})$
  4. $\mathsf{Enc\_}sk_c \leftarrow \mathsf{Encrypt}(\mathsf{PubKey}_{DS}, sk_c)$

public RegisterUser
  Input: $\mathsf{cred}_i$ and $\mathsf{PubKey}_{\mathsf{User}_i}$
  Output:
    • Enc_gsk_i and $(i, A_i)$ if verification of cred passes
    • $\perp$ if verification of cred fail
  1. if the verification of cred fail then
       output $\perp$
  2. else
    1) $\mathsf{gsk}_i, (i, A_i) \leftarrow \mathsf{GroupSig.Join}(\mathsf{gmsk})$
       // gmsk is in the form of constant
    2) $\mathsf{Enc\_gsk_i} \leftarrow \mathsf{Encrypt}(\mathsf{PubKey}_{\mathsf{User}}, \mathsf{gsk}_i)$
    3) output Enc_gsk_i and $(i, A_i)$

internal CountVote
  Input: $\langle \mathsf{vote}_1 \rangle_{\sigma_G}, \langle \mathsf{vote}_2 \rangle_{\sigma_G}, \cdots, \langle \mathsf{vote}_n \rangle_{\sigma_G}$
  Output: count
  1. for $i$ from 1 to $n$ do
    1) $A_i \leftarrow \mathsf{GroupSig.Open}(\mathsf{gmsk}, \sigma_G)$
       // gmsk is in the form of constant
  2. $\mathsf{count} \leftarrow 0$
  3. for non-duplicate $A_i$ do
       $\mathsf{count} \leftarrow \mathsf{count} + 1$
```

```
public Trace
  Input:
    • $\langle \mathsf{vote}_1 \rangle_{\sigma_G}, \langle \mathsf{vote}_2 \rangle_{\sigma_G}, \cdots, \langle \mathsf{vote}_n \rangle_{\sigma_G}$
    • $\sigma'_G$
  Output:
    • $A_i$ if the number of valid votes exceeds threshold $h$
    • $\perp$ if the number of valid votes less than $h$
  1. $\mathsf{count} \leftarrow \mathsf{Vote\ Count}(\langle \mathsf{vote}_1 \rangle_{\sigma_G}, \langle \mathsf{vote}_2 \rangle_{\sigma_G}, \cdots, \langle \mathsf{vote}_n \rangle_{\sigma_G})$
  2. if $\mathsf{count} > h$ then
    1) $A_i \leftarrow \mathsf{GroupSig.Open}(\mathsf{gmsk}, \sigma'_G)$
       // gmsk is in the form of constant
    2) output $A_i$
  3. else
       output $\perp$

public Revoke
  Input:
    • $\langle \mathsf{vote}_1 \rangle_{\sigma_G}, \langle \mathsf{vote}_2 \rangle_{\sigma_G}, \cdots, \langle \mathsf{vote}_n \rangle_{\sigma_G}$
    • $i$
  Output:
    • Acc if the number of valid votes exceeds threshold $h$
    • $\perp$ if the number of valid votes less than $h$
  1. $\mathsf{count} \leftarrow \mathsf{Vote\ Count}(\langle \mathsf{vote}_1 \rangle_{\sigma_G}, \langle \mathsf{vote}_2 \rangle_{\sigma_G}, \cdots, \langle \mathsf{vote}_n \rangle_{\sigma_G})$
  2. if $\mathsf{count} > h$ then
    1) $\mathsf{Acc} \leftarrow \mathsf{GroupSig.Revoke}(\mathsf{gmsk}, i, \mathsf{Acc})$
       // gmsk is in the form of constant
    2) output Acc
  3. else
       output $\perp$
```

Fig. 2: Group Contract. Note that, the symbols σ_G and σ'_G both represent group signatures. To facilitate the description of the protocol, we use σ_G to denote group signatures of honest users and σ'_G to represent group signatures of malicious users.

we use private smart contracts to replace the group manager role and leverage its features to protect group manager private key and group member private key. Through Group Contract, users can sign and verify messages using the original group signature algorithm, while tracing and revoking malicious users by invoking private smart contract.

Specifically, we represent the group signature functions used in our work using a tuple of polynomial-time algorithms denoted by GroupSig $\overset{\text{def}}{=}$ (Setup (for initialization), Join (for group member registration), Sign (for outputting a signature), Verify (for verifying a signature), Open (for tracing which group member the signature comes from), and Revoke (for revoking a group member)) [10]. To further ensure privacy within the Group Contract, we employ contracts with the help of private smart contract [11]. This allows us to protects both the private key of group managers and group members. Additionally, input and output parameters are encrypted in TEE. Some parameters, such as the group public key, need to be publicly accessible in the realm of group signatures.

The Group Contract consists of five functions (detailed in Fig. 2): **InitializeGroup**, **RegisterUser**, **CountVote**, **Trace** and **Revoke**.

Among these five functions, CountVote is an internal function, which can only be invoked by Group Contract internally, and the other four functions are public, which can be invoked by external users. The code of these functions is protected by TEE-DS [11] in encryption form, and workers need to interact with it securely to get the encrypted code and decrypt it in TEE. It is important to note that the remaining four functions will only be generated after InitializeGroup is executed. Some parameters are stored in those functions in the form of constant. Next, we would describe each function in detail.

- **InitializeGroup**: The InitializeGroup function initializes a group and generates key parameters used for the remaining functions. It takes 1^λ and PubKey_{DS} as input, where PubKey_{DS} is the public key used by TEE-DS for encryption and 1^λ is the security parameter. First, InitializeGroup calls GroupSig.Setup, which takes 1^λ as inputs to generate a group public key gpk and a group manager private key gmsk. Then, it generates signature key (pk_c, sk_c) for the Group Contract, and then encrypts gmsk and sk_c using PubKey_{DS}. Finally, InitializeGroup outputs gpk, pk_c, Enc_gmsk and Enc_sk_c. After InitializeGroup is executed, Enc_gmsk and Enc_sk_c will be decrypted in TEE-DS. Subsequently, these two parameters will be used to generate the remaining four functions, in which these two parameters are in constant form. Because the code is protected by TEE-DS, these two parameters will not be leaked.

- **RegisterUser**: The RegisterUser function is used for user registration. It takes cred and $\mathsf{PubKey}_{\mathsf{User}}$, where cred is the credential of a user and $\mathsf{PubKey}_{\mathsf{User}}$ is the user's public key. For a user X_i, this function first validates the user's cred_i, and then uses GroupSig.Join algorithm to generate the group member private key gsk_i and the group member identity (i, A_i) for X_i. Finally, the function outputs Enc_gsk$_\mathsf{i}$, which is encrypted using $\mathsf{PubKey}_{\mathsf{User}_i}$ so that no one can access it except for the owner X_i. Additionally, (i, A_i) is stored locally by a group manager but in this scheme, the centralized group manager is removed and it is stored on the blockchain for future traceability purposes. (i, A_i) is merely a notation and is not directly linked to the signature, thus it does not disclose any private identity information about user X_i.
- **CountVote**: The CountVote function is designed as an internal mechanism restricted to use solely within the Trace and Revoke functions. Its primary function is to scrutinize and enumerate non-duplicate votes. To accomplish this, the function leverages the GroupSig.Open algorithm to authenticate the identities of signatories. The anonymous nature of group signature necessitates this approach to avoid double-counting from a single entity. Upon successful verification, the function tallies the votes and returns the count as count.
- **Trace**: The primary purpose of this function is to trace malicious users. Firstly, the Trace function calls upon the CountVote function to enumerate the votes of other users. If the vote count count surpasses the predefined threshold h, then it utilizes GroupSig.Open to produce the identity parameter A_i of the malicious user X_i' based on the group manager's private key gmsk and the group signature σ_G' (generated by the malevolent user). The blockchain enables all users to obtain the label i that corresponds to A_i, which signifies the successful association of signatures with user identity. In the subsequent Revoke function, revocation of the malicious user can be accomplished by providing the label i.
- **Revoke**: The purpose of this procedure is to invalidate a user with the identifier i who is acting maliciously. Following the invocation of the CountVote procedure to tally the votes of the remaining users, the Revoke procedure updates the Acc parameter of the dynamic accumulator with the aid of the GroupSig.Revoke algorithm if the specified threshold h is surpassed, thereby signifying that the user X_i is now invalidated.

Based on the above design, Group Contract can provide basic technical support for our scheme ADC and contributes to properties including anonymity, traceability, and decentralization.

B. The Protocol of ADC

Based on the technical support of Group Contract, ADC can be divided into four phases: Initialization, User Registration, Decentralized Computing, and Trace&Revoke, as illustrated in Fig. 3.

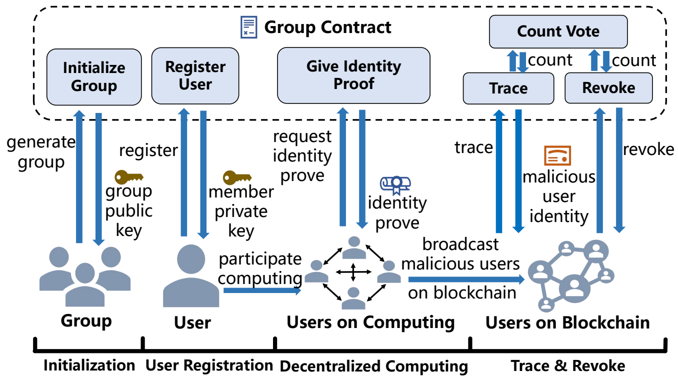

Fig. 3: Overview of ADC

1) Initialization: We set up an interactive scene, where users $X_1, X_2, \cdots, X_t$ construct a group GR_k. They invoke the InitializeGroup function of Group Contract with a security parameter 1^λ and the TEE-DS's public encryption key PubKey_{DS} as inputs. The function outputs multiple parameters for the next phases, such as the group public key gpk, the contract signature public key pk_c, the encrypted group manager private key Enc_gmsk and encrypted contract signature private key Enc_sk.

2) User Registration: During the user registration, users obtain their member private key gsk and join ADC system. User Registration consists of the following steps:

Step 1: User X_i invokes the RegisterUser function of Group Contract with cred_i and $\mathsf{PubKey}_{\mathsf{User}_i}$ as input, in which cred_i is credential of user and $\mathsf{PubKey}_{\mathsf{User}_i}$ is the user's public encryption key. The RegisterUser function outputs the encrypted group member private key Enc_gsk$_\mathsf{i}$, which can be decrypted by the corresponding user X_i. Additionally, the RegisterUser function also outputs user's group member identity information (i, A_i), which will be recorded on the blockchain for subsequent tracing. Since (i, A_i) is purely a notation, and as long as it is not associated with the signature, it does not reveal any private information.

Step 2: User X_i decrypts Enc_gsk$_\mathsf{i}$, obtaining its group signature private key gsk_i. With this signature private key, user X_i can participate in the decentralized computing process.

3) Decentralized Computing: The decentralized computing in ADC is divided into two parts: group membership validation and computing. Group membership validation is used to confirm the qualifications of users and in the computing part, users validated by group membership validation undergo computing.

a) Group Membership Validation: Here, we add a public function to the original Group Contract called GiveIdentityProof, which is presented in Fig. 4. This function is used to verify a user's temporary identification R and issue an identity proof P. This function first uses GroupSig.Verify to verify the correctness of signature based on the group public key gpk and

```
public GiveIdentityProof
  Input:
    • $R$
    • $DL$
  Output:
    • $P$ if verification of $\sigma_G$ passes
    • $\perp$ if verification of $\sigma_G$ fail
  1. get $\sigma_G$ from $R$
  2. if $\mathsf{GroupSig.Verify}(\mathsf{gpk}, \sigma_G) == 0$ then
       output $\perp$
  3. $A_i \leftarrow \mathsf{GroupSig.Open}(\mathsf{gmsk}, \sigma_G)$
  // gmsk is in the form of constant
  4. $P \leftarrow \langle DL, R, \mathsf{Hash}(DL, A_i)\rangle_{\sigma_{\mathsf{contract}}}$
  // $\sigma_{\mathsf{contract}}$ is signature using contract signature private key
  $sk_c$ that is in the form of constant
  5. output $P$
```

Fig. 4: GiveIdentityProof

group signature σ_G. And then the GroupSig.Open algorithm is executed to obtain the member identity parameter A_i of the user. Then, the hash value of the label of decentralized computing DL and the member identity parameter A_i is computed. This method ensures that different decentralized computing would yield different hash values but, within a specific decentralized computing, signatures from the same users would result in the same hash value. ADC can effectively resist Sybil attacks through this method.

During the group membership validation phase, our main objective is to authenticate the users participating in the computing, exclude ineligible users, and prevent Sybil attacks. Ultimately, the users reach a consensus and form a computing users list U. We adopt the Practical Byzantine Fault Tolerant (PBFT) [13] protocol as the backbone for achieving consensus, which operates independently from the digital currency system and demonstrates high efficiency. The group membership validation phase comprises four phases: PRE-PREPARE, PREPARE, COMMIT, and ACCOMPLISH. We show it in Algorithm 1.

In the PRE-PREPARE phase, each user participating in decentralized computing DL_u generates a one-time and non-repeating public and private key pair (pk_i, sk_i) using the Elliptic Curve Digital Signature Algorithm (ECDSA) [25]. We denote the signature using ECDSA as σ, and the group signature as σ_G. Each user X_i generates a temporary identification $R_i = \langle ID_i, GR_k, pk_i\rangle_{\sigma,\sigma_G}$, where ID_i is a randomly generated number and GR_k represents the group label to which the group signature obtained in the previous User Registration phase belongs. Subsequently, X_i invokes the GiveIdentityProof function of the Group Contract using R_i and the label of decentralized computing DL_u and receives the output P_i, which serves as a temporary identity proof. Here it can be understood that X_i requests from the blockchain a temporary identity that would be valid for this particular decentralized computing process. Next, X_i broadcasts P_i to its peers and receives P_v from them.

Algorithm 1: Group Membership Validation

```
1  PRE-PREPARE
2  $R_i \leftarrow \langle ID_i, GR_k, \mathsf{gpk}_k, pk_i\rangle_{\sigma,\sigma_G}$
3  $P_i \leftarrow \mathsf{GiveIdentityProof}(R_i, DL_u)$
4  broadcast $P_i$ and receive $P_v$ from peers
5  PREPARE
6  while receive $P_v$ do
7      if $\mathsf{Verify}(P_v) == 1$ then
8          get $\mathsf{Hash}(DL_u, A_v)$ from $P_v$
9          $\mathsf{M\_SET} \leftarrow \mathsf{Hash}(DL_u, A_v)$
10 for non-repeating $\mathsf{Hash}(DL_u, A_v) \in \mathsf{M\_SET}$ do
11     get $R_v$ from corresponding $P_v$
12     $\mathsf{SET}_R \leftarrow R_v$
13 for $R_v \in \mathsf{SET}_R$ do
14     broadcast $\langle \mathsf{PREPARE}, R_i, R_v\rangle_\sigma$
15 COMMIT
16 while receive $2f+1$ of $\langle \mathsf{PREPARE}, R_{i'}, R_v\rangle_\sigma$ do
17     broadcast $\langle \mathsf{COMMIT}, R_i, R_v\rangle_\sigma$
18 ACCOMPLISH
19 while receive $2f+1$ of $\langle \mathsf{COMMIT}, R_{i'}, R_v\rangle_\sigma$ do
20     $U \leftarrow U \cup R_v$
21 $B_u^{(0)} \leftarrow (DL_u, U)$
22 append $(BC_u^{(0)}, B_u^{(0)})$
```

In the PREPARE phase, we embed the determination of Sybil attacks to ensure that the computing user list does not contain users pretending to be multiple users by sending multiple identity information R. We will discuss in Section V why this phase can resist the Sybil attacks. First, user X_i verifies P_v (denoted as $\mathsf{Verify}(P_v)$) received from user X_v. If the embedded information is duplicated or the signature is invalid, P_v is discarded. And the $\mathsf{Hash}(DL_u, A_v)$ in the remaining verified P_v is deposited into M_SET, which is used to judge and defend against Sybil Attacks by checking for duplicate content. Then, X_i stores R corresponding to the non-repeating $\mathsf{Hash}(DL_u, A_v)$ in SET_R. For each $R_v \in \mathsf{SET}_R$, user X_i signs it and broadcasts $\langle \mathsf{PREPARE}, R_i, R_v\rangle_\sigma$ to the other users. Here, the size of SET_R is denoted as n. And the maximum allowable number of Byzantine users $f = \frac{n}{3}$.

It's important to emphasize that we only utilize group signatures in the PRE-PREPARE and PREPARE phases. Group signatures are not required in the subsequent processes since the group signature is already bound to the ECDSA signature. This optimization significantly reduces the signature overhead of the system. Moreover, the temporary nature of the ECDSA keys in this system ensures the preservation of identity privacy as they will be changed in the next decentralized computing process.

In the COMMIT phase, if user X_i receives $\langle \mathsf{PREPARE}, R_j, R_v\rangle_\sigma$ from at least $2f + 1$ different users for any R_v, it broadcasts $\langle \mathsf{COMMIT}, R_i, R_v\rangle_\sigma$.

In the ACCOMPLISH phase, for any R_v, if user X_i receives

$\langle \mathsf{COMMIT}, R_j, R_v\rangle_\sigma$ from at least $2f+1$ different users, it adds R_v to its computing users list U. Finally, user X_i sets its broadcast list to the users in U. A new block $B_u^{(0)}$ is generated, which records U and DL_u, and is appended to the blockchain $BC_u^{(0)}$.

b) Computing: We will not delve into the detailed algorithm flow of computing, as our focus is on verifying the messages sent by each user during the computing process, which we describe in detail below. In this context, we represent the message sent by user X_i in round t of computing as (t, mes_i).

During computing, each user maintains an array $C[|U|]$ to keep track of the number of times it receives information from a particular user, thereby preventing replay attacks. Each message to be transmitted is packaged as $\langle ID_i, (t, \mathsf{mes}_i)\rangle_\sigma$, where σ denotes the ECDSA signature. Group signature is not required here because it is already bound to the ECDSA signature during group membership validation. Additionally, since the ECDSA signature is temporary and disposable and will be changed in the next decentralized computing process, it does not disclose any identity privacy. We solely utilize it to verify that the message originates from user X_i. Subsequently, for each received $\langle ID_v, (t, \mathsf{mes}_v)\rangle_\sigma$, user X_i searches for the corresponding R_v associated with ID_v. If R_v in U cannot be found or if the round is not the current round t, user X_i disregards $\langle ID_v, (t, \mathsf{mes}_v)\rangle_\sigma$. User X_i then verifies the signature using the corresponding public key pk_v. If the signature is valid and the message is received, it is utilized for computing.

At each round, a new block $B_u^{(t+1)}$ is appended to the blockchain $BC_u^{(t+1)}$. This block records the pertinent computing information for round $t+1$. Through the combination of group membership validation and message verification, as discussed in the preceding paragraph, user X_i ensures that it only interacts with users from its computing users list U. Consequently, an adversary cannot create more Byzantine users than the number of users it controls to disrupt the computing process. Throughout this process, users communicate anonymously and securely, without revealing their identity privacy.

4) Trace & Revoke: In ADC, users have the ability to vote on the identification R_v of user X_v for tracing purposes. Once a series of votes $\langle \mathsf{vote1}\rangle_{\sigma_G}, \langle \mathsf{vote2}\rangle_{\sigma_G}, \cdots, \langle \mathsf{vote}n\rangle_{\sigma_G}$ is collected, users invoke the Trace function of the Group Contract. If the number of valid votes exceeds the threshold h, the Trace function outputs the group member identity parameter A_i. Subsequently, users can retrieve the label i associated with A_i from the blockchain, indicating that the signature of the malicious user has been successfully traced.

The process of revoking users follows a similar approach to tracing users. After collecting a series of votes $\langle \mathsf{vote1}\rangle_{\sigma_G}, \langle \mathsf{vote2}\rangle_{\sigma_G}, \cdots, \langle \mathsf{vote}n\rangle_{\sigma_G}$, users invoke the Revoke function of the Group Contract, providing the label i of the malicious user as input. If the number of valid votes exceeds the threshold h, the malicious user is revoked by updating the new dynamic accumulator parameter Acc.

V. Security Analysis

Here we will analyze the security of the system, mainly including its anonymity, unforgeability and Sybil-resistance. As for traceability, Byzantine fault tolerance and decentralization, which we mentioned in our design goal, they are designed into the system itself and do not need to be analyzed.

Theorem 1 (Anonymity): ADC does not reveal any private identity information about users.

Proof: In each decentralized computing instance, the user X_i identifies itself with a temporary identity prove P_i. If the adversary $\mathcal{V}$ cannot associate two temporary identity prove P_i and $P_i^{'}$ of user X_i in two decentralized computing instances, it means that ADC does not reveal any private identity information about users. Note that the group is not considered private because it needs to be published to verify the signature using the group public key. The following game can be used to model this scenario, where the challenger $\mathcal{C}$ acts as the ADC system and the adversary $\mathcal{V}$ attempts to break the system:

1) $\mathcal{C}$ sends identity prove P_i of user X_i in one decentralized computing instance to $\mathcal{V}$.
2) Then $\mathcal{C}$ selects a decentralized computing instance DL_u where user X_i participates.
3) $\mathcal{V}$ performs arbitrary behavior on DL_u. After finishing, $\mathcal{V}$ notifies $\mathcal{C}$.
4) $\mathcal{C}$ selects P_i' of user X_i and P_j' of user X_j on decentralized computing DL_u. X_i and X_j both come from group GR_k. Then $\mathcal{C}$ randomly chooses one of P and sends it to $\mathcal{V}$. We define $B=1$ if P comes from X_i and $B=0$ if P comes from X_j.
5) $\mathcal{V}$ predicts its guess B' and wins the game if $B'=B$.

In this game, we define the advantage that the guess is right as $Ad_{\mathcal{V}} = 2 \times \left(\Pr\left[B'=B\right] - \frac{1}{2}\right)$.

Next, we follow this game to prove the anonymity of ADC. First, $\mathcal{V}$ obtains identity prove P_i of user X_i. Then we assume decentralized computing DL_u runs normally. We recall the composition of R_i, $R_i = \langle ID_i, GR_k, pk_i\rangle_{\sigma,\sigma_G}$. We recall the composition of P_i, $P_i = \langle DL_u, R_i, \mathsf{Hash}(DL_u, A_i)\rangle_{\sigma_{contract}}$, in which $R_i = \langle ID_i, GR_k, pk_i\rangle_{\sigma,\sigma_G}$. We can see that R_i has five components, of which GR_k is public information, not considered private information. The ID_i and pk_i are random and one-time numbers, from which $\mathcal{V}$ does not get any information. The ECDSA signature is different in every decentralized computing instance. As for the group signature, according to the anonymity of group signature, $\mathcal{V}$ has no way to know exactly who signed it. Therefore, $\mathcal{V}$ cannot obtain any private information directly from the components of R_i. As for other components of P_i, they include the label of decentralized computing DL_u, hash values for the label of decentralized computing and identity information $\mathsf{Hash}(DL_u, A_i)$ and $\sigma_{\mathsf{contract}}$, where DL and $\sigma_{\mathsf{contract}}$ do not contain any privacy. For $\mathsf{Hash}(DL_u, A_i)$, since DL is different for each decentralized computing instance, according to randomness of hash function, $\mathsf{Hash}(DL_u, A_i)$ and $\mathsf{Hash}(DL_{u'}, A_i)$ are unrelated, so the hash value of (DL_u, A_i) can be regarded as a kind of random number. In summary, the guess of $\mathcal{V}$

for B can only pick a random value, i.e., $\Pr[B' = B] = \frac{1}{2}$. According to the formula of $Ad_{\mathcal{V}}$, the advantage of $\mathcal{V}$ in this game is zero. Therefore, we can conclude that ADC does not reveal any private identity information about users.

Theorem 2 (Unforgeability): An adversary $\mathcal{V}$ without a signature private key cannot masquerade as other users.

Proof: Adversary $\mathcal{V}$ may try to disguise itself as other users. This can be modeled by the following game:

1) $\mathcal{C}$ selects a decentralized computing DL_u where user X_i participates.
2) $\mathcal{V}$ performs arbitrary behavior on DL_u.
3) $\mathcal{V}$ gives a forged message $\langle ID_f, (t, \mathsf{mes}_f)\rangle_\sigma$ that is not sent by a user.
4) If the message passes validation, $\mathcal{V}$ wins this game.

If $\mathcal{V}$ wants to masquerade as a user X_f that does not participate in this decentralized computing instance, it needs to construct the corresponding R_f in the group membership validation phase. However, R_f requires a group signature using gsk_f, which $\mathcal{V}$ does not have. If $\mathcal{V}$ wants to masquerade as a user X_j that participates in this decentralized computing instance, it will not work if it tries to construct R_j, as we discussed above, so it can only do so in the next computing phase. In the computing phase, $\mathcal{V}$ wants to give a forged message $\langle ID_f, (t, \mathsf{mes}_f)\rangle_\sigma$. For ID_f, because every time users receive a message, they need to verify whether R_f corresponding to ID_f is in the computing users list U, $\mathcal{V}$ can only use ID_f sent by X_f in the group membership validation phase. However, the signature σ in message is the EDCSA signature, and $\mathcal{V}$ cannot give the corresponding signature because it does not have the corresponding signature private key sk_f. Here, we explain why group signatures are not used in this phase. Now, let's assume that if the signature σ is a group signature, and $\mathcal{V}$ and X_f belong to the same group. Due to the properties of group signatures, $\mathcal{V}$ can use its group signature private key $\mathsf{gsk}_{\mathcal{V}}$ to construct a signature that can pass verification of the group signature. This is why we need to introduce a one-time EDCSA signature in ADC. In summary, $\mathcal{V}$ cannot win this game.

Theorem 3 (Sybil-resistance): ADC is secure against Sybil attacks.

Proof: In decentralized computing, ADC has Sybil-resistance if the adversary $\mathcal{V}$ cannot construct two different messages to be accepted by other users during a single message interaction. It can be modeled by the following game:

1) $\mathcal{C}$ selects a decentralized computing instance DL_u.
2) $\mathcal{V}$ performs arbitrary behavior on DL_u.
3) $\mathcal{V}$ sends two different messages $\langle ID_{\mathcal{V}}, (t, \mathsf{mes})\rangle_\sigma$ and $\langle ID'_{\mathcal{V}}, (t, \mathsf{mes}')\rangle_\sigma$ that are not sent by users during a single message interaction.
4) If these two messages pass validation, $\mathcal{V}$ wins this game.

In group membership validation phase, if the adversary $\mathcal{V}$ wants to create two identifications R and R', according to Theorem 2, $\mathcal{V}$ cannot masquerade as another user. Therefore, $\mathcal{V}$ can only use its own group signature private key $\mathsf{gsk}_{\mathcal{V}}$ to construct R and R'. In the PRE-PREPARE phase of group membership validation, users interact with the Give Identity Prove function and get the return value $P = \langle DL, R, \mathsf{Hash}(DL, A)\rangle_{\sigma_{\text{contract}}}$. $\mathcal{V}$ will get $P_{\mathcal{V}}$ and $P'_{\mathcal{V}}$. Here, $A_{\mathcal{V}} = \mathsf{GroupSig.Open}(\sigma_G^{(\mathcal{V})})$ and $A_{\mathcal{V}'} = \mathsf{GroupSig.Open}(\sigma_G^{(\mathcal{V}')})$. Since R and R' use the same group signature private key $\mathsf{gsk}_{\mathcal{V}}$, $A_{\mathcal{V}} = A_{\mathcal{V}'}$. So $\mathsf{hash}(DL, A_{\mathcal{V}}) = \mathsf{Hash}(DL, A_{\mathcal{V}'})$. In PREPARE phase, R and R' will not be added to the set SET_R by other users because their $\mathsf{Hash}(DL, A)$ is equal. Note that if $\mathcal{V}$ wants to forge another $GR_{k'}$, the Give Identity Prove function will verify the group signature of $\mathcal{V}$ with the corresponding $\mathsf{gpk}_{k'}$. Since the group signature of $\mathcal{V}$ will not pass the verification, $\mathcal{V}$ can not get identity prove P. In computing phase, if $\mathcal{V}$ constructs two messages $\langle ID, (t, \mathsf{mes})\rangle_\sigma$ and $\langle ID', (t, \mathsf{mes}')\rangle_\sigma$, the identification R and R' corresponding to ID and ID' are not all created by $\mathcal{V}$ according to the above. Hence, if ID and ID' are both constructed from $\mathcal{V}$, then there must be a corresponding identification R that is not in the computing users list U, and one of these two messages will be rejected by the user that receives them. If $\mathcal{V}$ wants to use ID_j of another user X_j, according to Theorem 2, $\mathcal{V}$ cannot masquerade as another user. If $\mathcal{V}$ constructs two messages $\langle ID_{\mathcal{V}}, (t, \mathsf{mes})\rangle_\sigma$ and $\langle ID_{\mathcal{V}}, (t, \mathsf{mes}')\rangle_\sigma$ using the same ID, other users receive the first message and execute $C[\mathcal{V}] = C[\mathcal{V}] + 1$. At this point, $C[\mathcal{V}] = 1$. Upon receiving the second message, users check and find that $C[\mathcal{V}] = 1$, so the second message is dropped, and it is still not possible to construct two messages successfully. In summary, $\mathcal{V}$ cannot win this game.

VI. Evaluation

A. Configuration

In evaluation section, we take decentralized learning as the application scenario of ADC. This system is built upon a blockchain employing the PBFT consensus algorithm. We employ gRPC framework as the module for users communication. The ECDSA signature uses Python's ecdsa library, while the BBS04 group signature scheme is implemented using Java's Jpbc library. We utilize the open-source code of SPDL[1] for computing part. Our smart contract is implemented using Remix[2], and the code is written in Solidity[3].

We conduct experiments on a Aliyun server that has a vCPU (Intel Xeon Platinum 8269) with 24 cores (2.5GHz) and 48 GB of RAM. We test the performance of ADC using the MNIST dataset for image classification. This dataset comprises 70,000 28×28 images of handwritten digits, which is distributed across 10 different classes. The dataset is divided into N pieces, with each assigned to one user.

B. Performance

Latency of ADC: In this experiment, we denote N as the number of participating users. We evaluate the latency required for group membership validation in decentralized learning

[1]Available: https://github.com/isSPDL/SPDL
[2]Available: http://remix.ethereum.org
[3]Available: https://docs.soliditylang.org/en/v0.8.20/

(the additional latency brought by ADC compared to SPDL) and the model training latency for each round. As depicted in Fig. 5, the computation latency spent on group membership validation is relatively small compared to the latency consumed by each round of model training. Because the computing process of decentralized computing often requires many rounds, but in ADC, group membership validation only needs to be done once, so this time overhead is acceptable. In Fig. 6, we present the computation latency of the other parts, i.e. Initialization, User Registration, Trace and Revoke. From this figure, we can see that the computation latency of other parts is very small. Additionally, these parts are not executed frequently in the entire system. The Initialization is performed only once per group, User Registration is executed only once per user, and the Trace and Revoke are executed only once for malicious users. Therefore, these computation latencys are perfectly acceptable for the overall system.

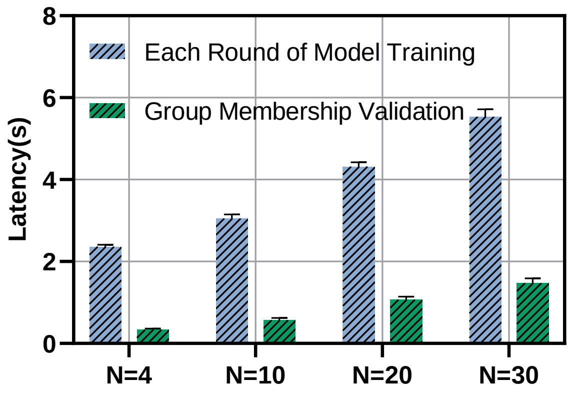

Fig. 5: Computation Latency of Decentralized Learning

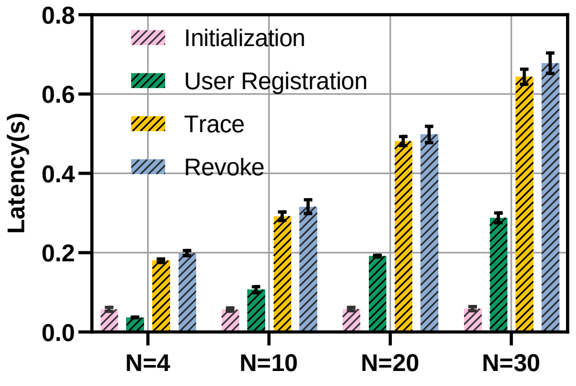

Fig. 6: Computation Latency of Other Parts

Gas Cost of Group Contract: It is important to note that the smart contract deployed on Ethereum is a bounty contract (inspired by ShadowEth [11]), and its gas cost depends primarily on the input and output of the function. In this experiment, we set the number of participating users to $N = 20$, which has an impact on the gas cost of the Trace and Revoke functions. The gas cost for executing each public function is presented in TABLE II. We can see that Trace function costs 10.09 USD and Revoke function costs 9.71 USD. Although these costs may seem high, they only need to be executed once against malicious user. If we include a deposit mechanism in practical business applications, where after the Trace and Revoke function is executed on the malicious user X_i, the deposit of X_i is used to compensate the invoking user of these functions, then this cost can be offset by this compensation. For other functions, these costs are acceptable compared to the overhead of decentralized computing itself.

TABLE II: Gas Cost of Smart Contract

Function	Gas Cost	ETH Cost	USD
InitializeGroup	237689	0.00024	0.46
RegisterUser	184986	0.00018	0.34
Trace	5293656	0.0053	10.09
Revoke	5068370	0.0051	9.71
GiveIdentityProof	617333	0.00062	1.18

1 Gas = 1 Gwei, 1 ETH = 1903.79 USD

Storage: In this experiment, we present the storage costs for each component in different phases of decentralized learning in ADC. In group membership validation, we set the size of the ECDSA public key to 96 bytes, considering it to be a reasonably secure size. The ECDSA signature size is 96 bytes, while the BBS04 [23] signature size is 191 bytes. The size of GR is set to 1 byte, accommodating 256 groups. Both ID and DL are set to 8 bytes. In summary, the total size of R is 392 bytes, and the total size of P is 528 bytes. In computing, we assume that the label of each round occupies 2 bytes, allowing support for up to 65536 rounds. Additionally, we denote the number of users in decentralized learning as N, and the size of mes as $|\mathsf{mes}|$. We calculate the storage costs and present them in TABLE III. From this table, we observe that the storage cost in group membership validation is approximately $392N$ bytes, and the additional message size is around 106 bytes. These storage sizes are relatively small and acceptable for large-scale decentralized computing.

TABLE III: Storage Cost

	Components	Storage cost
Group Membership Validation	R	392 bytes
	P	528 bytes
	U	$392N$ bytes
Computing	U	$392N$ bytes
	C	N bits
	sent message	$106 + |\mathsf{mes}|$ bytes

Byzantine Fault Tolerance: We test the Byzantine fault tolerance of ADC by training the model with different Byzantine Ratios (BR), which represents the ratio of the number of Byzantine users to the total number of users. In this experiment, Byzantine users attempt to interfere with training

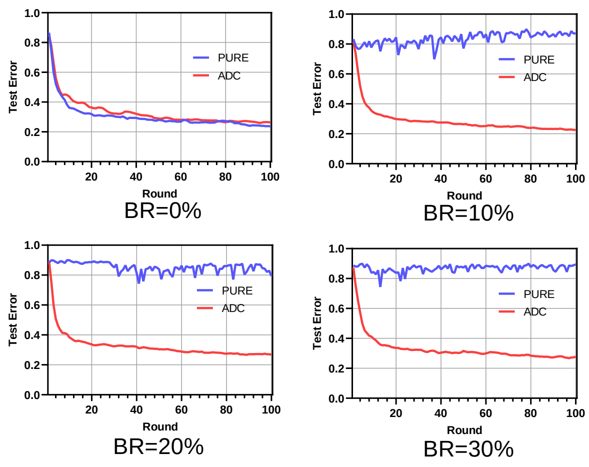

Fig. 7: Byzantine Fault Tolerance

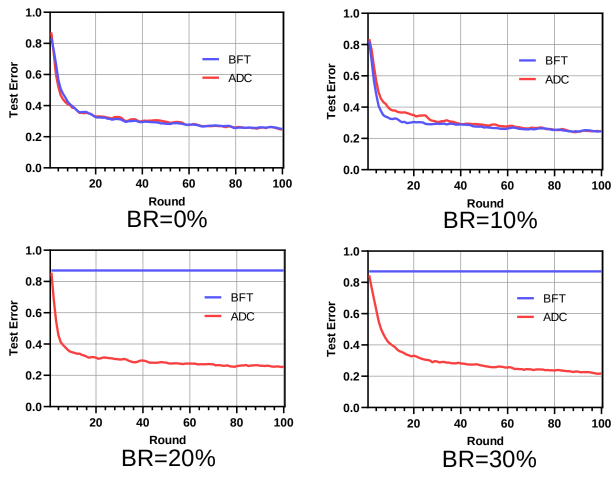

Fig. 8: Sybil Attacks Resistance

effect by sending extremely large gradients, and interfere with consensus process by not sending any messages after the PREPARE phase in the PBFT algorithm of "ADC". We set the total number of users to 20, and the BR values were 0%, 10%, 20%, and 30%, considering $f = 33\% \times N$, which is the upper limit of the PBFT algorithm. We show the test error (the proportion of incorrect predictions made on the test dataset out of the total number of predictions made) of the model training on Fig. 7. In this figure, we denote "PURE" as the system with all Byzantine fault tolerance related components removed, including group membership validation in ADC, BFT GARs, PBFT algorithm for model training, and the blockchain system. From the figure, we can observe that the "PURE" system is directly disrupted in the presence of Byzantine users within the system, leading to the inability of the test error to converge. In contrast, our "ADC" system, which incorporates Byzantine fault tolerance, demonstrates the ability to achieve convergence to a favorable test error value even when the number of Byzantine users is below 30%. So, one can get that ADC has good Byzantine fault tolerance.

Sybil Attacks Resistance: We extend the Byzantine fault-tolerance experiment to test the Sybil attacks resistance of ADC. We make each Byzantine user launch Sybil attacks and they both create two additional Byzantine users to join ADC. Byzantine users behave in this experiment as in the Byzantine fault tolerance experiment. In Fig. 8, we show the test error for different Byzantine Ratio (BR) in this scenario, where we denote "BFT" as the system that removes ADC's procedures for handling Sybil attacks. In this figure, "BFT" in the case of BR=20% and BR=30%, their test error is in a straight line at a higher position. This is because Byzantine user percentage has exceeded 33% at this case due to Sybil attacks, and "BFT" cannot be trained properly (It exceeds the upper limit of the PBFT algorithm, making it impossible to reach consensus among users because Byzantine users interfere with consensus process by not sending any messages after the PREPARE phase). However, comparing with the Byzantine fault tolerance experiments above, we can see that "BFT" can get good training results at this byzantine ratio without the presence of Sybil attacks. This underscores the significance of addressing Sybil attacks in our system.

VII. Conclusion

In this paper, we propose ADC, an anonymous system for decentralized computing. It provides anonymity while also ensuring traceability and revocability, as well as maintaining good Sybil-resistance and Byzantine fault tolerance. Furthermore, ADC can achieve the above functions without the need for trusted authority. Finally, the analysis and evaluation show that ADC has strong security and practicality with only a minor additional time overhead.

In our future research, we will explore how the private smart contract in ADC can be simplified to a regular smart contract as a way to increase the scalability. Additionally, we also intend to explore blockchain consensus algorithms for decentralized learning, one idea is to utilize zero-knowledge machine learning to use the computation in machine learning as a proof of work for users.

References

[1] Lian, X., Zhang, C., Zhang, H., Hsieh, C. J., Zhang, W., & Liu, J. (2017). Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. Advances in neural information processing systems, 30.

[2] Mouton, F., Leenen, L., & Venter, H. S. (2016). Social engineering attack examples, templates and scenarios. Computers & Security, 59, 186-209.

[3] Chen, X., Ji, J., Luo, C., Liao, W., & Li, P. (2018, December). When machine learning meets blockchain: A decentralized, privacy-preserving and secure design. In 2018 IEEE international conference on big data (big data) (pp. 1178-1187). IEEE.

[4] Minghui Xu, Zongrui Zou, Ye Cheng, Qin Hu, Dongxiao Yu, and Xiuzhen Cheng. "SPDL: A Blockchain-Enabled Secure and Privacy-Preserving Decentralized Learning System." IEEE Transactions on Computers, vol. 72, no. 2, 2023, pp. 548-558.

[5] Douceur, J. R. (2002, March). The sybil attack. In International workshop on peer-to-peer systems (pp. 251-260). Berlin, Heidelberg: Springer Berlin Heidelberg.

[6] Fan, M., Zhang, Z., Li, Z., Sun, G., Yu, H., & Guizani, M. (2023). Blockchain-Based Decentralized and Lightweight Anonymous Authentication for Federated Learning. IEEE Transactions on Vehicular Technology.

[7] Shayan, M., Fung, C., Yoon, C. J., & Beschastnikh, I. (2020). Biscotti: A blockchain system for private and secure federated learning. IEEE Transactions on Parallel and Distributed Systems, 32(7), 1513-1525.

[8] Chunchi Liu, Minghui Xu, Hechuan Guo, Xiuzhen Cheng, Yinhao Xiao, Dongxiao Yu, Bei Gong, Arkady Yerukhimovich, Shengling Wang, and Weifeng Lyu. ”TBAC: A Tokoin-based Accountable Access Control Scheme for the Internet of Things.” IEEE Transactions on Mobile Computing, 2023, pp. 1-16.

[9] Chunchi Liu, Hechuan Guo, Minghui Xu, Shengling Wang, Dongxiao Yu, Jiguo Yu, and Xiuzhen Cheng. ”Extending On-Chain Trust to Off-Chain – Trustworthy Blockchain Data Collection Using Trusted Execution Environment (TEE).” IEEE Transactions on Computers, vol. 71, no. 12, 2022, pp. 3268-3280.

[10] Fan, C. I., Hsu, R. H., & Manulis, M. (2011). Group signature with constant revocation costs for signers and verifiers. In Cryptology and Network Security: 10th International Conference, CANS 2011, Sanya, China, December 10-12, 2011. Proceedings 10 (pp. 214-233). Springer Berlin Heidelberg.

[11] Yuan, R., Xia, Y. B., Chen, H. B., Zang, B. Y., & Xie, J. (2018). Shadoweth: Private smart contract on public blockchain. Journal of Computer Science and Technology, 33, 542-556.

[12] Huayi Qi, Minghui Xu, Dongxiao Yu, and Xiuzhen Cheng. ”SoK: Privacy-preserving smart contract.” High-Confidence Computing, vol. 4, no. 1, 2024, p. 100183. doi: https://doi.org/10.1016/j.hcc.2023.100183

[13] Castro, M., & Liskov, B. (1999, February). Practical byzantine fault tolerance. In OsDI (Vol. 99, No. 1999, pp. 173-186).

[14] Qu, Y., Gao, L., Luan, T. H., Xiang, Y., Yu, S., Li, B., & Zheng, G. (2020). Decentralized privacy using blockchain-enabled federated learning in fog computing. IEEE Internet of Things Journal, 7(6), 5171-5183.

[15] Lu, Y., Huang, X., Dai, Y., Maharjan, S., & Zhang, Y. (2019). Blockchain and federated learning for privacy-preserved data sharing in industrial IoT. IEEE Transactions on Industrial Informatics, 16(6), 4177-4186.

[16] Xu, C., Qu, Y., Eklund, P. W., Xiang, Y., & Gao, L. (2021, September). Bafl: An efficient blockchain-based asynchronous federated learning framework. In 2021 IEEE Symposium on Computers and Communications (ISCC) (pp. 1-6). IEEE.

[17] Warnat-Herresthal, S., Schultze, H., Shastry, K. L., Manamohan, S., Mukherjee, S., Garg, V., ... & Schultze, J. L. (2021). Swarm learning for decentralized and confidential clinical machine learning. Nature, 594(7862), 265-270.

[18] Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., & Gao, Y. (2021). A survey on federated learning. Knowledge-Based Systems, 216, 106775.

[19] Hasırcıoğlu, B., & Gündüz, D. (2021, June). Private wireless federated learning with anonymous over-the-air computation. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 5195-5199). IEEE.

[20] Zhao, B., Fan, K., Yang, K., Wang, Z., Li, H., & Yang, Y. (2021). Anonymous and privacy-preserving federated learning with industrial big data. IEEE Transactions on Industrial Informatics, 17(9), 6314-6323.

[21] Jiang, Y., Zhang, K., Qian, Y., & Zhou, L. (2022). Anonymous and efficient authentication scheme for privacy-preserving distributed learning. IEEE Transactions on Information Forensics and Security, 17, 2227-2240.

[22] Chaum, D., & Van Heyst, E. (1991). Group signatures. In Advances in Cryptology—EUROCRYPT’91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10 (pp. 257-265). Springer Berlin Heidelberg.

[23] Boneh, D., Boyen, X., & Shacham, H. (2004, August). Short group signatures. In Annual international cryptology conference (pp. 41-55). Berlin, Heidelberg: Springer Berlin Heidelberg.

[24] Camenisch, J., Kohlweiss, M., & Soriente, C. (2009). An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Public Key Cryptography–PKC 2009: 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings 12 (pp. 481-500). Springer Berlin Heidelberg.

[25] Johnson, D., Menezes, A., & Vanstone, S. (2001). The elliptic curve digital signature algorithm (ECDSA). International journal of information security, 1, 36-63.

[26] Devidas, S., Rao YV, S., & Rekha, N. R. (2021). A decentralized group signature scheme for privacy protection in a blockchain. International Journal of Applied Mathematics and Computer Science, 31(2).

[27] Devidas, S., Rukma Rekha, N., & Subba Rao, Y. V. (2022, November). Dynamic Decentralized Group Signature Scheme for Privacy Protection in Blockchain. In International Conference on Innovative Computing and Communications: Proceedings of ICICC 2022, Volume 3 (pp. 745-760). Singapore: Springer Nature Singapore.

[28] Cao, Y., Li, Y., Sun, Y., & Wang, S. (2019, July). Decentralized group signature scheme based on blockchain. In 2019 International Conference on Communications, Information System and Computer Engineering (CISCE) (pp. 566-569). IEEE.

[29] Zhao, X., Wang, L., Wang, L., & Lu, Z. (2022, December). A Privacy-Enhanced Federated Learning Scheme with Identity Protection. In 2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys) (pp. 1188-1195). IEEE.

[30] Yuan, L., He, Q., Tan, S., Li, B., Yu, J., Chen, F., ... & Yang, Y. (2021, April). Coopedge: A decentralized blockchain-based platform for cooperative edge computing. In Proceedings of the Web Conference 2021 (pp. 2245-2257).