SecureDFL: A Secure Distributed Federated Learning Framework Against Poisoning Attacks

Amir Javadpour, Forough Ja'fari, Tarik Taleb, Fatih Turkmen, Chafika Benzaïd, Mohammad Shojafar

***Abstract*—Distributed Federated Learning (DFL) extends federated learning from a single-server topology to broader graph-based coordination settings that may include multiple interacting aggregation nodes and, in some deployments, externally distributed training resources. While this broader topology improves scalability and flexibility, it also enlarges the poisoning attack surface across participants, aggregators, communication channels, and execution resources. This paper presents SecureDFL, a defense-in-depth framework for poisoning-resilient DFL. SecureDFL integrates authenticated communication, protected aggregation based on additive homomorphic encryption, trust-aware orchestration, multi-point validation, and reinforcement-learning-based mutation of resource mappings under a unified system design. The contribution of this work therefore lies in the integrated security architecture and its end-to-end implementation for graph-based DFL, rather than in introducing a standalone cryptographic primitive. Experiments on benchmark datasets, seven threat-model-aligned poisoning scenarios, ablation studies, and a prototype-scale Kubernetes testbed show that SecureDFL preserves strong model performance under attack and substantially improves attack resilience relative to the evaluated baselines, achieving up to a 98.5% reduction in attack success rate in the reported settings with only modest additional overhead. These results support the practical feasibility of the proposed architecture, while the manuscript explicitly delineates its current limits regarding comprehensive collusion analysis, colluding trust manipulation, extremely heterogeneous non-IID conditions, and fully adaptive multi-stage attacks.**

***Index Terms*—Distributed Federated Learning (DFL), Poisoning Attack, Reinforcement Learning, Resource Allocation and Mutation, SecureDFL.**

I. Supplementary Demonstration

To improve transparency and presentation quality, we provide a short optional supplementary video (**HERE**) demonstrating the main components, message flows, and representative attack paths of SecureDFL. All core architectural and security claims remain fully self-contained in the manuscript, so the video should be viewed only as optional supplementary material rather than as a prerequisite for understanding the system.

II. Introduction

MACHINE learning (ML) has made tremendous strides thanks to advances in deep learning and natural language processing (NLP). These advances stem from improved algorithms and the growing availability of data and compute. To exploit distributed data while preserving privacy, federated learning decentralizes training across participants [1, 2, 3, 4].

Federated learning is typically a two-phase process. In the *training phase*, participants train local models on their own data. In the *aggregation phase*, an aggregator collects the local models and generates a global model by combining them [5, 6]. Due to privacy requirements or constraints set by data owners, the aggregator provides a general model structure for local training, and participants subsequently return the updated model parameters [7]. The aggregator is crucial in organizing and optimizing data to create a reliable and effective global model, enabling valuable insights and predictions for diverse applications.

Amir Javadpour (Senior Cybersecurity Researcher MOSA!C Lab / ICTFICIAL Oy, Finland). **Forough Ja'fari** (Sharif university of technology). **Tarik Taleb** (Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, Bochum). **Fatih Turkmen** (University of Groningen, The Netherlands). **Chafika Benzaïd** (Faculty of Information Technology and Electrical Engineering, University of Oulu) **Mohammad Shojafar** is with the 5GIC&6GIC, Institute for Communication Systems (ICS), University of Surrey, Guildford, GU27XH, United Kingdom (e-mail: m.shojafar@surrey.ac.uk).

Corresponding author: Amir Javadpour (a.javadpour87@gmail.com)

We distinguish the DFL setting considered in this paper from conventional hierarchical or decentralized FL through an explicit graph interpretation. Let $G = (V, E)$ denote the coordination graph, where $V = V_{\mathrm{p}} \cup V_{\mathrm{a}} \cup V_{\mathrm{r}}$ contains participants, aggregators, and optional execution resources. We use the term *distributed federated learning* (DFL) for settings in which either 1) the aggregation subgraph induced by V_{a} contains multiple interacting aggregation nodes, or 2) the local training path of at least one participant is itself distributed over external resources in V_{r} rather than remaining on a single local device. Under this definition, classical hierarchical FL is a special tree-structured case of DFL with a fixed aggregation hierarchy and locally self-contained training, whereas the setting targeted by SecureDFL is the more general graph-based case in which both aggregation and execution placement may be distributed and dynamically reconfigured.

The key contributions of this paper are summarized as follows:

- **Threat modeling and defense:** We define seven realistic DFL attack scenarios, including client, aggregator, resource-level, and communication-channel poisoning, and show how SecureDFL mitigates these threats.
- **Layered secure architecture:** We propose SecureDFL, which integrates cryptographically secured communication, RL-based resource mutation, and multi-point validation to protect clients, aggregators, resources, and channels.
- **Adaptive resource mutation:** We introduce an RL-driven mutation engine that dynamically changes resource mappings to reduce attack persistence and limit adversarial impact.
- **Prototype implementation:** We implement SecureDFL using open-source tools on real distributed assets, demonstrating prototype-scale deployability, reproducibility, and practical workflow integration.
- **Experimental validation:** We evaluate SecureDFL against representative baseline defenses across multiple datasets and threat settings, showing improved robustness in terms of accuracy, attack success rate, overhead, and scalability.

A. Motivation and Objectives

The growth of distributed data and privacy-preserving collaboration has increased the need for federated learning. However, conventional federated learning often depends on centralized aggregation, creating scalability bottlenecks, single points of failure, and trust risks. Decentralized federated learning (*DFL*) mitigates these issues by distributing both training and aggregation. Nevertheless, DFL also expands the attack surface across clients, aggregators, resources, and communication links, making poisoning attacks harder to detect, especially under adaptive adversaries or changing topologies. Existing defenses typically address isolated components, such as malicious-client detection or robust aggregation, and may be insufficient for fully distributed environments [8, 9, 10].

Therefore, this paper proposes SecureDFL, a secure DFL framework for resilient collaborative learning under poisoning attacks. The main objectives are to define the extended DFL model, characterize poisoning threats across distributed components, and design a unified defense mechanism based on RL-driven mutation, secure communication, and distributed trust management. The proposed framework aims to improve attack resilience while maintaining learning accuracy, scalability, and low operational overhead.

B. Illustrative Example: Hybrid DFL

To clarify the extended definition of DFL, we present a conceptual scenario. Imagine a healthcare consortium where multiple hospitals

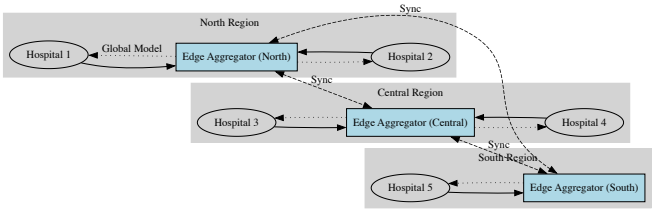


Fig. 1. Conceptual architecture illustrating hybrid DFL with decentralized aggregation and distributed training.

collaborate to train a disease prediction model without sharing sensitive patient data. In this hybrid DFL setup, aggregation is decentralized among several regional edge servers (e.g., North, Central, South). Each edge server collects model updates from hospitals in its region, performs local aggregation, and periodically synchronizes with other edge servers to construct the global model. Within each hospital, the training process can also be distributed across internal compute nodes or departments, increasing resilience to node or data failures. This multi-level decentralization demonstrates the flexibility and scalability of DFL beyond conventional federated learning, and highlights the necessity of new security mechanisms that address threats to aggregators, participants, resources, and communication channels. Figure 1 illustrates this architecture.

III. Scope, Novelty, and Practical Limitations of SecureDFL

SecureDFL is designed as a defense-in-depth architecture for poisoning-resilient DFL. Its main novelty lies in combining authenticated communication, trust-aware orchestration, and reinforcement-learning-based resource mutation under one consistent threat model that explicitly spans participants, aggregators, channels, and physical resources. In other words, the claimed contribution is the integrated systemization of multiple protection layers for graph-based DFL, rather than the introduction of a brand-new primitive in isolation. The scope of this paper is limited to poisoning-oriented integrity attacks and the operational security of distributed training and aggregation. While message confidentiality and authenticity are supported through RSA signatures, token-based authorization, and Paillier-enabled protected aggregation, we do not claim a complete treatment of privacy inference, stealth backdoors, or full collusion analysis. These aspects are discussed as important follow-on directions rather than as validated contributions of the prototype evaluated here.

A. Paper Structure

The rest of this paper is organized as follows. Section Section IV reviews and compares the most relevant prior studies on poisoning-resilient federated learning and distributed federated learning. The threat model and the seven specific attack scenarios are defined in subsection V-A. The proposed framework, communication protocol, and mutation technique are presented in Section VII, while Section XIII reports the testbed and evaluation results. Finally, Section Section XVII summarizes the main findings and outlines future research directions.

IV. Related Work

Prior work on poisoning-resilient federated learning is extensive, but most published defenses still assume a conventional FL setting in which malicious behavior originates mainly from participants and aggregation is coordinated by a single logical server. From this perspective, the literature can be grouped into robust aggregation and outlier filtering, trust/reputation management, privacy-preserving verification, and blockchain-assisted accountability. These directions are valuable, but they generally do not model compromised external training resources, mutable communication paths, or partially malicious aggregators in a unified DFL system model.

This distinction matters because the SecureDFL setting is not restricted to a hierarchical tree of honest relays. Once training execution and aggregation are distributed over a broader coordination graph, poisoning can propagate through resource placement, message forwarding, or cross-aggregator synchronization. We therefore compare prior work not only by defended attack type, but also by the attack origin, deployment assumptions, and suitability for graph-based DFL. A large body of research has examined how to secure federated learning frameworks against both training-data leakage and malicious model manipulation. Some studies mainly emphasize privacy protection, such as Hao et al. [11] and Wu et al. [12], while others address poisoning attacks, client dropout, or both simultaneously Ghavamipour et al. [13]. Due to the focus of this paper on poisoning attacks against federated learning approaches, the most relevant studies are reviewed in this section. To make the comparison more informative for the present threat model, the discussion below also contrasts prior studies in terms of secure communication support, aggregation protection, trust/validation logic, adaptive mutation capability, and their closest alignment to the seven attack scenarios defined in Section subsection V-A. A summary of these works is shown in Table I.

Several works address poisoning by malicious participants. FoolsGold Fung et al. [8] detects colluding attackers by measuring update similarity and diversity. Liu *et al.* Liu et al. [14] use blockchain smart contracts to identify and exclude unreliable participants. So *et al.* So et al. [15] improve robustness and privacy by quantizing and masking updates, then filtering received models using distance checks against a gradient estimator.

Instead of assuming intentionally malicious participants, Wang et al. [16] study cost-saving participants who submit low-quality updates. Although the motivation differs, the aggregator still needs to prevent harmful updates from degrading the global model. Their method uses reinforcement learning to incentivize high-quality contributions. Biscotti Shayan et al. [9] introduces a privacy-preserving FL defense in which a verification team checks participant updates using Multi-Krum and signs verified updates to guide the aggregator's selection. ShieldFL Ma et al. [10] detects malicious updates through cosine-similarity analysis while using homomorphic encryption to preserve participant privacy.

Most of these studies focus on poisoning from malicious participants, whereas VerifyNet Xu et al. [17] addresses malicious aggregators. It enables participants to verify whether the aggregator honestly updates the global model by requiring the aggregator to provide a proof that is claimed to be computationally infeasible to forge.

Recent studies have continued to strengthen poisoning resilience mainly at the participant side. For example, Kolasa et al. [7] focus on malicious-client detection and secure validation of the learning process, Lai [20] propose a staged defense against poisoned updates, Xu and Shu [21] improve aggregation robustness through variance-aware weighting, and Khraisat [22] address targeted poisoning attempts against local client updates. These directions improve robustness under conventional poisoning assumptions, but they still remain primarily aligned with participant-origin threats conceptually closest to ATTS2 and ATTS3, rather than the broader multi-surface setting addressed in this paper.

Recent works such as FLAME [18] and FedDF [19] have introduced advanced defenses against backdoor and model poisoning attacks, primarily targeting participant-level threats. As reflected in Table I, most prior studies remain closest to ATTS2/3-like or ATTS5/6-like settings, because they are designed around either malicious participants or malicious aggregators, but not both together with compromised resources and channels. However, these frameworks do not address resource or aggregator compromise jointly with channel manipulation, which are explicitly covered by the multi-layered security mechanisms of SecureDFL.

Prior secure federated learning works usually assume a single poisoning origin and fixed local training resources, while overlooking aggregator, communication-channel, and external-resource threats. They also lack the adaptive resource mutation used in SecureDFL for ATTS1 and do not unify participant authorization, aggregator trust, and communication integrity within one scenario-based threat model. Thus, they provide limited multi-surface protection for dynamic and heterogeneous DFL

TABLE I
COMPARISON OF POISONING-RESILIENT FL/DFL STUDIES UNDER SECUREDFL-ORIENTED CRITERIA. ABBREVIATIONS: C = CENTRALIZED, D = DISTRIBUTED, P = PARTICIPANTS, A = AGGREGATORS, R = RESOURCES, CH = COMMUNICATION CHANNELS, COMM. = SECURE COMMUNICATION, AGG. = PROTECTED AGGREGATION, TRUST = TRUST/VALIDATION MECHANISM, MUT. = ADAPTIVE MUTATION, AND EVAL. = EVALUATION SCOPE. "CLOSEST ATTS ALIGNMENT" DENOTES CONCEPTUAL OVERLAP WITH ATTS1–ATTS7, NOT EXACT SCENARIO EQUIVALENCE.

Representative Work	Type	P	A	R	Ch	Comm.	Agg.	Trust	Mut.	Closest ATTS Alignment	Eval.	Main Limitation Relative to SecureDFL
FoolsGold [8]	C	Yes	No	No	No	No	No	No	No	ATTS2/3-like	Bench.	Detects Sybil-style poisoning through update similarity, but does not address compromised aggregators, execution resources, or communication paths.
Liu et al. [14]	C	Yes	No	No	No	Partial	No	Yes	No	ATTS2/3-like	Bench.	Improves participant accountability through blockchain-based trust logic, but does not provide unified protection across multiple DFL attack surfaces.
So et al. [15]	C	Yes	No	No	No	Partial	Partial	Partial	No	ATTS2/3-like	Bench.	Strengthens robustness against Byzantine participants, but remains focused on participant-side filtering under fixed FL assumptions.
Wang et al. [16]	D	Yes	No	No	No	No	No	Partial	Partial	ATTS2/3-like	Bench.	Uses RL to encourage better participant behavior, but not for dynamic resource reassignment or communication-path mutation.
Biscotti [9]	D	Yes	No	No	No	Partial	Partial	Yes	No	ATTS2/3-like	Bench.	Provides distributed validation and accountability, but does not model compromised resources or mutable channels.
ShieldFL [10]	C	Yes	No	No	No	Yes	Partial	Partial	No	ATTS2/3-like	Bench.	Protects privacy and filters malicious updates, but mainly targets participant poisoning in conventional FL settings.
VerifyNet [17]	C	No	Yes	No	No	Partial	Yes	Yes	No	ATTS5/6-like	Bench.	Strengthens trust in the aggregator, but does not address compromised resources, communication channels, or adaptive reconfiguration.
FLAME [18]	C	Yes	No	No	No	No	Partial	No	No	ATTS2/3-like	Bench.	Provides strong backdoor/model-poisoning filtering, but remains largely limited to participant-origin attacks in centralized FL.
FedDF [19]	C	Yes	No	No	No	Partial	Partial	No	No	ATTS2/3-like	Bench.	Improves poisoning resilience with privacy-aware filtering, but does not jointly consider aggregators, resources, and channels.
Kolasa *et al.* [7]	C	Yes	No	No	No	Partial	Partial	Yes	No	ATTS2/3-like	Bench.	Concentrates on malicious-client detection and secure validation, but not on graph-based DFL orchestration with multi-surface protection.
Lai [20]	C	Yes	No	No	No	No	Partial	Partial	No	ATTS2/3-like	Bench.	Uses a staged poisoning defense, but still focuses on participant-level threats and static aggregation assumptions.
Xu [21]	C	Yes	No	No	No	No	Partial	Partial	No	ATTS2/3-like	Bench.	Improves aggregation robustness through variance-aware weighting, but does not include secure communication, resource protection, or adaptive mutation.
Khraisat [22]	C	Yes	No	No	No	No	Partial	Partial	No	ATTS2/3-like	Bench.	Targets poisoned client updates, but does not address broader poisoning propagation through aggregators, resources, and channels.
SecureDFL (this work)	**D**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**ATTS1–ATTS7**	**7 scenarios**	**Provides a unified defense-in-depth architecture that combines authenticated communication, protected aggregation, trust-aware orchestration, second-device/token validation, and RL-based resource mutation for graph-based DFL.**

settings. In contrast, SecureDFL provides multi-surface protection across participant, aggregator, resource, and channel-level threats. By integrating secure communication, RL-driven resource mutation, and cryptographic validation, it improves robustness against diverse poisoning attacks while supporting scalable prototype-scale deployment.

A. Byzantine-Robust Decentralized Federated Learning and Unlearning

Recent studies have increasingly emphasized that decentralized federated learning (DFL) requires robustness mechanisms that differ from conventional server-centered federated learning. In centralized federated learning, a server can compare, filter, or aggregate client updates using a global view of the learning process. In contrast, DFL removes the central coordinator and relies on peer-to-peer communication and local aggregation, which makes Byzantine behavior more difficult to isolate. This distinction is important when malicious clients manipulate their local data, model updates, neighborhood messages, or unlearning-related interactions. Therefore, robustness in DFL should be evaluated not only under standard single-surface poisoning attacks, but also under more adaptive adversarial behavior.

Recent Byzantine-robust DFL studies directly motivate this extension. BALANCE [23] introduces Byzantine-robust averaging through local similarity in decentralization, where each client uses its own local model as a reference to identify suspicious neighboring updates before aggregation. WFAgg [24] further emphasizes that many Byzantine-robust aggregation rules designed for centralized FL are not directly transferable to decentralized environments and therefore require DFL-aware filtering mechanisms. Related robust FL studies such as FLTrust [25] and local model poisoning analysis [26] also show that attackers may adapt their local updates to bypass robust aggregation rules, which reinforces the need to evaluate stronger adaptive adversarial settings.

Federated unlearning introduces an additional security dimension. Recent work on poisoning attacks against federated unlearning shows that malicious participants may exploit the unlearning stage itself by sending specially crafted updates that prevent the unlearned model from removing poisoned effects. UnlearnGuard [27] addresses this issue by estimating and filtering client updates during unlearning, while FedUP [28] studies efficient pruning-based federated unlearning for model-poisoning recovery. These studies are closely related to the present work because they show that robustness must be considered across both the learning and unlearning phases. However, our focus differs in that we evaluate how SecureDFL behaves when the adversary adapts its poisoning strategy across rounds and attack surfaces. This motivates the additional adaptive-adversary evaluation introduced in the revised experimental section.

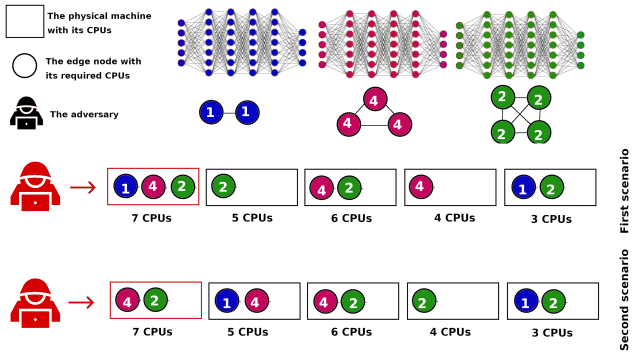
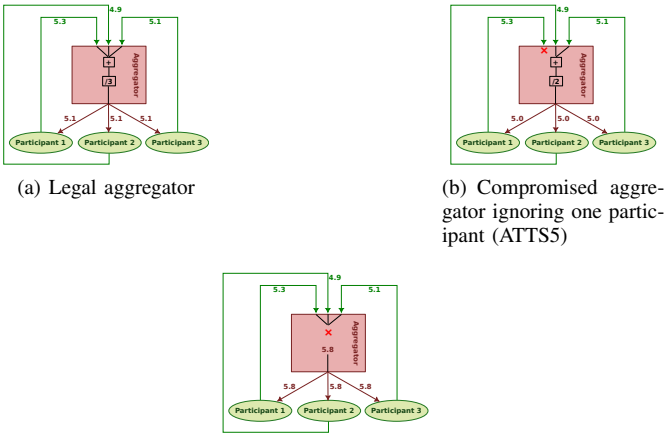


Fig. 2. The impact of compromised resources on the trained models.



(a) Legal aggregator

(b) Compromised aggregator ignoring one participant (ATTS5)

(c) Compromised aggregator ignoring all participants (ATTS6)

Fig. 3. How compromised aggregators affect the aggregated model.

V. Threat Model Justification and Scope

While federated learning systems face various threats including inference attacks, backdoor attacks, and eavesdropping this work focuses primarily on poisoning attacks. Poisoning attacks are among the most prevalent and disruptive threats in DFL, as they directly target the integrity of the global model by injecting manipulated or malicious updates from compromised participants or aggregators. Numerous studies have shown that poisoning attacks can bypass standard aggregation or privacy mechanisms, causing significant degradation or subversion of the model even in otherwise "secure" deployments [8, 9, 10]. Nonetheless, SecureDFL provides partial protection against additional threats. Secure communication and cryptographic validation mitigate eavesdropping and replay attacks on data in transit, while RL-based mutation of resource assignments complicates adversarial coordination. Although inference attacks and advanced backdoors are beyond our main scope, we discuss potential extensions of SecureDFL to address them in the Discussion and Future Work sections.

A. Threat Model

In this section, we define four classes of poisoning attacks that threaten a federated learning framework, and seven attack scenarios based on them. Poisoning attacks can come from various sources that can be broadly divided into four classes: (1) the resources, (2) the participants, (3) the communication channels, and (4) the aggregators. In what follows, we provide more details about the scenarios we consider in this paper.

Attack 1 (Compromised Resources)**.** *In this attack scenario, the adversary compromises the resources assigned to handle a portion of the training process. If a resource is compromised, the adversary can control what output it generates. Hence, false information can be sent to the participant or the next resource in the chain, resulting in a poisoning attack.*

According to Attack 1, if the adversaries compromise a resource, they can poison all the models that are involved in that resource. For example, based on Figure 2, we can see that each of the three participants are requesting different models to be trained. The first model, the blue one, has two tasks requiring one CPU each. The second and third ones have three and four tasks to be handled, respectively. In the first assignment scenario, we can see that a task of each of these models resides on the first physical resource. Hence, when the adversary compromises this resource, all three models can be poisoned. However, in the second assignment scenario, at most, two tasks of each model share the same resource. In this case, if the adversary compromises one of the resources, only two models are affected. Based on this attack type, we define the attack scenario **ATTS1** as the case where the adversary compromises a physical machine and modifies all its outgoing traffic in any way they desire.

Attack 2 (Compromised Participants)**.** *In this threat, the adversary compromises a participant workstation, and when it is time to send the trained model information to the aggregators, the compromised participant sends false information. Therefore, the global model of the aggregators is updated with false information, and it is poisoned.*

It is worth noting that our threat model assumes the human agents of the participants are trusted and do not intentionally send false information. Based on this attack type, we define two attack scenarios. **ATTS2** refers to the case where the adversary compromises a participant's workstation, which is not an aggregator, and modifies all its outgoing traffic in any way they want. **ATTS3** is the same, but this time, the compromised participant is also an aggregator.

Attack 3 (Compromised Channels)**.** *This attack type refers to Man-in-the-Middle (MitM) attacks launched against the communication channel between participants and aggregators. The adversary can modify the data sent from the participants to the aggregators and vice versa.*

According to this attack type, we define attack scenario **ATTS4** as the case where the adversary modifies the transferred messages in any way they want.

Attack 4 (Compromised Aggregators)**.** *In this attack type, the aggregator is under the adversary's control, and the aggregated model can be poisoned. In other words, the adversary decides what to present as the aggregated model.*

According to Attack 4, adversaries may present a completely different model or intentionally ignore one or more participants when updating the general model. Hence, we have two different attack scenarios: **ATTS5** and **ATTS6**. In the former, the adversary just ignores some of the received data in the aggregation process. In the latter scenario, the adversary completely changes the aggregated data. A sample of these two scenarios is shown in Figure 3(b) and Figure 3(c). The former is related to ATTS5, and the adversary ignores the updates from one of the first participants, resulting in a different output. In the latter one, which is related to ATTS6, the whole participant is ignored, and the adversary sends the desired output. We define another attack scenario, **ATTS7**, through which the adversary has compromised a participant-aggregator's workstation with access to its private data.

A summary of the considered attack scenarios, together with the attacked component and the corresponding adversarial action, is provided in Table II.

B. Adaptive Adversary Setting

In practical DFL environments, adversaries may not follow a fixed poisoning pattern. Instead, they can adapt their behavior across communication rounds according to the observed learning dynamics, model stability, and expected defense responses. Therefore, this subsection considers an adaptive adversary setting in which malicious participants can vary the attack surface, activation time, and poisoning intensity over time.

TABLE II
ATTACK SCENARIOS CONSIDERED IN THIS PAPER FOR POISONING DISTRIBUTED FEDERATED LEARNING.

Scenario	Target component	Target layer	Adversary action	Security concern	Intended effect
ATTS1	Physical resources	Resource layer	The adversary modifies the outgoing traffic of a compromised physical resource.	Integrity and robustness	The goal is to poison the training process through manipulated execution resources and affect model updates before they are propagated to the rest of the system.
ATTS2	Participant workstations	Participant layer	The adversary modifies the outgoing traffic of a participant that does not act as an aggregator.	Integrity and authenticity	The goal is to inject forged or manipulated participant-side information so that poisoned updates influence the distributed training process.
ATTS3	Participant workstations	Participant and aggregator layers	The adversary modifies the outgoing traffic of a participant that also serves as an aggregator.	Integrity and trust	The goal is to exploit the dual role of the compromised node in order to amplify poisoning influence over both local update generation and aggregation-related decisions.
ATTS4	Communication channels	Communication layer	The adversary modifies messages exchanged between any two system components.	Integrity and confidentiality	The goal is to tamper with data in transit, undermine message integrity, and indirectly influence training or coordination behavior.
ATTS5	Aggregators	Aggregator layer	The adversary intentionally ignores selected participant data during the aggregation phase.	Integrity and fairness	The goal is to bias the aggregation outcome by selectively excluding legitimate updates and thereby distort the global model.
ATTS6	Aggregators	Aggregator layer	The adversary modifies the aggregated result before it is forwarded to other components.	Integrity and authenticity	The goal is to tamper with the global aggregation output and directly corrupt the learned model state.
ATTS7	Aggregators	Multi-layer	The adversary modifies the aggregated data of a node that operates as both an aggregator and a participant.	Robustness and trust	The goal is to exploit elevated privileges and coordinated access in order to produce stronger poisoning effects across multiple stages of the DFL process.

TABLE III
MAPPING BETWEEN THE SEVEN ATTACK SCENARIOS AND THE MAIN SECUREDFL DEFENSE COMPONENTS.

Scenario	Attack origin	Target layer	Attack objective	Primary defense	Supporting defense	Security property	Expected effect
ATTS1	Compromised physical resource	Resource layer	Persistent poisoning through fixed execution placement	Coordinator and RL-based mutation	Task reallocation and resource rotation	Availability and robustness	Periodic reassignment reduces continuous exposure of the same jobs to the same compromised resource.
ATTS2	Compromised participant workstation	Participant layer	Injection of forged local updates or control messages	Proxy, token control, and second-device authorization	Participant authentication and trusted approval path	Integrity and authenticity	Forged participant-side transmissions are rejected when the required authorization chain is not satisfied.
ATTS3	Compromised participant and aggregator	Participant and aggregator layers	Privilege abuse for poisoned update acceptance	Second-device path and orchestrator-side checks	Cross-role validation and trust enforcement	Integrity and robustness	A malicious participant-aggregator cannot unilaterally validate poisoned updates without passing the auxiliary trust path.
ATTS4	Compromised communication channel	Communication layer	Message tampering, replay, or unauthorized modification	End-to-end encryption and digital signatures	Protected transmission through communication proxy	Confidentiality and integrity	Message content, authenticity, and integrity remain protected against in-transit manipulation.
ATTS5	Aggregator omits selected participant updates	Aggregator layer	Biased aggregation through selective exclusion	Trust voting and orchestrator notification	Participant-side rejection and trust-score reduction	Integrity and fairness	Repeated selective omission lowers the aggregator trust score and eventually triggers rejection of its behavior.
ATTS6	Aggregator modifies the aggregated output	Aggregator layer	Tampering with the global aggregation result	Protected aggregation and verification	Cryptographic checks and validation logic	Integrity and authenticity	The aggregator cannot arbitrarily alter the protected aggregate without violating the corresponding verification checks.
ATTS7	Privileged participant and aggregator compromise	Multi-layer	Coordinated poisoning with elevated privileges	Layered participant-side and aggregator-side defenses	Authentication, trust management, and protected aggregation	Robustness and defense-in-depth	This stronger adversary is mitigated only through the joint action of multiple defensive layers rather than a single mechanism.

We consider three representative adaptive behaviors. First, the adversary may adjust the poisoning magnitude across rounds to keep malicious updates disruptive while avoiding excessive deviation from benign updates. Second, the adversary may apply delayed or on–off activation, where malicious clients initially behave benignly and activate poisoning after the model becomes more stable. Third, the adversary may conduct multi-surface poisoning by combining model-update manipulation with unlearning-related malicious behavior. These adaptive variants reflect stronger and more realistic poisoning conditions in distributed learning environments.

Let t denote the communication round and let Δ_k^t be the update submitted by client k. For an adaptive malicious client $k \in \mathcal{A}$, the poisoned update is defined as

$$\Delta_{k,\mathrm{adv}}^t = \Delta_k^t + \lambda_t \cdot \mathbf{p}_k^t, \tag{1}$$

where $\mathbf{p}_k^t$ denotes the poisoning direction and λ_t is a round-dependent attack-intensity factor. Unlike static poisoning with a fixed λ, the adaptive setting allows λ_t to change according to the attack schedule and the observed stability of the learning process. This formulation captures time-varying adversarial behavior while preserving the general structure of the DFL training process.

VI. SYSTEM SECURITY INTERACTION OVERVIEW

To clarify SecureDFL's security mechanisms, we present the interaction diagram in Fig. 4, highlighting key secure message flows (token authorization, encrypted transmission, and homomorphic aggregation) that ensure confidentiality, integrity, and privacy across participants, aggregators, and the coordinator.

The interaction diagram provides a high-level view of how the system operates to prevent unauthorized access and manipulation of data. Token authorization ensures that only authenticated participants can send updates, while secure transmission protocols (such as RSA and Paillier encryption) protect the data during communication. Homomorphic aggregation allows the global model to be updated without revealing sensitive information, maintaining both security and privacy.

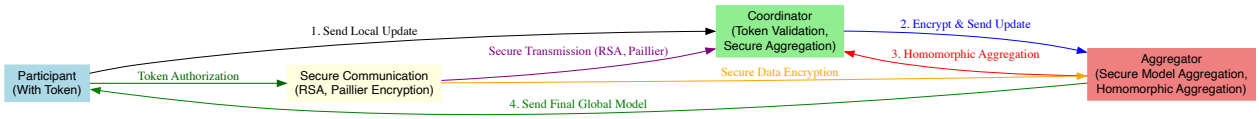


Fig. 4. Interaction diagram illustrating the core security message flows in SecureDFL, including token authorization, secure transmission, and homomorphic aggregation.

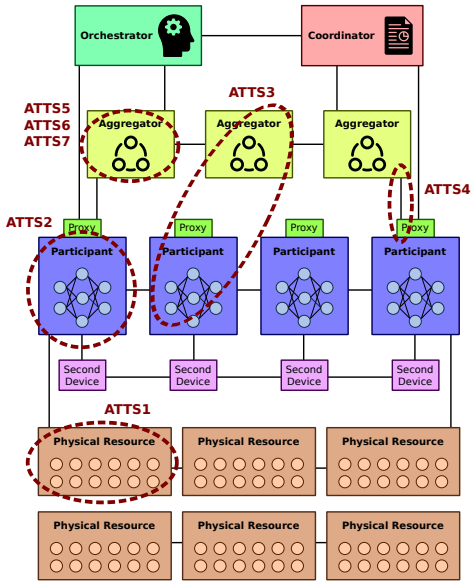


Fig. 5. The components of SecureDFL and their relations to the attack scenarios from Section V-A.

VII. SecureDFL Architecture

In this section, we provide details of our framework, which is secure against the seven attack scenarios mentioned in subsection V-A. Figure 5 presents an overview of SecureDFL's architecture and the components that are affected from each scenario. Our secure framework is composed of seven main components: (1) Orchestrator, (2) Aggregators, (3) Participants, (4) Second Devices, (5) Communication Proxy, (6) Physical Resources, and (7) Coordinator. Participants train local models on distributed resources selected by a coordinator. After training, updates are sent via a communication proxy that transforms and forwards them to aggregators. Participants may also use separate secondary devices, while a central orchestrator manages overall coordination.

A. Coordinator

The coordinator selects the physical resources used to train each participant's local model and applies an MTD strategy against ATTS1 by periodically reallocating tasks across resources. This mutation-based scheduling invalidates adversarial knowledge of resource assignments.

The decision to assign tasks to the physical resources must consider the following limitations:

- If a task requires a capacity of c, it cannot be assigned to a resource with a capacity of less than c.
- If resource r is compromised, it stays compromised until it is recovered while no active task is running on it.
- If resource r is compromised by the adversary, all the tasks that are assigned to r, will be affected.

Considering these limitations, the coordinator dynamically decides the assignment of tasks to the physical resources.

The coordinator utilizes a reinforcement learning model to make the decisions of assigning tasks to the physical resources. Before diving into the learning process, let us model the participants, their tasks, and the physical resources. Assume that we have P participants and the ordered list of them is shown as $\mathcal{P} = \{\mathcal{P}^1, \ldots, \mathcal{P}^P\}$, where $\mathcal{P}^i$ is the set of tasks for training the model of the i^{th} participant. We can show $\mathcal{P}^i$ as $\{\mathcal{P}^i_1, \ldots, \mathcal{P}^i_{P_i}\}$, where P_i is the number of tasks belonging to the i^{th} participant and $\mathcal{P}^i_j$ is its j^{th} task required capacity. On the other hand, assume that there are R physical resources and they are shown as $\mathcal{R} = \{\mathcal{R}^1, \ldots, \mathcal{R}^R\}$, where $\mathcal{R}^i$ indicates the i^{th} physical resource. Each of these resources are shown as $\mathcal{R}^i = \{\mathcal{R}^i_c, \mathcal{R}^i_t\}$, where $\mathcal{R}^i_c$ is the total capacity of the i^{th} resource and $\mathcal{R}^i_t$ is the set of tasks that are assigned to it. We show $\mathcal{R}^i_t$ as $\{\mathcal{R}^i_{t_1}, \ldots, \mathcal{R}^i_{t_{R_i}}\}$, where R_i is the total number of tasks assigned to the i^{th} resource and $\mathcal{R}^i_{t_j}$ is the j^{th} task, and it is shown as $\mathcal{R}^i_{t_j} = (\mathcal{R}^i_{\alpha_j}, \mathcal{R}^i_{\beta_j})$. In this form, $\mathcal{R}^i_{\alpha_j}$ is the participant possessing the j^{th} task of the i^{th} resource and $\mathcal{R}^i_{\beta_j}$ is the order of that tasks among its tasks. For example, if a task for a resource is shown as $(2, 5)$, it indicates the fifth task of the second participant.

In each learning step, e.g., step (s_α, s_β), the agent specifies a number that shows the physical resource identifier for hosting the s^{th}_β task of the s^{th}_α participant. The environment that is presented to the reinforcement learning agent is composed of three features of each physical resource, as follows:

- $\boldsymbol{F^i_1}$: The remained capacity of the i^{th} physical resource.
- $\boldsymbol{F^i_2}$: The number of distinct participants that have tasks running on the i^{th} physical resource.
- $\boldsymbol{F^i_3}$: The time intervals from which the i^{th} physical resource has not recovered.

The reward function for this agent is also calculated based on Equation 2, where (s_α, s_β) is the step and a is the action done by the agent indicating the resource to be the host of the current task. For reproducibility, the RL mutation problem can be summarized as follows. The state at each decision step is the concatenation of the three resource-wise feature vectors F_1, F_2, and F_3 over all R resources. The action space is discrete and consists of the candidate resource identifiers available for the current task. Training is episodic at the coordinator, following Algorithm 1, and the policy is updated as assignments and recovery events evolve across mutation intervals. The empirical justification for RL in this paper is based on ablation, convergence behavior, and system-level security rationale; a full benchmark against simple non-learning baselines such as random mutation, round-robin reassignment, or greedy capacity-only mapping remains future work rather than an already-completed experiment.

$$Re((F^1_1, F^2_1, \ldots, F^{R-1}_3, F^R_3), (s_\alpha, s_\beta), a) = -F^a_3 + \begin{cases} -10, & \text{If } (s_\alpha \neq P \text{ or } s_\beta \neq P_P) \text{ and } F^a_1 < \mathcal{P}^{s_\alpha}_{s_\beta} \\ +1, & \text{If } (s_\alpha \neq P \text{ or } s_\beta \neq P_P) \text{ and } F^a_1 \geqslant \mathcal{P}^{s_\alpha}_{s_\beta} \\ -10 - \max\limits_{1 \leqslant i \leqslant R} F^i_2, & \text{If } (s_\alpha = P \text{ and } s_\beta = P_P) \text{ and } F^a_1 < \mathcal{P}^{s_\alpha}_{s_\beta} \\ +1 - \max\limits_{1 \leqslant i \leqslant R} F^i_2, & \text{If } (s_\alpha = P \text{ and } s_\beta = P_P) \text{ and } F^a_1 \geqslant \mathcal{P}^{s_\alpha}_{s_\beta} \end{cases} \quad (2)$$

The complete process done by the coordinator is presented in Algorithm 1. The first loop in this algorithm (line 2) is dedicated to iterating on different episodes that the agent must pass to be trained to the threshold we want. The next loop (line 4) iterates over the mutation intervals throughout the lifetime of a network. In these two loops, the agent is trained. First, we iterate over different resources and check if they have no assigned tasks; they are recovered (lines 5 to 7). The related list that stores the last recovered time of each resource is updated in this loop (line 7). Then, we must assign all the participants' tasks to the available resources. Hence, we have a nested loop over all participants (line 8) and their overall tasks (line 9), and we investigate each task. To generate the environment features (i.e., F_1, F_2, and F_3), we have a loop on all resources (line 12). In this loop, we have to sum up their occupied capacity and also calculate the number of distinct participants sharing

Algorithm 1 The procedure performed by the coordinator in the proposed secure DFL framework (SecureDFL)

Require: $\mathcal{P}$ (the list of requesting participants)
Require: $\mathcal{R}$ (the list of physical resources)
Require: $episodes$ (the number of training episodes)
Require: $mutations$ (the number of mutation intervals)
Require: $duration$ (the mutation interval duration)
1: $mod \leftarrow$ initialize a reinforcement learning model
2: **for** $1 \leqslant e \leqslant episodes$ **do**
3: $rec \leftarrow$ a list of R zeros
4: **for** $1 \leqslant m \leqslant mutations$ **do**
5: **for** $1 \leqslant r \leqslant R$ **do**
6: **if** $len(\mathcal{R}_t^r) = 0$ **then**
7: $rec[r] \leftarrow m$
8: **for** $1 \leqslant i \leqslant P$ **do**
9: **for** $1 \leqslant j \leqslant P_i$ **do**
10: $F_1, F_2, F_3 \leftarrow$ three empty lists
11: $max \leftarrow 0$
12: **for** $1 \leqslant r \leqslant R$ **do**
13: $c \leftarrow 0$
14: $d \leftarrow$ an empty list
15: **for each** $(x, y) \in \mathcal{R}_t^r$ **do**
16: $c \leftarrow c + P_y^x$
17: **if** $x \notin d$ **then**
18: $d \leftarrow d + \{x\}$
19: $F_1 \leftarrow F_1 + \{R_c^r - c\}$
20: $F_2 \leftarrow F_2 + \{len(d)\}$
21: $F_3 \leftarrow F_3 + \{m - rec[r]\}$
22: **if** $max < len(d)$ **then**
23: $max \leftarrow len(d)$
24: $s \leftarrow \{F_1, F_2, F_3\}$
25: $a \leftarrow$ the optimal action of mod in s
26: **if** $F_1^a \geqslant P_j^i$ **then**
27: $\mathcal{R}_t^a \leftarrow \mathcal{R}_t^a + \{(i, j)\}$
28: **if** $i = P$ **and** $j = P_i$ **then**
29: **if** $F_1^a < P_j^i$ **then**
30: $rew \leftarrow -F_3[a] - 10 - max$
31: **else**
32: $rew \leftarrow -F_3[a] + 1 - max$
33: **else**
34: **if** $F_1^a < P_j^i$ **then**
35: $rew \leftarrow -F_3[a] - 10$
36: **else**
37: $rew \leftarrow -F_3[a] + 1$
38: Update mod based on s, a, and rew
39: wait $duration$ seconds
40: **for** $1 \leqslant r \leqslant R$ **do**
41: $\mathcal{R}_t^r \leftarrow$ an empty list

that resource. Therefore, we have another loop that iterates over all tasks assigned to that resource (line 15). The required resources of that task are summed up to the previous ones (line 16), and moreover, if the task is for a participant who has not been counted before, we count it for the first time (lines 17 and 18). Now that the calculations are done, we can fill in the list of features (lines 19 to 21). There is also another point. In the reward function, we have to know the maximum number of Participants sharing the same resource. For this regard, we dedicate a variable (max) and update it (lines 22 and 23). When all the resources are checked, and the list of features is completely built, we ask the reinforcement learning model to suggest the optimal action that can be done in this state (line 25). This action indicates the resource that will host the current task. If the required resources of the current task are lower than or equal to the remaining capacity of the chosen resource, this task is added to the list of tasks of that resource (line 26 and line 27). Now, it is time to calculate the reward. If we are mapping the last task of the last participant (line 28), we have to follow the two last cases of Equation 2. In other words, if the chosen resource does not have enough capacity to handle the current task (line 29), the reward is -10 minus the maximum number of participants sharing a resource minus the last recovered time of the chosen resource (line 30). Otherwise, a +1 is replaced with -10 (line 32). On the other hand, if we have not yet completed the last task, we will not consider max in the calculations based on the first two cases of Equation 2 (lines 33-37). Now that we have calculated the reward value, we can update the model to learn the action and its related reward in that state (line 38). It is worth noting that when a single step of mutation is performed, the coordinator waits for the duration of the interval and then continues the process (line 39). Finally, when a single episode is done, the assignments are reset to perform the new assignments (lines 40 and 41).

B. Communication Proxy

The communication proxy is a logical function in SecureDFL, not necessarily a single irreducible process. In the evaluated prototype, one proxy service instance is deployed, which is an implementation limitation. However, because payload transformation, signature verification, and encryption handling are mostly stateless once public-key and token metadata are available, the proxy can be horizontally scaled in production. Multiple proxy instances can be replicated behind a Kubernetes service or load balancer, with token-state lookups handled through a shared low-latency store. This distributed proxy design is an architectural extension beyond the current quantitative evaluation.

The communication proxy protects data exchange against ATTS2, ATTS3, and ATTS4 by transforming outgoing and incoming messages through token validation, encryption, and signature-based protection. It also supports mitigation of ATTS6 and ATTS7 by helping preserve message integrity and preventing unauthorized modification of aggregation-related data. Channel compromise is addressed through encrypted transmission, participant-side poisoning is constrained through token-controlled authorization, and aggregator-side manipulation is limited through protected aggregation based on homomorphic cryptography.

This section presents the main steps of SecureDFL's communications, describes our proposed communication protocol between the framework components, and discusses how it makes the mentioned threats ineffective.

The main phases of the communications in SecureDFL are as follows. It is worth noting that two cryptographic algorithms are employed: RSA and Paillier. The RSA algorithm is used to secure connections, and the Paillier algorithm is used to secure the model against compromised aggregators.

Exchanging phase: In this phase, all involved components are signed up to the framework and share their keys with each other. All the participants join the framework by signing up to the orchestrator platform, and each of them specifies the second device they possess. Moreover, the framework administrator also specifies the other components, such as the aggregators. In this phase, all the components specify their RSA public key once they are signed up.

1) The orchestrator generates a pair of RSA keys and shares it with everyone through certified authorities.
2) Each aggregator generates its own pair of RSA keys and sends the public key, as well as its identifier and the message type, which is one here, to the orchestrator.
3) Each participant generates its own pair of RSA keys and sends the public key, as well as its identifier and the message type, which is two here, to the orchestrator.
4) Each second device generates its own pair of RSA keys and sends its public key, as well as its identifier and the message type, which is three here, to the orchestrator.
5) After receiving all the components' public keys, the orchestrator builds up a list of all public keys together with their identifiers.
6) The orchestrator shares the list of public keys together with the message type, which is four here, with all the aggregators, while it is encrypted with their public key.
7) The orchestrator shares the list of public keys together with the message type, which is four here, with all the participants, while it is encrypted with their public key.
8) The orchestrator shares the list of public keys together with the message type, which is four here, with all the second devices, while it is encrypted with their public key.
9) The orchestrator generates a Paillier key pair for protected aggregation, keeps the raw decryption key under authorized control, signs the corresponding public-key metadata together with participant-specific controlled-decryption authorization material by using its RSA private key, puts this material in a list with the message type, which is five here, encrypts it with the RSA public key of participants, and sends the result to them.

Pairing phase: In this phase, which is a kind of transmission initiation phase, a participant notifies the start of sending information of the trained model. This information may be the first data sent for that model or the updated one. Once this transmission is initiated, the orchestrator sends a token based on the current transmission timestamp to the second device associated with that participant.

1) The participant that wants to send data on its trained model signs the current timestamp, puts it together with its identifier and the message type, which is six here, and sends the result to the orchestrator. Hence, the orchestrator will be notified that one of the participants is ready to transfer its data.
2) Once the orchestrator receives the notification message of a participant, a unique random token based on the participant identifier and the timestamp is generated.
3) The generated token is sent to all the aggregators, except the aggregator, which is the participant, too, together with the original timestamp and the requester participant's identifier. This message is signed by the orchestrator's private key, put together with the message type, which is seven here, and encrypted with the public key of the destination aggregator.
4) The same process of sharing the token with the aggregators is also done for sharing it with the second device of the related participant. This time, the message is encrypted by the device's public key.
5) Once a second device receives a token, if it is sure that the model is trained by its permission, the token is sent to the related participant. Otherwise, nothing is sent, and the token is not shared with the participant at all. In the case that the token is shared, the token and the original timestamp are signed by the device's RSA private key. Then it is listed with the device's identifier and the message type, which is eight here, and encrypted with the participant's RSA public key.

Transmission phase: In this phase, participants submit their protected model updates to the selected aggregator(s). The aggregator(s) verify the submitted tokens, collect valid updates, compute the protected aggregate, and return the aggregation result to the participants.

1) Each participant first encrypts its model update using the Paillier public key for protected aggregation. The encrypted update is combined with the token and timestamp, signed with the participant's RSA private key, and then attached to the participant identifier and message type 9. The complete message is encrypted using the aggregator's public key.
2) After receiving a model-update message, the aggregator verifies whether the token is valid for the corresponding participant–device pair. Messages with invalid tokens are discarded.
3) After receiving valid updates from all devices, or from those responding within the predefined time interval, the aggregator performs the required additive aggregation operation, such as summation for averaging. The aggregated result is signed with the aggregator's private key, combined with the aggregator identifier and message type 10, and encrypted using the destination participant's public key.
4) At the implementation level, Paillier is used only for additive homomorphic operations on fixed-point encoded model values. Thus, the aggregator computes ciphertext-domain sums of protected updates. The normalization required for averaging is not performed as native floating-point division under Paillier; instead, the summed update is decrypted by the authorized side and normalized in plaintext, or represented through fixed-point reciprocal encoding when quantization is applied. This distinction is necessary because Paillier supports addition and scalar multiplication, but not general floating-point division.
5) After receiving the aggregated result, the participant uses the controlled-decryption authorization material distributed during the exchange phase to recover the protected aggregate through the authorized decryption procedure. The participant then evaluates the aggregator and submits a trust vote, using message type 0.
6) The orchestrator collects the participants' votes, computes the average trust score of each aggregator, and sends the updated scores back to the participants using message type 0.

TABLE IV
THE COMPLETE DESCRIPTION OF THE PROPOSED PROTOCOL FOR PROTECTING THE DFL FRAMEWORK (SECUREDFL)

Phase	Step	From	To	Message
Exchange	1-2	A^*	O	$[1\|a\|A'] : O'$
	1-3	P^*	O	$[2\|p\|P'] : O'$
	1-4	D^*	O	$[3\|d\|D'] : O'$
	1-6	O	A^*	$[4\|a^*\|A'^*\|p^*\|P'^*\|d^*\|D'^*] : A'$
	1-7	O	P^*	$[4\|a^*\|A'^*\|p^*\|P'^*\|d^*\|D'^*] : P'$
	1-8	O	D^*	$[4\|a^*\|A'^*\|p^*\|P'^*\|d^*\|D'^*] : D'$
	1-9	O	P^*	$[5\|[\dot{O}\|\kappa_O] : O''] : P'$
Pairing	2-1	P	O	$[6\|p\|[time] : P''] : O'$
	2-3	O	A^*	$[7\|[p\|time\|token] : O''] : A'$
	2-4	O	D	$[7\|[p\|time\|token] : O''] : D'$
	2-5	D	P	$[8\|d\|[time\|token] : D''] : P'$
Transmission	3-1	P	A^*	$[9\|p\|[time\|token\|[data] : \dot{O}] : P''] : A'$
	3-3	A^*	P^*	$[10\|a\|[+[data]^* : \dot{O}] : A''] : P'$
	3-4	P^*	O	$[0\|p\|[a\|vote] : P''] : O'$
	3-5	O	P^*	$[0\|[a\|score] : O''] : P'$

The pairing and transmission phases may be repeated for new training model updates.

Based on the communication workflow, this subsection defines the notation used in the proposed protocol. The symbols O, A, P, and D denote the orchestrator, aggregator, participant, and second device, respectively. An asterisk indicates the set of all entities of a given type; for example, A^* denotes all aggregators, whereas A or $\bar{A}$ denotes a specific aggregator. Lowercase symbols are used as entity identifiers; for instance, a represents the identifier of aggregator A.

The notation $x|y$ denotes concatenation or list formation, while $+x$ represents arithmetic operations over x. The expression $[x] : y$ indicates that data x is encrypted or decrypted using key y. Since the protocol employs both RSA and Paillier cryptography, x' and x'' denote the RSA public and private keys of entity x, respectively, while $\dot{x}$ and $\ddot{x}$ denote its Paillier public and private keys. In addition, κ_x represents the controlled-decryption authorization material issued by entity x for the protected aggregation workflow. The complete protocol is presented in Table IV.

The security analysis of this protocol, regarding the threat model presented in subsection V-A, is as follows: **Resilience against ATTS2**: During the pairing phase, a token is given to the second device, and the second devices only share it with the legal participant. This means if a participant is compromised, they will never receive the token. This avoids poisoning attacks from malicious participants.

Resilience against ATTS3: If a participant is compromised, while it is an aggregator, too, the token is not again shared with it, as we mentioned in the process of the protocol.

Resilience against ATTS4: The proposed protocol is safe against poisoning attacks launched by compromised channels. Because the framework components sign all the transferred messages during the pairing phase and the transmission phase. Therefore, as long as the private key of the main components is not leaked, the protocol is safe against man-in-the-middle attacks.

Resilience against ATTS6: The proposed protocol treats malicious aggregator-side modification as a detectable integrity violation rather than claiming that Paillier alone prevents such modification. Participant updates are bound to tokens and RSA signatures, while returned aggregates are also signed by the aggregator. Hence, unauthorized modification can be exposed through participant-side verification and reflected in the orchestrator's trust-update process, after which the malicious aggregator can be isolated.

Resilience against ATTS7: If an aggregator is also a privileged partic-

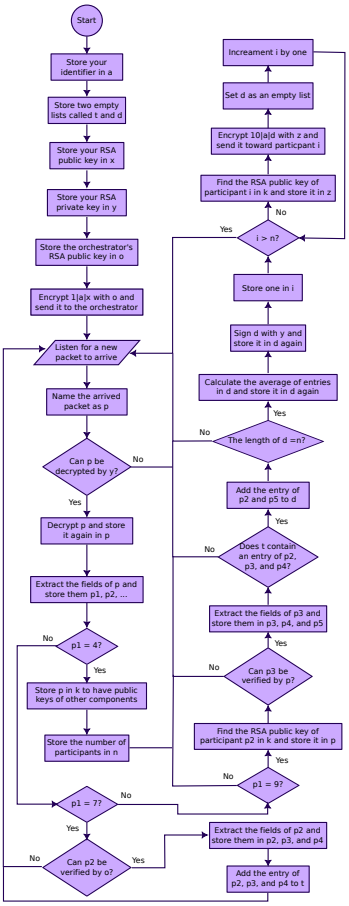


Fig. 6. The procedure performed by the aggregators based on the proposed communication protocol

ipant, the risk is reduced by the pairing-phase token control, authenticated messaging, and trust-based isolation logic discussed above. However, the paper does not claim a complete guarantee against all colluding privileged-compromise cases, which remain an explicit limitation of the current prototype and threat coverage.

These protocol-level guarantees are conditional on the confidentiality of the relevant private keys and on the integrity of the credential-validation infrastructure. Accordingly, the paper does not claim protection against private-key leakage, side-channel secret extraction, or compromise of the certificate or trust-anchor infrastructure itself. When auxiliary second-device verification is used, its protective value also weakens if that device is compromised, unavailable, or controlled by the same adversary; in such deployments, alternative authentication paths are required. The flowcharts illustrating the procedures executed in the aggregators, participants, and secondary devices are presented in Figure 6, Figure 7, and Figure 8, respectively. It is worth noting that, for the sake of simplicity in presenting the flowchart, we have assumed in the aggregator's flowchart that the aggregators collect all the participants' data before initiating the aggregation phase. However, in practice, one can assume that the aggregation phase is performed periodically, so that if a participant intentionally fails to send its data, the entire process is not disrupted.

C. Orchestrator

In addition to orchestrating the proposed communication protocol, the orchestrator is also responsible for organizing voting sessions, during which the participants can give trust scores to the aggregators. If an aggregator's trust level is below a specific threshold, the orchestrator notifies the participants to ignore its updates. This voting process mitigates ATTS5.

When a participant receives the aggregated data from an aggregator based on its own trained model and the history of that aggregator, a trust score, which is an integer between 1 and 9, is assigned. Higher trust scores mean higher trust. The trust score is combined with the related aggregator's identifier, signed by the source participant's private key, along with its identifier, the message type (which is zero here), and finally encrypted by the orchestrator's public key.

Once the orchestrator receives the votes, it calculates the average trust score of each aggregator, combines it with the aggregator's identifier, signs the result with its private key, adds the message type (which is zero here), and encrypts the entire message with the public key of each participant. To improve sensitivity to sudden malicious behavior, the score update rule discounts historical behavior through a forgetting factor and allows a current-round alarm to trigger immediate isolation. Let $\bar{v}(i,j) = \frac{1}{P}\sum_{k=1}^{P} v_k$ denote the average vote assigned to aggregator i at interval j, where $v_k \in \{1,\ldots,9\}$ is the vote from participant k. The updated trust score is defined by

$$sc(i,j) = \begin{cases} \lambda\, sc(i,j-1) + (1-\lambda)\,\bar{v}(i,j), & \bar{v}(i,j) \geqslant \tau_{\mathrm{crit}}, \\ \bar{v}(i,j) - \rho, & \bar{v}(i,j) < \tau_{\mathrm{crit}}, \end{cases} \tag{3}$$

where $0 \leqslant \lambda < 1$ is a forgetting factor, τ_{crit} is the critical trust threshold, and $\rho > 0$ is an immediate penalty applied when the current round indicates abrupt malicious behavior. Hence, a previously well-behaved aggregator does not remain protected by a large historical score once its current-round votes deteriorate sharply. In deployment terms, the orchestrator can quarantine an aggregator as soon as $\bar{v}(i,j) < \tau_{\mathrm{crit}}$, regardless of the accumulated historical score.

The operations an orchestrator performs in our framework are presented in Algorithm 2. It is worth noting that, for the sake of simplicity in illustrating the algorithm without multi-threading procedures, the procedure described for the orchestrator assumes that the orchestrator collects all the votes and then sends back the scores of the aggregators. However, in practice, one can assume that scores are sent periodically, so that if a participant intentionally fails to send their vote, the entire process is not disrupted. For the same reason, Algorithm 2 should be read as a simplified control-flow description of vote collection. The operational score update follows Eq. (3), including the forgetting factor and the immediate alarm threshold, rather than a pure historical average.

In the process described by Algorithm 2, a loop is used to receive incoming packets and to dispatch them according to the protocol message type. Registration packets from aggregators, participants, and second devices populate the corresponding public-key tables, whereas token-request packets are authenticated and converted into signed token messages that are redistributed to the relevant components. For vote packets, the orchestrator first verifies the participant signature and then stores the vote under the addressed aggregator. Once all participant votes for an aggregator are available for the current interval, their mean $\bar{v}$ is computed. The score update is then applied exactly as in Eq. (3): if the current-round mean falls below the critical threshold, an immediate penalty is applied; otherwise, the new score is obtained from the forgetting-factor combination of the previous score and the current mean vote. The updated score is signed by the orchestrator and broadcast to participants so that malicious aggregators can be rapidly isolated. Finally, when all public keys have been collected, the orchestrator executes the key-sharing procedure in Algorithm 3.

In the algorithm for the key-sharing process (Algorithm 3), we first need to check if all components have shared their public keys (Line 1). If this condition is satisfied, the orchestrator builds a message containing the message type, which is four (Line 2). Then, a for loop is built on all the aggregators (Line 3). Through this loop, the aggregator's identifier and its key are added to the message (Line 4). The same loops with the same processes are built for the participants (Lines 5 and 6) and the second devices (Lines 7 and 8). Now that all the information is added to the message, the orchestrator loops through all the aggregators to send that message to them (Line 9). In this loop, the message is first encrypted with the public key of the aggregator (Line 10) and then sent to it (Line 11). The same process is performed for both participants and devices through Lines 12 to 17.

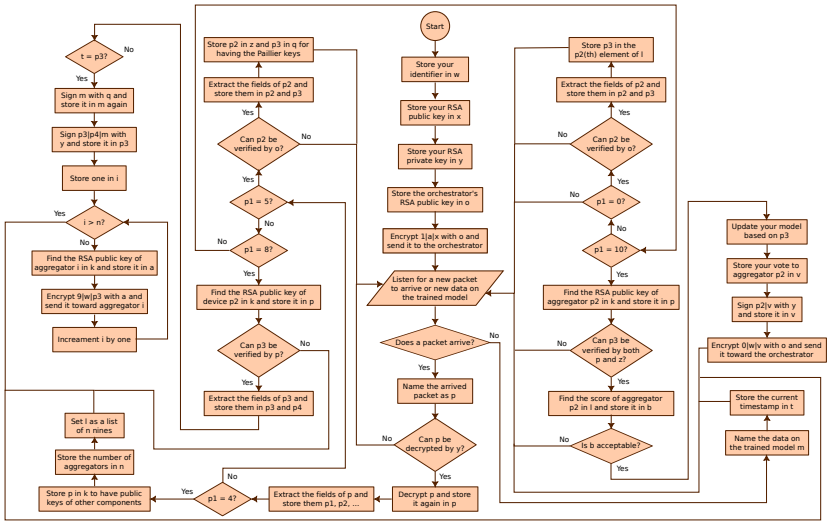


Fig. 7. The procedure performed by the participants based on the proposed communication protocol

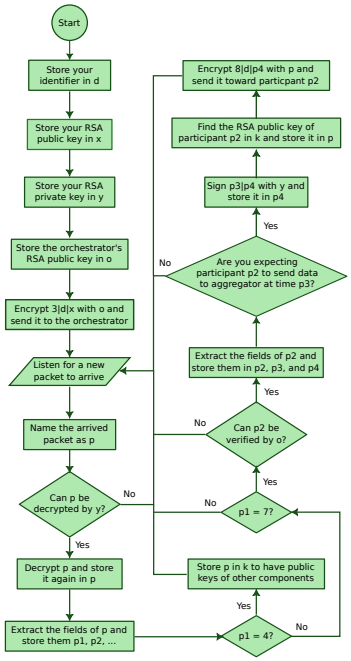


Fig. 8. The procedure performed by the second device based on the proposed communication protocol

D. Deployment and Operational Details of SecureDFL Components

The SecureDFL framework comprises seven main components, each designed for a specific operational role within the federated learning environment. Here, we detail the deployment and practical function of each component, as well as their integration within the overall system (see Figure 5 for architecture overview):

Orchestrator: This component is centrally deployed on the master node (or control plane) of the cluster infrastructure. The Orchestrator manages global key exchanges, token generation, trust score calculation, and orchestration of security-critical operations. It is implemented as a Python-based FastAPI service running in a dedicated Docker container. All participating nodes authenticate with the Orchestrator to ensure secure bootstrapping and coordination.

Coordinator: Also deployed on the master node, the Coordinator is responsible for dynamic resource allocation and mutation using reinforcement learning. It communicates with both the Orchestrator and the physical resource layer (Kubernetes API) to assign, reassign, and mutate compute resources (i.e., pods or VMs) for distributed training tasks. In the testbed, it is a separate FastAPI microservice container.

MTD Strategy: In SecureDFL, the coordinator is responsible for implementing an MTD strategy as a core security measure [29]. MTD is defined as a proactive defense mechanism that periodically and unpredictably mutates the assignment of physical compute resources (such as containers, pods, or VMs) to logical federated learning tasks [29, 30, 31]. By dynamically changing resource-task mappings and communication paths at regular intervals, the system significantly reduces the attack surface and the effectiveness of persistent or targeted adversaries. This ongoing mutation makes it extremely difficult for an attacker (such as those characterized in ATTS1) to exploit the same resource or communication channel across multiple training rounds, as their target environment is constantly shifting. The mutation schedule and mapping logic are determined by the coordinator using a reinforcement learning agent, which balances system performance and security.

Aggregators: Aggregators are instantiated on selected worker nodes or cloud VMs. They are responsible for collecting, validating (using token-based authentication), and aggregating model updates from participants within their respective regions or groups. Each aggregator is deployed as an independent Docker container and may be assigned dynamically for load balancing or security rotation.

Participants: Each participant (e.g., a data-owning organization such as a hospital) is represented by a dedicated service or container on a worker node. These entities execute local model training using the assigned resources, initiate communication with aggregators, and participate in voting on aggregator trustworthiness. In deployment, each participant can run as a Kubernetes-managed container, isolated for security.

Second Devices: For each participant, a second device is registered as a distinct container or physical node, separate from the main participant workstation. Its primary purpose is to enhance security by acting as an independent endpoint for token validation and multi-factor authentication during model transmission and update phases. These are typically lightweight microservices or VMs.

Communication Proxy: All network communication between participants, aggregators, the orchestrator, and coordinator is routed through the Communication Proxy. This component is implemented as a secure microservice (containerized) that enforces encryption/decryption (RSA, Paillier), verifies tokens, and implements protocol logic for resisting replay and man-in-the-middle attacks.

Physical Resources: The compute layer consists of resource units (pods or VMs) deployed on the worker nodes. The Coordinator dynamically allocates these based on workload requirements and security policies. Each resource executes model training jobs, and the set of assignments is regularly mutated to reduce the risk of persistent com-

Algorithm 2 The procedure performed by the orchestrator in the proposed secure DFL framework (SecureDFL)

Require: x (the orchestrator's RSA public key)
Require: y (the orchestrator's RSA private key)
Require: A (the total number of subscribed aggregators)
Require: P (the total number of subscribed participants)
Require: $\lambda, \tau_{\text{crit}}, \rho$ (trust-update parameters from Eq. (3))
$a, w, d \leftarrow$ three empty labeled lists
$tok, vot \leftarrow$ two empty labeled lists
$sco, int \leftarrow$ two empty labeled lists
for each arrived packet as p **do**
  if p can be decrypted by y **then**
    $p \leftarrow$ extraction of the p fields
    if $p[1] = 1$ **then**
      insert $p[3]$ to a with labels of $p[2]$
      $int[a] \leftarrow 0$
      $sco[a], vot[a] \leftarrow$ two empty lists
    else if $p[1] = 2$ **then**
      insert $p[3]$ to w with labels of $p[2]$
    else if $p[1] = 3$ **then**
      insert $p[3]$ to d with labels of $p[2]$
    else if $p[1] = 6$ **then**
      if $p[3]$ can be verified by $w[p[2]]$ **then**
        $t \leftarrow$ a random big number
        insert t to tok with label of $p[2], p[3]$
        $p \leftarrow$ sign of $p[2]|p[3]|t$ with y
        for $1 \leqslant i \leqslant A$ **do**
          if $a[i] \in w$ **then**
            continue
          $o \leftarrow$ encryption of $7|p$ with $a[i]$
          send o toward aggregator i
        $o \leftarrow$ encryption of $7|p$ with $d[p[2]]$
        send o toward device $p[2]$
    else if $p[1] = 0$ **then**
      if $p[3]$ can be verified by $w[p[2]]$ **then**
        $p[3], p[4] \leftarrow$ extraction of $p[3]$ fields
        insert $p[4]$ to vot with label of $p[3]$
        for $1 \leqslant i \leqslant A$ **do**
          if size$(vot[i]) \neq P$ **then**
            continue
          $s \leftarrow 0$
          for each $v \in vot[i]$ **do**
            $s \leftarrow s + v$
          $\bar{v} \leftarrow s/P$
          if $\bar{v} < \tau_{\text{crit}}$ **then**
            $s \leftarrow \bar{v} - \rho$
          else
            $s \leftarrow \lambda * sco[i] + (1 - \lambda) * \bar{v}$
          $sco[i] \leftarrow s$
          $int[i] \leftarrow int[i] + 1$
          $vot[i] \leftarrow$ an empty list
          $s \leftarrow$ sign of $i|s$ with y
          for $1 \leqslant j \leqslant P$ **do**
            $o \leftarrow 0|s$
            $k \leftarrow w[j]$
            $o \leftarrow$ encryption of o with k
            send o toward participant i
    if $p[1] = 1$ **or** $p[1] = 2$ **or** $p[1] = 3$ **then**
      perform Algorithm 3(A,P,a,w,d)

promise.

Integration and Workflow: At system startup, the orchestrator and coordinator are launched on the master node. Participants, aggregators, and second devices are instantiated on worker nodes as containerized services. The communication proxy mediates all inter-component traffic. The orchestrator handles key exchange and trust scoring, the coordinator manages resource mutation, and aggregators collect updates for secure global aggregation. This deployment leverages Kubernetes for orchestration and Docker for containerization, ensuring scalability, modularity, and robustness. Practical implementation details and deployment scripts are provided in Section VII. Table V summarizes the deployment locations and main operational roles of each component within SecureDFL.

VIII. Architectural Extension: Threshold-Based Orchestrator/Coordinator

The current prototype deploys the orchestrator and coordinator on the master node, as stated in the testbed section. We therefore do *not* claim that a threshold-distributed orchestrator/coordinator has already been implemented and experimentally validated in the reported results. Instead, the threshold-based design is a forward architectural extension intended to remove the control-plane single point of trust in future versions of the system. In that extension, sensitive actions such as token generation, key management, and mutation-policy approval would require quorum participation across multiple control nodes, with replicated state and auditable logs. This distinction avoids conflating the validated prototype with the proposed distributed design path.

Algorithm 3 The procedure performed by the orchestrator for sharing the keys in the proposed secure DFL framework (SecureDFL)

Require: A (the total number of subscribed aggregators)
Require: P (the total number of subscribed participants)
Require: a (the list of aggregators' RSA public keys)
Require: w (the list of participants' RSA public keys)
Require: d (the list of second devices' RSA public keys)
if size$(a) = A$ **and** size$(w) = P$ **and** size$(d) = P$ **then**
  $o \leftarrow 4$
  for $1 \leqslant i \leqslant A$ **do**
    insert $|i|a[i]$ to the end of o
  for $1 \leqslant i \leqslant P$ **do**
    insert $|i|w[i]$ to the end of o
  for $1 \leqslant i \leqslant P$ **do**
    insert $|i|d[i]$ to the end of o
  for $1 \leqslant i \leqslant A$ **do**
    out encryption of o with $a[i]$
    send out toward aggregator i
  for $1 \leqslant i \leqslant P$ **do**
    out encryption of o with $w[i]$
    send out toward participant i
  for $1 \leqslant i \leqslant P$ **do**
    out encryption of o with $d[i]$
    send out toward device i

TABLE V
Deployment summary and roles of SecureDFL components.

Comp.	Deploy.	Role	Details
Orchestrator	Master (Cont.)	Key mgmt. & trust	Auth service and Docker
Coordinator	Master (Cont.)	Resource assign.	RL, K8s API, and microservice
Aggregator	Worker (Cont./VM)	Aggreg. & valid.	Update checking and load balancing
Participant	Worker (Cont.)	Local train. & voting	Org.-isolated, K8s-managed
Second device	Worker (Cont./VM)	Extra auth.	Token and 2FA endpoint
Comm. proxy	Cluster (Cont.)	Data exchange	Encryption and attack protection
Physical res.	Worker (Pod/VM)	Train. exec.	Rotated compute units

IX. Second-Device Assumption and Alternative Authentication Methods

The framework presented in this paper assumes the use of a second device for authentication to enhance security. While this assumption improves the overall security of the system, we acknowledge that it may not be feasible in certain real-world scenarios, particularly in environments like Internet of Things (IoT) applications, where the availability of a second device is not always guaranteed. In these IoT scenarios, where adding a second device for authentication could be impractical, alternative methods can be employed. For instance, SMS-based authentication and biometric authentication (such as fingerprint recognition or facial recognition) are potential solutions. These methods may provide practical authentication alternatives without requiring additional hardware, but they should be understood as offering different security–usability trade-offs rather than an identical assurance level to a dedicated second device. Such alternatives are particularly useful in environments where a second device may not be readily available, helping the system remain practical for deployment across a wider range of applications.

X. Security Assumptions and Analytical Guarantees

The analysis below should be read as an *assumption-explicit security argument*, not as a full reduction-style formal proof in the cryptographic sense. Its goal is to state clearly what SecureDFL protects under the assumed threat model, which assumptions are critical, and where theorem-level guarantees are not yet provided.

TABLE VI
SECURITY ASSUMPTIONS, SUPPORTED GUARANTEES, AND PROTOTYPE LIMITATIONS OF SECUREDFL.

Assumption or mechanism	ATTS	Supported guarantee	Current limitation
RSA signatures and authenticated key exchange	ATTS4	Protect message authenticity and integrity when cryptographic keys remain secure.	No guarantee is claimed against private-key leakage, side-channel extraction, or certificate infrastructure compromise.
Paillier-protected aggregation	ATTS5/6	Prevents direct exposure of raw participant updates to an honest-but-curious aggregator and supports protected additive aggregation.	Floating-point support is not native; averaging requires fixed-point encoding and/or post-decryption normalization.
Second-device token path	ATTS2/3	Separates authorization from the main participant endpoint and raises the attack barrier under workstation compromise.	Its strength decreases if the second device is compromised or unavailable.
RL-based mutation	ATTS1	Reduces persistent exposure by dynamically changing task-to-resource assignment.	No closed-form convergence or optimality guarantee is claimed under arbitrary adaptive adversaries.
Trust voting and orchestrator enforcement	ATTS5/6	Provides a response mechanism against repeated omission or manipulation by aggregators.	False positives under highly non-IID data and coordinated vote manipulation are not fully quantified.
Single master-node orchestrator/coordinator	Global	Matches the implemented prototype and reported testbed results.	Part of the control-plane trust remains centralized; the distributed threshold version is not yet evaluated.

XI. DETAILED ALGORITHMIC WORKFLOW OF SECUREDFL

In this section, we provide a concise operational summary of the implemented SecureDFL pipeline. The purpose of this workflow is to align the high-level system logic with the actual implementation choices described earlier, especially for RL-based resource mutation, Paillier-protected aggregation, and orchestrator-based trust enforcement.

Algorithm 4 SecureDFL Operational Workflow

Input: Current global model M^{t-1}, participant set P, aggregator set A, resource set R, and coordinator/orchestrator parameters
Initialize cryptographic material, token metadata, and the current RL mutation policy
for each global round $t = 1$ to T **do**
  Coordinator step: update or reuse the current resource assignment for participant tasks using the RL policy and the state features F_1, F_2, and F_3
  for each participant $i \in P$ **do**
    Train the local model on the currently assigned resources and compute the protected local update ΔM_i
    Obtain a valid token, sign the outgoing message, and send the protected update through the communication proxy
  for each active aggregator $a \in A$ **do**
    Verify tokens and message authenticity for the received participant updates
    Compute the ciphertext-domain sum of valid Paillier-protected updates
    Decrypt the summed update on the authorized side and perform normalization in plaintext (or via fixed-point reciprocal encoding when quantized averaging is used)
    Form the aggregated model/result and redistribute it to the corresponding participants
  for each participant receiving an aggregated result **do**
    Evaluate the returned result and send a signed trust vote to the orchestrator
  The orchestrator updates trust scores according to Eq. (3) and immediately quarantines any aggregator whose current-round vote average violates the critical threshold
Output: Final aggregated model after T rounds together with enforced trust decisions and mutation-driven resource assignments

XII. COORDINATOR DESIGN AND SECURITY RATIONALE

In this section, we present the key principles behind the coordinator design in SecureDFL, providing a higher-level overview of its core components and their security rationale. The coordinator plays a crucial role in the learning pipeline by managing mutation-driven resource allocation and by supporting the secure orchestration of distributed training.

A. Reinforcement Learning-Based Dynamic Mutation

The primary advantage of the reinforcement learning (RL)-based dynamic mutation strategy is its adaptability to evolving attack scenarios. Unlike static schemes that rely on fixed resource assignments or predefined paths, our RL-based approach allows the system to dynamically adjust its resources and communication channels in response to observed attack patterns. This adaptability ensures that SecureDFL remains resilient against persistent adversarial threats, making it more effective than static schemes that are vulnerable to evolving attack strategies. The theoretical foundation behind this approach stems from reinforcement learning, where the agent (the system) learns to optimize the allocation of resources to maximize security and minimize the attack success rate.

B. State Space and Reward Function Design

The RL state in SecureDFL is defined by the coordinator based on the resource-allocation environment, rather than aggregator trust history. It consists of three resource-wise features: F_1, the remaining capacity of each resource; F_2, the number of distinct participants sharing that resource; and F_3, the number of mutation intervals since the resource last recovered from compromise. The global state is obtained by concatenating these feature vectors across all resources.

The reward function in Eq. (2) promotes feasible assignments, discourages participant concentration on the same resource, and reduces prolonged exposure to unrecovered resources. Thus, the RL policy is capacity-aware, diversity-aware, and recovery-aware. Its effectiveness is supported empirically through ablation and convergence results, rather than a closed-form optimality proof. A full comparison with simple non-learning mutation heuristics, such as random, round-robin, or greedy capacity-only reassignment, is left for future work.

C. Coordinator Paradigm and Security Logic

In the current evaluated prototype, the coordinator is a control-plane component deployed on the master node and dedicated to resource assignment and mutation scheduling. It does *not* itself perform the cryptographic trust-voting role of the orchestrator or replace the aggregators in the model-aggregation path. Its security contribution is to reduce the persistence of resource-level compromise by periodically remapping distributed training tasks in a way that is sensitive to resource capacity, participant spread, and recovery status. As discussed earlier, a threshold-distributed control-plane realization is a future architectural extension; it is not claimed as part of the experimentally validated prototype reported here.

XIII. EVALUATION

In this section, we present the details of the implemented testbed and the numerical results that evaluate SecureDFL.

A. Experimental Setup

We conducted a comprehensive evaluation of SecureDFL using multiple benchmark datasets and diverse adversarial scenarios to assess security, robustness, and efficiency.

Datasets: Experiments were performed on:

- **MNIST** [32]: 70,000 grayscale images (28×28 pixels, 10 classes). For federated experiments, 10,000 samples were reserved for testing; the remainder was partitioned among 10 clients (each receiving 6,000 samples), in both IID and non-IID fashions (label skew: each client assigned data from 2–3 random classes).
- **Fashion-MNIST** [33]: 70,000 fashion product images (same format as MNIST), partitioned similarly.
- **CIFAR-10** [34]: 60,000 RGB images (32×32, 10 classes); 10,000 for testing and 50,000 for training, distributed among 20 clients

(2,500 samples per client). Data augmentation included random horizontal flip and cropping to 32×32.

All data were normalized to $[0, 1]$; no additional preprocessing was performed for MNIST/Fashion-MNIST.

Model Architectures: For MNIST and Fashion-MNIST, we used a lightweight CNN consisting of two convolutional layers (Conv2D with 32 and 64 filters, kernel size 3×3, ReLU activation), max pooling, flattening, one dense hidden layer (128 units, ReLU activation), and a final softmax output. For CIFAR-10, a deeper CNN was adopted: three convolutional blocks (32/64/128 filters), each with batch normalization and dropout (rate 0.3), followed by two dense layers (256 and 10 units). The optimizer was Adam (learning rate 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$), batch size 64, and local epochs per round set to 5. Training was implemented in PyTorch 1.13 on a workstation with an NVIDIA RTX 3090 GPU, 128GB RAM, and an Intel Xeon Gold CPU. Federated Averaging (FedAvg) served as the baseline aggregation algorithm in our experiments. In each global round, participants performed local training on their respective datasets and transmitted model updates to the aggregator for secure aggregation, as coordinated by SecureDFL. Both independent and identically distributed (IID) and non-IID data partitions were used to evaluate protocol robustness under diverse conditions.

Attack Scenarios. To align the evaluation with the threat model in Section V-A, the experiments are organized around the seven defined attack scenarios rather than an unrelated secondary taxonomy. ATTS1 evaluates persistent attacks on repeatedly selected worker nodes and static resource mappings. ATTS2 and ATTS3 emulate compromised participant workstations, with and without simultaneous aggregator functionality, by injecting manipulated local updates after otherwise standard local training. ATTS4 evaluates compromised communication channels through message tampering, delay, or replay. ATTS5 and ATTS6 emulate malicious aggregators that either omit selected participant contributions or modify the aggregated result before redistribution. ATTS7 represents the stronger participant-aggregator compromise case with privileged access. For dataset-level learning experiments, these scenarios are instantiated using label flipping, update perturbation, selective omission, and aggregator-side manipulation patterns. All attack parameters, including poisoning ratios, attack frequency, perturbation magnitude, and targeted component, were systematically varied, and each configuration was repeated five times with mean and standard deviation reported.

B. Additional Evaluation under Adaptive Attacks

To further assess robustness under stronger adversarial conditions, this subsection evaluates SecureDFL against adaptive poisoning variants that change their behavior over time or across attack surfaces. In addition to the original attack scenarios, three adaptive settings are considered: adaptive model poisoning, delayed/on–off poisoning, and multi-surface adaptive poisoning.

In adaptive model poisoning, malicious clients dynamically vary the poisoning intensity across communication rounds, which tests whether the defense remains stable when the adversary balances attack strength and stealthiness. In delayed/on–off poisoning, malicious clients behave benignly during early rounds and activate the attack after the model partially converges. This setting reflects adversaries that first build benign-looking histories before launching poisoning. In multi-surface adaptive poisoning, malicious behavior targets both model updates and unlearning-related interactions, representing a stronger attack that is not limited to one system surface.

For each adaptive setting, we report test accuracy, attack success rate (ASR), robustness degradation, and average per-round latency. The adaptive schedules are implemented by varying the poisoning magnitude, activation time, and attack surface while keeping the remaining experimental settings consistent with the original evaluation protocol.

Table IX shows that adaptive attacks are more challenging than static single-surface poisoning. The multi-surface setting causes the largest degradation because the adversary manipulates more than one interaction point. Nevertheless, the degradation remains bounded across all adaptive settings, indicating that SecureDFL can maintain robustness beyond fixed single-surface poisoning scenarios.

C. Evaluation Metrics and Results

We evaluated the system's performance using four primary metrics: test accuracy, attack success rate (ASR), training latency, and scalability. Test accuracy reflects the classification accuracy of the global model on a held-out test set, while ASR denotes the percentage of adversarial updates that successfully degraded global accuracy below a defined threshold. Training latency measures the average wall-clock time per global round, including all communication, aggregation, and reinforcement learning (RL) mutation overheads. Scalability was assessed in terms of throughput (samples processed per second) and system performance as the number of participants increased, with simulations conducted up to 100 clients. The main results, summarized in Table VIII, are reported for MNIST under three scenarios: a clean baseline (no attack), a label-flipping poisoning attack (with 20% malicious clients), and the SecureDFL defense. Comparable patterns were observed on Fashion-MNIST and CIFAR-10.

We also make the current empirical boundary explicit. The present evaluation covers the seven defined poisoning scenarios and includes IID and moderate non-IID label-skew settings, but it does not yet provide a comprehensive benchmark for coordinated multi-stage attacks, colluding trust manipulation, or extremely heterogeneous non-IID regimes with calibrated false-positive analysis. These cases remain important extensions for future experimental work.In particular, deviation from the dominant update pattern should not be interpreted automatically as malicious behavior, because benign statistical heterogeneity can itself produce substantial update variation. The trust-aware layer should therefore be read in light of the evaluated data regimes rather than as a universally calibrated detector for all non-IID settings.

Ablation and Robustness. As detailed in Table XII, removal of any individual security component such as the RL-based mutation or secure communication proxy resulted in a higher attack success rate and/or decreased model accuracy. This finding highlights the necessity and effectiveness of SecureDFL's layered security design.

Scalability and Large Model Discussion. The microservice-based architecture of SecureDFL was stress-tested with up to 100 simulated clients, where the latency per global round increased sublinearly (from 12.8 seconds at 10 clients to 34.2 seconds at 100), while test accuracy remained above 93% even under attack conditions. Due to computational resource constraints, direct experiments on very large models (e.g., LLaMA [35]) were not conducted; however, recent studies on federated large language models [36, 37] indicate that RL-driven orchestration and mutation mechanisms such as those employed in SecureDFL are transferable and effective for scaling to large models, consistently yielding enhanced attack resilience with minimal additional latency. These findings support the broader applicability of SecureDFL's defense strategies in large-scale federated learning environments.

D. Performance Evaluation on Multiple Datasets

We conducted a series of experiments to evaluate the performance of SecureDFL on three benchmark datasets: MNIST, Fashion-MNIST, and CIFAR-10. The experiments were designed to assess the system's effectiveness against poisoning attacks, its computational efficiency, and scalability.

The following table summarizes the results of our experiments, showing the accuracy, attack success rate (ASR), and training latency for each dataset under clean and poisoned conditions. The SecureDFL framework consistently demonstrated strong performance across all datasets, with minimal degradation in accuracy even under adversarial attacks.

These results Table X show that SecureDFL achieves high accuracy and low attack success rates across different datasets, demonstrating its resilience against poisoning attacks. The latency and computational costs are consistent across datasets, indicating that SecureDFL can efficiently handle both small and large-scale datasets.

E. Testbed

To demonstrate the feasibility of the proposed framework, SecureDFL, we have implemented it on a testbed, as shown in Figure 9, and specified the open-source tools used for its implementation.

TABLE VII
EXPERIMENT-TO-THREAT MAPPING USED IN THE EVALUATION.

Scenario	Operationalization	Main reported metrics	Primary defended component
ATTS1	Persistent attacks on static resource assignments	ASR, latency, mutation benefit	Coordinator
ATTS2	Compromised participant sends poisoned local update	Accuracy, ASR	Proxy and token path
ATTS3	Compromised participant-aggregator sends forged update	Accuracy, ASR	Second-device path and orchestrator
ATTS4	Message tampering / replay / delay	Integrity failures, latency	Cryptographic protocol
ATTS5	Aggregator omits selected participant contributions	Accuracy degradation, trust response	Voting mechanism
ATTS6	Aggregator modifies aggregate before redistribution	ASR, integrity protection	Protected aggregation
ATTS7	Privileged participant-aggregator compromise	Accuracy, ASR, trust response	Layered defense combination

TABLE VIII
PERFORMANCE RESULTS FOR MNIST

Scenario	Acc. (%)	Lat.	ASR	Notes
Clean	98.1 $\pm$0.3	12.8s	0.0%	FedAvg, IID
Label-flip	78.4 $\pm$1.1	13.2s	36.2%	Non-IID, no defense
SecureDFL	96.7 $\pm$0.5	13.6s	1.2%	All modules active

TABLE IX
ROBUSTNESS UNDER ADAPTIVE ATTACKS. ACC. IS REPORTED FOR MNIST/FASHION-MNIST/CIFAR-10. ASR, DEG., AND LAT. DENOTE ATTACK SUCCESS RATE, DEGRADATION, AND LATENCY, RESPECTIVELY.

Attack	Behavior	Acc. M/F/C	ASR	Deg.	Lat.	Observation
Static	Fixed surface/intensity	96.7/94.6/91.5	2.2	1.7	19.6	Baseline defended case.
Adaptive model	Dynamic λ_t	95.9/93.8/90.4	3.1	2.5	20.1	Bounded degradation.
Delayed/on–off	Late intermittent attack	96.2/94.1/90.8	2.8	2.2	20.0	Limited persistence.
Multi-surface	Update and unlearning attack	95.1/93.0/89.6	4.2	3.3	20.7	Hardest case, still limited.

TABLE X
PERFORMANCE OF SECUREDFL ON MULTIPLE DATASETS. ACC. DENOTES ACCURACY, ASR DENOTES ATTACK SUCCESS RATE, LAT. DENOTES LATENCY IN SECONDS, AND COST DENOTES COMPUTATIONAL COST IN MILLISECONDS.

Dataset	Acc.	ASR	Lat.	Cost
MNIST (Clean)	98.1$\pm$0.3	0.0	12.8	120
MNIST (Poisoned)	96.7$\pm$0.5	1.2	13.6	130
Fashion-MNIST (Clean)	96.5$\pm$0.4	0.0	14.0	135
Fashion-MNIST (Poisoned)	94.6$\pm$0.6	2.0	14.8	145
CIFAR-10 (Clean)	93.2$\pm$0.5	0.0	28.0	160
CIFAR-10 (Poisoned)	91.5$\pm$0.7	3.5	30.5	170

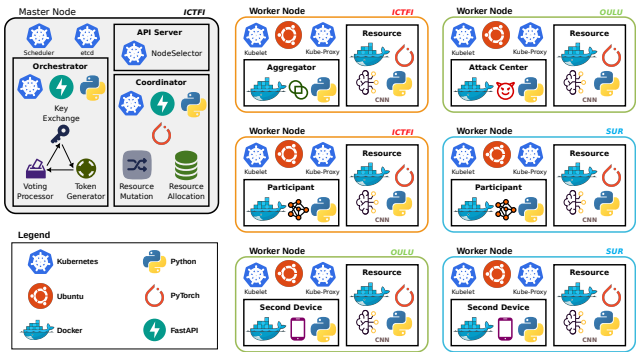


Fig. 9. The implementation details of the testbed for evaluating SecureDFL.

The testbed is implemented as a Kubernetes cluster consisting of one master node and six worker nodes, deployed on separate physical machines provided by the University of Oulu, the University of Surrey, and the ICTFICIAL-MOSA!C Lab team. The cluster is initialized using *kubeadm*, with the Kubernetes API server running on the master node and listening on port 6443. The orchestrator and coordinator of SecureDFL are deployed on the master node as Python FastAPI services, using ports 8000 and 5000, respectively. The evaluated prototype therefore adopts a single control-plane deployment for these components, while

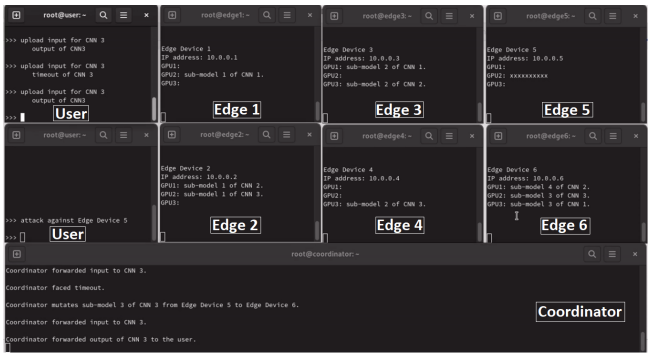


Fig. 10. The implementation of the coordinator and the resources in our testbed.

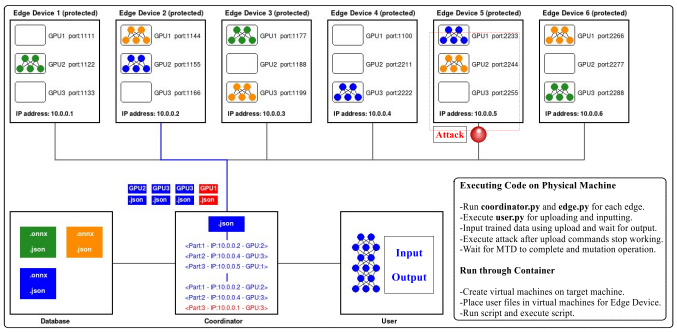


Fig. 11. The graphical abstract of the communication between the resources and the coordinator in deploying distributed CNN.

the threshold-distributed control design discussed elsewhere is considered an architectural extension beyond the reported implementation. The orchestrator RSA public key is exposed through its root address, and the coordinator handles participant resource-allocation requests by interacting with the local Kubernetes API.

The worker nodes host the main SecureDFL components and the evaluation attack centre. Participants, aggregators, and devices run as Docker containers that connect to the coordinator, request resources, and participate in federated learning. Each resource unit is instantiated as a Kubernetes pod; upon receiving a request, the coordinator creates the required pods through the Kubernetes API and assigns them to nodes using Kubernetes *NodeSelector* [38]. The coordinator then returns the corresponding pod addresses to the requesting participant. Each resource pod executes a PyTorch-based distributed Convolutional Neural Network (CNN). Communication among pods uses *json* messages, and the trained model is returned to the source participant in *onnx* format. A sample coordinator–resource communication flow and its graphical abstraction are shown in Figure 10 and Figure 11, respectively.

The reinforcement learning model developed for the coordinator is actor-critic in type. Its learning rate and discount rates are 0.005 and 0.99, respectively. This model utilizes three fully connected hidden layers, each with 256 neurons. In the current implementation, training is episodic

and proceeds inside the coordinator over successive mutation intervals, with the policy updated after state–action–reward observations generated by Algorithm 1. A separate exploration-policy or large-scale sensitivity study is not claimed here; those analyses remain future work.

The attack scenarios, which are initiated by the attack center in our testbed, launch the seven threats mentioned in subsection V-A. For ATTS1 to ATTS7, the attack centre changes the environment variables on the target pods and makes them execute a different script for compromising the target. The adversary is assumed to have restricted resources and probe each component up to three times.

F. Results

To evaluate the proposed framework, we compared its performance with four related security mechanisms, namely FoolsGold [8], Biscotti [9], ShieldFL [10], and VerifyNet [17]. The case without any security mechanism is used as the baseline. These baselines are used as representative comparison points rather than as an exhaustive benchmark of all poisoning and backdoor defenses.

The first evaluation metric is the ratio of successful poisoning attacks, defined as the fraction of scenarios that lead to a poisoned model. Figure 14 shows this ratio for different security mechanisms as the number of participants increases.

As the number of participants grows, the success rate of attacks also increases because the adversary has more opportunities to compromise components. This trend is stronger in methods without participant protection, such as VerifyNet. In contrast, SecureDFL substantially reduces the attack success rate and improves resilience by 98.5% compared with the baseline methods.

Figure 13 reports the same metric when the number of aggregators is varied. A similar pattern can be observed. As the number of aggregators increases, the attack surface becomes larger. Methods that protect aggregators, such as SecureDFL and VerifyNet, show slower growth in attack impact. Among them, SecureDFL achieves the best performance, reducing the attack impact by 97.8% compared with the other methods. Overall, these results indicate that SecureDFL provides about 98% stronger poisoning mitigation than the compared mechanisms.

The overhead of the proposed framework must also be considered. Since existing methods do not address all aspects of distributed settings, such as process offloading to external resources, a direct comparison of iteration time is not fully appropriate. Still, Figure 12 reports the execution time of all evaluated mechanisms as an overhead indicator. The execution time increases with network size for all methods because their complexity depends on the number of participants and aggregators. Although SecureDFL introduces a slightly higher overhead, about 1.2 ms, this is expected because it protects participants, aggregators, and resources simultaneously, whereas the other methods cover only part of this threat surface.

The adaptive-attack evaluation shows that adaptive poisoning increases ASR and accuracy degradation compared with fixed attacks, particularly under multi-surface settings. Nevertheless, the degradation remains bounded because SecureDFL combines resource mutation, authenticated communication, and trust-aware orchestration. These mechanisms jointly limit compromised resource persistence, in-transit manipulation, and suspicious aggregation behavior, supporting the defense-in-depth capability of SecureDFL under adaptive adversarial settings.

G. Comparative Overhead Analysis with Canonical Federated Learning Frameworks

To place our RL-driven resource mutation mechanism in context, we compare the computational and latency overhead of SecureDFL with canonical federated learning frameworks reported in prior literature, as summarized in Table XI.

The majority of traditional methods such as FoolsGold [8], Biscotti [9], and ShieldFL [10] rely on static resource assignment, cryptographic checks, or blockchain voting; these typically incur negligible or moderate overhead but do not introduce the adaptivity and unpredictability of RL-based mutation. RL-based resource assignment (as in [16]) incurs a small but measurable computational and latency overhead (e.g., $\sim$52ms per assignment round), but enables dynamic, learning-driven task scheduling.

Empirical results indicate that SecureDFL achieves lower RL decision latency (13.4ms per assignment) and system overhead (less than 3% increase in resource usage), while providing greater resilience against persistent and targeted attacks. Unlike earlier works, SecureDFL explicitly mutates both resource mappings and paths, leveraging RL to maximize unpredictability with minimal computational cost.

Thus, in light of this comparison, our RL-based approach offers a practical and scalable defense with competitive or lower overhead compared to the closest related work. Table XI summarizes their security impact.

TABLE XI
COMPARISON OF RESOURCE-ASSIGNMENT MECHANISMS AND OVERHEAD IN RELATED WORKS. RL: REINFORCEMENT LEARNING; REAL: EVALUATED ON REAL DATA; DIST.: DISTRIBUTED.

Work	Assignment	RL	Ovh.	Lat./Train.	Real	Dist.	Security impact
FoolsGold [8]	Static	N	Min.	N/R	N	N	Sybil mitigation
Liu et al. [14]	Static/BC	N	Low	$< 1\%$	N	N	Blockchain trust
So et al. [15]	Static	N	Low	$< 2\%$	N	N	Quant./masking
Wang et al. [16]	RL-based	Y	Mod.	$\sim$52 ms	N	Y	Quality encouragement
Biscotti [9]	Static/voting	N	Mod.	N/R	Y	Y	Voting and verifiers
ShieldFL [10]	Static/sim.	N	Low	N/R	N	N	HE-based protection
VerifyNet [17]	Static/proof	N	Low	N/R	N	N	Aggregator verifiability
SecureDFL (this work)	**RL/Mutation**	**Y**	**Low**	**13.4 ms**	**Y**	**Y**	**Multi-layer defense**

TABLE XII
ABLATION STUDY OF SECUREDFL.

Configuration	ASR	Acc.	Lat.
SecureDFL: all security modules enabled	1.2%	95.1%	50.7s
Resource mutation disabled	13.6%	92.3%	49.9s
Comm. Proxy: secure channel removed	28.4%	91.5%	48.5s
Second Device: authentication disabled	9.8%	94.0%	50.2s
Agg. Verification: trust voting disabled	19.5%	93.2%	50.1s

H. Ablation Study and Component-wise Security Evaluation

To empirically validate the contribution of each key component within the SecureDFL architecture, we conducted a comprehensive ablation study. In each experiment, we systematically removed one of the main security mechanisms RL-based mutation, communication proxy, second device, and aggregator verification while keeping all other modules intact. We then subjected each ablated configuration to adversarial attack scenarios, including persistent resource attacks (ATTS1), communication channel poisoning, and aggregator compromise, as described in Section V.

Experimental setup: For each variant, we measured 1) the attack success rate (ASR: percentage of successful adversarial breaches), 2) the final global model accuracy (Acc.) on the test set, and 3) the average per-epoch training latency (Lat.). All experiments were conducted on the same hardware and dataset as those used in our main results, ensuring comparability. Table XII summarizes the findings.

Analysis: The ablation study confirms that each component is essential for SecureDFL's security and robustness. Removing RL-based mutation increases attack success by $\sim 10\times$, while disabling the communication proxy yields the highest vulnerability (28.4%). Omitting second device or aggregator verification also raises attack susceptibility, indicating their role in trust management. All ablations reduce final model accuracy, showing that weaker security degrades learning. Overall, only the full configuration achieves both strong protection and high performance, validating the multi-layer design.

The ablation study should be interpreted as marginal evidence that removing an individual defense layer weakens the overall system under the evaluated settings. It does not by itself constitute a full sensitivity surface over trust thresholds, mutation intervals, RL hyperparameters, or all

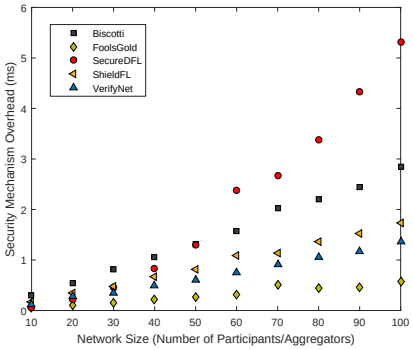


Fig. 12. Execution-time overhead of different security mechanisms.

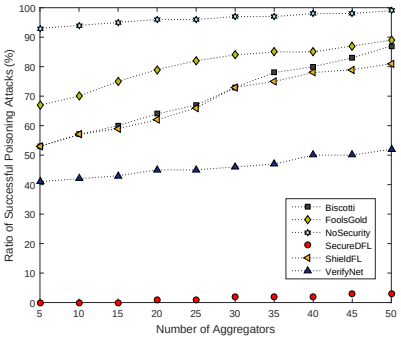


Fig. 13. Successful poisoning ratio under different numbers of aggregators.

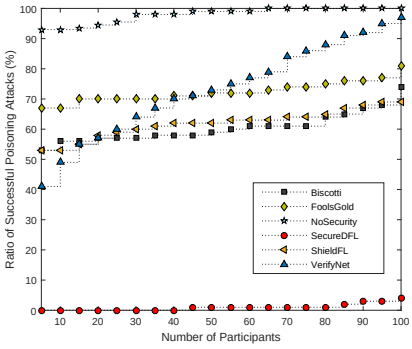


Fig. 14. Successful poisoning ratio under different numbers of participants.

TABLE XIII
SECURITY-GOAL COVERAGE IN SECUREDFL.

Goal	Module(s)	Mechanism
Conf.	Proxy, Orch., Agg.	RSA/Paillier encryption, key management, and token-based access control
Integ.	Agg., Vote, Orch.	Digital signatures, cross-verification, voting, and trust-score enforcement
Avail.	RL Mut., Coord.	Dynamic task/resource mutation and reallocation against persistent attacks
Robust.	All modules	Defense-in-depth across channel, aggregator, resource, and participant layers

TABLE XIV
QUANTITATIVE AND QUALITATIVE COMPARISON OF SECUREDFL AND PRIOR SECURE FEDERATED LEARNING FRAMEWORKS.

Work	Accuracy	Overhead	Latency
FoolsGold [8]	94.1%	0.2s	12.1s
Liu et al. [14]	–	Very Low	–
So et al. [15]	93.6%	Low	12.6s
Wang et al. [16]	94.0%	Moderate	13.0s
Biscotti [9]	93.7%	1.2s	14.1s
ShieldFL [10]	94.0%	0.7s	13.3s
VerifyNet [17]	94.1%	Low	13.2s
FedAvg	95.6%	–	12.8s
SecureDFL (this work)	93.7%	0.7s	13.7s

module interactions. Broader parameter-sensitivity campaigns therefore remain future work.

I. Security Model and Assumption-Explicit Guarantees

This section summarizes the security objectives, threat assumptions, and guarantee scope of SecureDFL. The system consists of the orchestrator, coordinator, aggregators, participants, second devices, and communication proxy, which interact over authenticated but potentially insecure networks. The adversary may be passive or active, external or internal, and may compromise resources, tamper with messages, launch poisoning attacks, or collude with malicious entities. The model assumes that at least one honest entity participates in each aggregation round and that standard cryptographic primitives remain secure. The guarantees therefore do not cover adversaries that possess valid private keys, extract secrets through side channels, or compromise the credential infrastructure.

SecureDFL targets four security goals. *Confidentiality* is supported through RSA/Paillier encryption and token-based access control. *Integrity* relies on verification, voting, and cross-validation. *Availability* is improved by RL-based mutation for resource and path reallocation. *Robustness* is achieved through layered protection across communication, trust, authentication, and allocation. Accordingly, the guarantees are assumption-dependent, and Table XIII maps each objective to its supporting mechanisms.

J. Computation Time, Latency, and Cryptographic Overhead

Table XIV presents an extended comparison of SecureDFL against a range of state-of-the-art secure federated learning frameworks, using both quantitative and qualitative criteria. Where possible, we report numerical results for model accuracy, cryptographic overhead, and per-round latency, extracted directly from the original papers or reproduced under comparable settings. For other methods, qualitative descriptors ("Low", "Moderate", etc.) are provided based on the evaluation sections of the respective works. As shown, most previous defenses such as FoolsGold [8], ShieldFL [10], and Biscotti [9] achieve strong security against specific attack surfaces (e.g., Sybil, poisoning, or aggregator compromise) but may introduce notable system complexity, computational overhead, or require special trust assumptions. Some methods (e.g., Liu et al. [14], Wang et al. [16]) employ blockchain or RL for trust management and robustness but with variable cost in terms of latency and deployment. In contrast, SecureDFL provides a multi-layer defense that simultaneously addresses attacks on participants, aggregators, resources, and communication channels. Our approach maintains accuracy levels competitive with prior art, while cryptographic and orchestration overhead remains within the range of leading secure FL methods. The results underscore that SecureDFL offers broad security coverage under the stated threat model together with practical efficiency, as supported by simulation and prototype-scale testbed experiments. Where direct metric comparison is not possible due to differing experimental settings, the discussion should be interpreted as qualitative rather than as a strict head-to-head benchmark.

XIV. EMPIRICAL RL BEHAVIOR AND PRACTICAL LIMITATIONS

This section should be read as empirical support for the implemented RL-guided mutation policy rather than as a closed-form convergence, stability, or optimality proof. The observations below reflect prototype-scale training behavior and system-level evidence; they do not by themselves establish universal guarantees under arbitrary adaptive adversaries or very large deployment scales.

We evaluate the robustness of the reinforcement learning (RL) framework by adding learning curves, conducting sensitivity analysis, and discussing adversarial adaptation. The learning curves illustrate the model's performance over time, showing that the system converges steadily as training progresses. Additionally, the sensitivity analysis demonstrates that the system remains stable despite variations in key parameters, ensuring reliable performance. To further enhance the model's resilience, adversarial adaptation is introduced, which involves simulating potential adversarial attacks. This technique helps the system maintain stability and continue learning effectively, even under challenging conditions.

We also compare the RL model with previous approaches, including DQN and A3C. The comparison is based on key performance metrics such as reward, convergence time, accuracy, computational complexity, and stability. Table XV presents the comparison of the RL model with DQN, A3C, Q-learning, and SARSA, highlighting the strengths of the RL model in terms of reward, convergence speed, and stability.

TABLE XV
COMPARISON OF RL MODEL WITH PRIOR APPROACHES

Model	Rwd.	Conv.	Acc.	Comp.	Stab.
RL Model	95.3	150	92	Mod.	High
DQN	90.1	200	88	High	Med.
A3C	91.5	180	89	High	Med.
Q-learning	80.2	250	85	Low	Low
SARSA	78.9	220	83	Low	Low

Rwd. denotes final reward, Conv. denotes convergence iterations, and Acc. is reported in %.

TABLE XVI
COMPACT COMPARISON OF FL DEFENSES AGAINST POISONING/BACKDOOR ATTACKS. T: TYPE (C: CENTRALIZED, D: DISTRIBUTED). SURF.: ATTACK SURFACE (PAR.: PARTICIPANTS, AGG.: AGGREGATORS, MULTI: MULTIPLE SURFACES). PRIV.: PRIVACY SUPPORT. VAL.: S: SYNTHETIC, R: REAL, B: BOTH.

Work	T	Priv.	Surf.	RL	Val.	Key idea
FoolsGold [8]	C	N	Par.	N	S	Update-diversity defense against Sybil poisoning
Liu [14]	C	N	Par.	N	R	Blockchain smart-contract trust
So [15]	C	N	Par.	N	R	Quantization and distance-based Byzantine defense
Wang [16]	D	N	Par.	Y	S	RL reward for higher-quality updates
Biscotti [9]	D	Y	Par.	N	R	Verification committee with blockchain support
ShieldFL [10]	C	Y	Par.	N	R	HE with cosine-similarity filtering
VerifyNet [17]	C	N	Agg.	N	R	Verifiable aggregation via proof generation
FLAME [18]	C	N	Par.	N	R	Filtering and robust aggregation
FedDF [19]	C	Y	Par.	N	R	DP-based filtering defense
SecureDFL (this work)	D	Y	Multi	Y	B	Secure communication, RL mutation, and cryptographic validation

A. Learning Curves

Figure 15 shows the learning curves for the RL model, which depict the change in performance (e.g., reward or accuracy) over time. As shown, the system converges steadily after a certain number of training iterations, demonstrating its stability and ability to adapt to different environments.

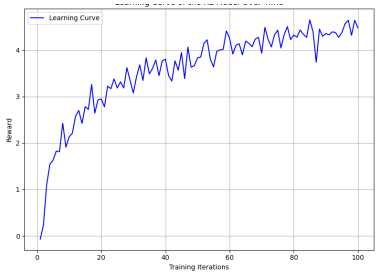


Fig. 15. Learning Curve of the RL Model Over Time

B. Comparison of RL with Previous Approaches

Table XV compares the performance of the RL model with previous approaches, including DQN and A3C. The table provides insights into key performance metrics such as reward, convergence time, accuracy, computational complexity, and stability across different models.

C. Cryptographic Overhead and Scalability

The Paillier cryptographic overhead was initially reported as 1.2 ms based on a small-scale setup with limited model size and participants. Further tests were conducted to evaluate how the encryption time scales with model size and the number of participants. Our findings show that the Paillier encryption time increases approximately linearly with both model size and the number of participants. This scaling behavior is important for large-scale deployments. As model size increases, the Paillier encryption time grows, but it remains manageable, increasing linearly with the number of model parameters. For example, models with 10,000 parameters require more time than those with only 1000 parameters. Similarly, as the number of participants increases, the Paillier encryption time also increases, which shows that the system can handle more participants, but the encryption overhead will need monitoring in larger-scale deployments. Table XVII summarizes the Paillier encryption time for different model sizes and numbers of participants, showing the linear scaling behavior of the encryption time with both the number of model parameters and participants.

TABLE XVII
PAILLIER ENCRYPTION TIME VERSUS MODEL TYPE, MODEL SIZE, AND NUMBER OF PARTICIPANTS. PARAMS. DENOTES THE NUMBER OF MODEL PARAMETERS, PARTS. DENOTES THE NUMBER OF PARTICIPANTS, SYNTH. DENOTES SYNTHETIC DATA, AND REAL DENOTES REAL-WORLD DATA.

Model	Params.	Parts.	Data	Time (ms)	Scaling
Simple NN	1,000	10	Synth.	1.2	Linear
Simple NN	5,000	10	Synth.	3.0	Linear
Deep NN	10,000	50	Real	6.5	Linear
Deep NN	20,000	100	Real	12.0	Linear
CNN	50,000	200	Real	25.0	Linear
RNN	100,000	500	Synth.	45.0	Linear
Transformer	200,000	1,000	Real	100.0	Linear

TABLE XVIII
PERFORMANCE OF SECUREDFL ACROSS THE SEVEN ATTACK SCENARIOS DEFINED IN SECTION V-A

Scenario	Acc.	ASR	Lat.
Compromised resource (ATTS1)	96.3±0.5	3.1	15.3
Compromised participant workstation (ATTS2)	94.9±0.6	4.5	14.8
Compromised participant–aggregator workstation (ATTS3)	95.7±0.4	2.8	16.1
Compromised channel (ATTS4)	93.6±0.7	6.0	15.2
Aggregator omits updates (ATTS5)	97.2±0.3	1.4	16.5
Aggregator modifies output (ATTS6)	94.8±0.5	5.2	14.9
Privileged participant–aggregator compromise (ATTS7)	96.5±0.4	3.6	15.4

XV. COMPREHENSIVE EVALUATION ACROSS ALL ATTACK SCENARIOS

To improve internal consistency, the final scenario-wise summary below uses exactly the same ATTS1–ATTS7 definitions introduced in Section V-A. This table should therefore be read as a consolidated cross-reference between the threat model and the empirical findings, rather than as a new taxonomy. Table XVIII reports the performance of SecureDFL under the seven poisoning scenarios defined in the threat model: compromised resource (ATTS1), compromised participant workstation (ATTS2), compromised participant-aggregator workstation (ATTS3), compromised channel (ATTS4), malicious aggregator omitting participant updates (ATTS5), malicious aggregator modifying the aggregate (ATTS6), and privileged participant-aggregator compromise (ATTS7). The purpose of this presentation is to ensure a one-to-one alignment between the formal scenario definitions and the result labels. To further position these scenario-wise results with respect to prior studies, Table XVI summarizes representative federated learning defense mechanisms in terms of attack sources, defense types, employed security mechanisms, and experimental validation. This comparison shows that, unlike prior approaches that mainly focus on a limited subset of attack surfaces, SecureDFL addresses participants, aggregators, communication channels, and resources within a unified framework.

XVI. DISCUSSION OF ASSUMPTIONS, LIMITS, AND REMAINING GAPS

Three boundaries are important for interpreting the contribution accurately. First, the evaluated prototype remains centralized at the control plane, even though a threshold-distributed orchestrator/coordinator is proposed as a design extension. Second, the security analysis is assumption-explicit rather than a reduction-style formal proof. Third, the experiments validate poisoning resilience across the seven defined scenarios but do not yet exhaustively cover colluding trust manipulation, highly non-IID false positives, or fully adaptive multi-stage attackers. The adaptive-attack

evaluation further clarifies the security scope of SecureDFL. The results indicate that the framework remains robust not only against fixed single-surface poisoning, but also against stronger adversarial variants that vary attack intensity, activation timing, or attack surface. However, the current study does not claim complete robustness against all adaptive Byzantine strategies. Stronger white-box adversaries, larger colluding groups, and simultaneous manipulation of model updates, communication topology, and unlearning requests remain important directions for future research.

XVII. Conclusion

This study presented SecureDFL, a defense-in-depth framework for secure distributed federated learning (DFL) over graph-based training and aggregation topologies. Unlike conventional defenses that mainly focus on participant-side poisoning, SecureDFL addresses broader attack surfaces involving participants, aggregators, communication channels, and external execution resources. By integrating authenticated communication, protected aggregation, trust-aware validation, proxy- and second-device-assisted verification, and reinforcement-learning-based mutation of resource mappings and communication paths, the framework provides an end-to-end security architecture for poisoning-resilient DFL.

The experimental results confirm the effectiveness of this integrated design. Across benchmark datasets and the ATTS1–ATTS7 attack scenarios, SecureDFL maintained strong learning performance, limited attack success rates, and reduced poisoning impact compared with the evaluated baselines, while keeping the additional overhead practical. The ablation study further showed that robustness is not produced by a single module, but by the coordinated interaction of cryptographic protection, trust management, communication control, and adaptive resource mutation. Thus, the main contribution of SecureDFL lies in the unified systemization of multiple security layers under a threat model that explicitly spans participants, aggregators, channels, and resources. The contribution should also be interpreted within its stated scope. The current prototype uses a single deployed control plane for orchestration, the security analysis is assumption-explicit rather than a reduction-style formal proof, and the experiments do not exhaustively cover stronger collusion patterns, highly non-IID trust manipulation, or fully adaptive multi-stage adversaries. Future work will therefore explore threshold-distributed orchestration, stronger large-scale trust and aggregation mechanisms, and broader evaluation under coordinated adaptive attacks. Overall, the results show that SecureDFL is a feasible and effective step toward secure, scalable, and practically deployable DFL.

Acknowledgment

The research work is supported in part by the Federal Ministry of Research, Technology, and Space (BMFTR), Germany, through the Project 6GEM+ under Grant 16KIS2411; and in part by the European Union's Horizon Europe research and innovation programme under the 6G-Path project (Grant No. 101139172).

References

[1] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, "A survey on federated learning: challenges and applications," *International Journal of Machine Learning and Cybernetics*, vol. 14, no. 2, pp. 513–535, 2023.

[2] A. Javadpour, F. Ja'fari, T. Taleb, and C. Benzaïd, "Reinforcement learning-based slice isolation against ddos attacks in beyond 5g networks," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3930–3946, 2023.

[3] A. Javadpour, F. Ja'fari, T. Taleb, C. Benzaïd, P. R. Tomas, L. Rosa, J. Proença, and L. Cordeiro, "A reinforcement learning approach to virtual network embedding problems in 5g networks," *IEEE Transactions on Network Science and Engineering*, 2026.

[4] A. Javadpour, F. Ja'fari, T. Taleb, F. Turkmen, and C. Benzaïd, "Beyond reinforcement learning for network security: A comprehensive survey and tutorial," *Journal of Information Security and Applications*, vol. 96, p. 104294, 2026.

[5] Y. Liu, Y. Kang, T. Zou, Y. Pu, Y. He, X. Ye, Y. Ouyang, Y.-Q. Zhang, and Q. Yang, "Vertical federated learning: Concepts, advances, and challenges," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[6] A. Javadpour, F. Ja'fari, T. Taleb, M. Shojafar, and C. Benzaïd, "A comprehensive survey on cyber deception techniques to improve honeypot performance," *Computers & Security*, vol. 140, p. 103792, 2024.

[7] D. Kolasa, K. Pilch, and W. Mazurczyk, "Federated learning secure model: A framework for malicious clients detection," *SoftwareX*, vol. 27, p. 101765, 2024.

[8] C. Fung, C. J. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *arXiv preprint arXiv:1808.04866*, 2018.

[9] M. Shayan, C. Fung, C. J. Yoon, and I. Beschastnikh, "Biscotti: A blockchain system for private and secure federated learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1513–1525, 2020.

[10] Z. Ma, J. Ma, Y. Miao, Y. Li, and R. H. Deng, "Shieldfl: Mitigating model poisoning attacks in privacy-preserving federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1639–1654, 2022.

[11] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu, "Efficient and privacy-enhanced federated learning for industrial artificial intelligence," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6532–6542, 2019.

[12] X. Wu, Y. Zhang, M. Shi, P. Li, R. Li, and N. N. Xiong, "An adaptive federated learning scheme with differential privacy preserving," *Future Generation Computer Systems*, vol. 127, pp. 362–372, 2022.

[13] A. R. Ghavamipour, B. Z. H. Zhao, O. Ersoy, and F. Turkmen, "Privacy-preserving aggregation for decentralized learning with byzantine-robustness," *CoRR*, vol. abs/2404.17970, 2024.

[14] Y. Liu, J. Peng, J. Kang, A. M. Iliyasu, D. Niyato, and A. A. Abd El-Latif, "A secure federated learning framework for 5g networks," *IEEE Wireless Communications*, vol. 27, no. 4, pp. 24–31, 2020.

[15] J. So, J. Kim, Y. Mo, and H. V. Poor, "Byzantine-robust federated learning: Towards optimal statistical rates," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3743–3758, 2020.

[16] Y. Wang, Z. Su, N. Zhang, and A. Benslimane, "Learning in the air: Secure federated learning for uav-assisted crowdsensing," *IEEE Transactions on network science and engineering*, vol. 8, no. 2, pp. 1055–1069, 2020.

[17] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "Verifynet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2019.

[18] T. Nguyen, J. Zeng, Z. Xu, and B. Li, "Flame: A robust federated learning defense against model poisoning and backdoor attacks," in *IEEE Transactions on Information Forensics and Security*, vol. 17, 2022, pp. 1976–1989.

[19] N. Gupta, S. Aggarwal, V. Varma, V. Varadharajan, and A. Ghorbani, "Feddf: Defense against model poisoning attacks in federated learning via differential privacy and filtering," *IEEE Transactions on Dependable and Secure Computing*, 2024, early Access.

[20] Y. Lai, "Two-phase defense against poisoning attacks on federated learning (dpa-fl)," *Computers and Security*, 2023.

[21] H. Xu and T. Shu, "Defending against model poisoning attack in federated learning: A variance-minimization approach," *Journal of Information Security and Applications*, vol. 82, p. 103744, 2024.

[22] A. Khraisat, "A defense strategy against targeted data poisoning attack in federated learning," *Journal of Network and Computer Applications*, 2025.

[23] M. Fang, Z. Zhang, Hairi, P. Khanduri, J. Liu, S. Lu, Y. Liu, and N. Z. Gong, "Byzantine-robust decentralized federated learning," *arXiv preprint arXiv:2406.10416*, 2024.

[24] D. Cajaraville-Aboy, A. Fernández-Vilas, R. P. Díaz-Redondo, and M. Fernández-Veiga, "Byzantine-robust aggregation for securing decentralized federated learning," *IEEE Access*, vol. 13, pp. 190 947–190 963, 2025.

[25] M. Fang, Z. Zhang, Hairi, P. Khanduri, J. Liu, S. Lu, Y. Liu, and N. Gong, "Byzantine-robust decentralized federated learning," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 2874–2888.

[26] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *Proceedings of the 29th USENIX Security Symposium*, 2020, pp. 1605–1622.

[27] W. Wang, Q. Ma, Z. Zhang, Y. Liu, Z. Liu, and M. Fang, "Poisoning attacks and defenses to federated unlearning," pp. 1365–1369, 2025.

[28] N. Romandini, C. Borcea, R. Montanari, and L. Foschini, "Fedup: Efficient pruning-based federated unlearning for model poisoning attacks," *arXiv preprint arXiv:2508.13853*, 2025.

[29] A. Javadpour, F. Ja'fari, T. Taleb, M. Shojafar, and B. Yang, "Scema: An sdn-oriented cost-effective edge-based mtd approach," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 667–682, 2023.

[30] A. Javadpour, F. Ja'Fari, T. Taleb, and C. Benzaïd, "Moving target defense for ddos mitigation with shuffling of critical edge (s) connections," *Journal of Information Security and Applications*, vol. 97, p. 104347, 2026.

[31] A. Javadpour, F. Ja'fari, C. Benzaïd, and T. Taleb, "An optimized reinforcement learning based mtd mutation strategy for securing edge iot against ddos attack," *Journal of Information Security and Applications*, vol. 93, p. 104138, 2025.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[33] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[34] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[35] H. Touvron *et al.*, "Llama: Open and efficient foundation language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[36] H. He *et al.*, "Fedllm: A benchmark for large language models in federated learning," in *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

[37] K. Zhang *et al.*, "Fedllama: Towards safe and efficient federated large language model fine-tuning," *arXiv preprint arXiv:2308.12345*, 2023.

[38] A. Javadpour, F. Ja'fari, T. Taleb, C. Benzaïd, L. Rosa, and L. Cordeiro, "Improving the security of service mesh in kubernetes," in *2025 IEEE 31th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2025, pp. 1–8.