Uncovering Attacks and Defenses in Secure Aggregation for Federated Deep Learning

Yiwei Zhang
Department of Computer Science
Purdue University
West Lafayette, USA
yiweizhang@purdue.edu

Rouzbeh Behnia
School of Information Systems and Management
University of South Florida
Sarasota, USA
behnia@usf.edu

Attila A. Yavuz
Department of Computer Science and Engineering
University of South Florida
Tampa, USA
attilaayavuz@usf.edu

Reza Ebrahimi
School of Information Systems and Management
University of South Florida
Tampa, USA
ebrahimim@usf.edu

Elisa Bertino
Department of Computer Science
Purdue University
West Lafayette, USA
bertino@purdue.edu

***Abstract*—Federated learning enables the collaborative learning of a global model on diverse data, preserving data locality and eliminating the need to transfer user data to a central server. However, data privacy remains vulnerable, as attacks can target user training data by exploiting the updates sent by users during each learning iteration. Secure aggregation protocols are designed to mask/encrypt user updates and enable a central server to aggregate the masked information. MicroSecAgg (PoPETS 2024) proposes a single server secure aggregation protocol that aims to mitigate the high communication complexity of the existing approaches by enabling a one-time setup of the secret to be re-used in multiple training iterations. In this paper, we identify a security flaw in the MicroSecAgg that undermines its privacy guarantees. We detail the security flaw and our attack, demonstrating how an adversary can exploit predictable masking values to compromise user privacy. Our findings highlight the critical need for enhanced security measures in secure aggregation protocols, particularly the implementation of dynamic and unpredictable masking strategies. We propose potential countermeasures to mitigate these vulnerabilities and ensure robust privacy protection in the secure aggregation frameworks.**

***Index Terms*—secure aggregation, federated learning, deep learning, data privacy, inference attack**

I. Introduction

Federated Learning (FL) enables collaborative learning of a shared model between distributed parties while keeping the data local, mitigating data privacy and collection challenges common in traditional centralized learning. In large-scale FL, clients with limited computational resources, such as mobile devices, can contribute to training a global model with the assistance of a central server. In each iteration, the central server collects the local model updates from clients, trains the global model using the client's local data, aggregates them, and refines them. However, recent attacks have demonstrated that deploying a plain FL paradigm is insufficient to protect the privacy of the participating users' data [1], [2], [3]. More specifically, these attacks can undermine the confidentiality of the training data by only having access to the user updates.

To mitigate these risks, secure aggregation protocol has been proposed [4]. One prominent approach is FastSecAgg [5]. It utilizes multi-party computation (MPC) techniques to securely aggregate user updates. Another common scheme is masking-based [6], [7], [8], where random masking terms are added to user updates and finally cancel out during aggregation to prevent disclosure of individual information. Despite these security advancements, the communication and computational complexity of traditional secure aggregation methods remains a significant challenge, particularly when applied to large-scale FL settings or models such as large language models (LLMs) [9] with many participating users. In response to this, Guo et al. introduced MicroSecAgg (MicroSecAgg)[10], which improves upon existing methods by employing a one-time setup phase that distributes the necessary secret material for multiple iterations, reducing the overhead caused by continually refreshing the masking terms [4], [7].

Motivation and Contributions: This paper specifically evaluates the MicroSecAgg protocol and identifies a critical vulnerability that compromises its effectiveness. While MicroSecAgg introduces significant efficiency improvements, we have discovered that its handling of secret material during the aggregation phase leaves it vulnerable to privacy attacks. In particular, an adversary capable of intercepting masked updates can compute differences between user updates, which can be exploited to infer private training data [1], [2]. Such a vulnerability is particularly concerning for applications involving sensitive data, such as healthcare or finance, where data privacy is paramount.

To address this flaw in MicroSecAgg, we propose several improvements that strengthen the protocol's security while maintaining its efficiency. Specifically, we redesign how secret material is generated and shared to ensure that the masking process remains robust, even in the face of sophisticated inference attacks. We introduce a dynamic masking mechanism that generates unique masks for each iteration, combining a constant shared key with a dynamic component (e.g., a random value or iteration number). This ensures that masked updates remain secure across multiple iterations, preventing adversaries from inferring private data by comparing updates. Furthermore, we propose optimizations to the aggregation phase, which enhance both the security and efficiency of MicroSecAgg without introducing significant communication or computational overhead.

In summary, our contributions are as follows:

- We identify a fundamental vulnerability in the MicroSecAgg protocol that exposes user data to privacy breaches.
- We present an attack that exploits this vulnerability, demonstrating how an adversary can compromise the privacy of individual users' training data.
- We propose improvements to the MicroSecAgg protocol that enhance its security and efficiency by introducing more robust handling of secret material during the aggregation phase.

Through this work, we aim to advance the state of the art in privacy-preserving technologies for federated learning and provide stronger defenses against emerging privacy threats.

II. Preliminary

A. Notions and Cryptographic Primitives

We use $[n]$ to represent the set $\{1, \ldots, n\}$. Users are denoted by i and Server is denoted by $\mathcal{S}$. The number of users in a list $\mathcal{U}$ is represented by $|\mathcal{U}|$, while the set of online users is represented by $\mathcal{O}$. Vectors are denoted by x_i, where i indexes individual vectors of User i. A pseudorandom function $\mathtt{PRF} : \{0,1\}^* \rightarrow r$ is defined where r has the same dimension as the user update vectors x_i.

MicroSecAgg employs Shamir's t-out-of-n secret sharing [11], as defined below, to deal with offline users.

Definition 1 (Shamir Secret Sharing)**.** *Shamir Secret Sharing allows a secret s to be split into n shares such that any subset of t shares can reconstruct the secret, while fewer than t shares reveal nothing.*

- $\mathtt{SS.share}(s, n, t) \rightarrow \{s_1, \ldots, s_n\}$*: On the input of a secret s, the number of desired shares n, and a threshold t, it splits the secret s into n shares with a threshold t for reconstruction and returns the n shares $\{s_1, \ldots, s_n\}$.*
- $\mathtt{SS.recon}(\{s_1, \ldots, s_t\}, t) = s$*: On the input of at least t shares, $\{s_1, \ldots, s_t\}$, and the threshold t, it reconstructs and return the secret s.*

Definition 2 (Authenticated Encryption)**.** *Symmetric authenticated encryption (AE) scheme ensures that messages between honest parties cannot be extracted or tampered with by an adversary. The AE scheme is assumed to satisfy IND-CCA2 security, which includes:*

- $\mathtt{AE.gen}(1^\kappa) \rightarrow k$*: On the input of the security parameter κ, it returns a symmetric key k.*
- $\mathtt{AE.enc}(m, k) \rightarrow c$*: On the input of the plaintext m and the key k, it encrypts the message and returns the ciphertext c.*
- $\mathtt{AE.dec}(c, k) \rightarrow m$*: On the input of the ciphertext c and the key k, it decrypts ciphertext and return the plaintext message m.*

Definition 3 (Decisional Diffie-Hellman (DDH) Assumption)**.** *Given $p = 2q + 1$, a generator g of $\mathbb{Z}_p^*$, and random values a, b, c chosen from $\mathbb{Z}_q$, the distributions (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) are computationally indistinguishable.*

Definition 4 (Diffie-Hellman (DH) Key Exchange)**.** *Diffie-Hellman key exchange allows two parties to agree securely on a shared secret over a public channel.*

- $\mathtt{KA.setup}(\kappa) \rightarrow (G', g, q, H)$*: On the input of security parameter κ, it initializes a group G' of order q with generator g and a cryptographically secure hash function H.*
- $\mathtt{KA.gen}(G', g, q, H) \rightarrow (x, g^x)$*: On the input of group G' of order q with generator g and a cryptographically secure hash function H, it generates and returns a pair of keys, i.e., a secret key x and corresponding public key g^x.*
- $\mathtt{KA.agree}(x_u, g^{x_v}) \rightarrow s_{u,v} = H((g^{x_v})^{x_u})$*: On the input of a secret key x_u of party u and the public key g^{x_v} of party v, it computes and returns the shared secret $s_{u,v}$ between two parties u and v.*

B. Security Definitions

In this section, we briefly introduce the definitions of secure aggregation protocol in MicroSecAgg [10].

Definition 5 (Aggregation Protocol)**.** *An aggregation protocol $\Pi(\mathcal{U}, \mathcal{S}, K)$, with a set of users $\mathcal{U}$, a server $\mathcal{S}$, and parameters K, consists of two phases: the **Setup phase** and the **Aggregation phase**.*

- ***Setup phase**: This phase runs once at the beginning of the protocol execution.*
- ***Aggregation phase**: This phase runs for K iterations. At the beginning of each iteration $k \in [K]$, each user $i \in \mathcal{U}$ holds an input x_i^k. At the end of each iteration k, the server $\mathcal{S}$ computes and outputs the aggregated value:*

$$w_k = \sum_{i \in \mathcal{U}} x_i^k$$

Definition 6 (Correctness with Dropouts)**.** *Suppose the total number of users is $|\mathcal{U}| = n$, the aggregation protocol Π is said to ensure correctness with a dropout rate δ if, for each iteration $1 \leq k \leq K$ and for any set of offline users $\mathtt{offline}_k \subseteq U$ such that $|\mathtt{offline}_k| < \delta n$, the server $\mathcal{S}$ correctly outputs the aggregated value $w_k = \sum_{i \in \mathcal{O} x_i^k}$ at the end of iteration*

k. This holds as long as all users and the server follow the protocol, with the exception that users in offline$_k$ may drop offline during iteration k.

III. MicroSecAgg

To protect the privacy of a user's update w_i, traditional secure aggregation methods based on multi-party computation (MPC) (e.g., [4]) require each user P_i to generate a masking value h_i to obscure their update during aggregation. The mask is constructed so that when all masked updates are aggregated, the sum of the masks cancels out, i.e., $\sum_{i\in[n]} h_i = 0$. To achieve this, each pair of users P_i and P_j need to negotiate pair-wise masks for each iteration - ensuring that the masks cancel out when both users participate in the aggregation. Since these masks cannot be reused, users must renegotiate and generate new masks for each aggregation iteration.

However, traditional methods [4], [7] suffer from certain drawbacks, requiring users to renegotiate and generate new masks for each aggregation iteration. This increases both the communication and computation complexity of such protocols. This process involves exchanging information about the freshly generated masks with several others (either all users in [4] or a subset in [7]), resulting in communication costs growing significantly as the number of users grows.

MicroSecAgg addresses these limitations with a two-phase secure aggregation protocol, eliminating the need for agreeing on fresh masks at each iteration [10]. It introduces reusable masks generated during an initial one-time phase. These masks are then applied consistently across subsequent iterations. Specifically, MicroSecAgg operates in two distinct phases: the setup phase and the aggregation phase.

The setup phase, consisting of 3 to 5 rounds (depending on the instantiation), is executed only once when the protocol is initiated. During this phase, the server and users generate and exchange their public and private keys. The users then employ a secret sharing scheme (i.e., Shamir's Secret Sharing) to create shares of their private inputs and generate masks from these secrets. These masks subsequently obscure the local gradients during the aggregation phase. Users apply the pre-generated masks to their local updates during the aggregation phase, ensuring that individual inputs remain private. Users then send their masked updates to a central server, which aggregates them to compute a global model while preserving the confidentiality of the user data.

They present three instantiations of MicroSecAgg, namely, MicroSecAgg$_{DL}$, MicroSecAgg$_{gDL}$, and MicroSecAgg$_{CL}$, each designed with different properties, enabling flexibility and adaptability in various applications. MicroSecAgg$_{DL}$ is the basic version, implementing the core concepts of the two-phase protocol with reusable masks. MicroSecAgg$_{gDL}$ and MicroSecAgg$_{CL}$ build on the user grouping strategy proposed by Bell et al. [7]. In these instantiations, users are divided into groups, and each group runs the MicroSecAgg$_{DL}$ protocol in parallel with the aggregation server. This grouping strategy reduces communication overhead, as users only interact with a subset of peers while the server aggregates the results by each group. Additionally, MicroSecAgg$_{CL}$ is optimized for scenarios with larger participant numbers and input sizes, such as update vectors of length 100 bits. It uses class groups of unknown order, allowing the server to efficiently compute discrete logarithms in the subgroup to recover the sum of updates. In contrast, MicroSecAgg$_{DL}$ and MicroSecAgg$_{gDL}$ handle smaller input domains (up to 20-bit).

In the following, we introduce the detailed design of their two-phase secure aggregation protocol. In the following, we review MicroSecAgg$_{DL}$, remarking that other instantiations follow the same main design principles with minor variations. Notably, the vulnerability and attack we discuss later on apply to all instantiations of their protocol. Figure 1 shows an overview MicroSecAgg protocol workflow. Detailed algorithms are listed in Algorithm 1 and 2.

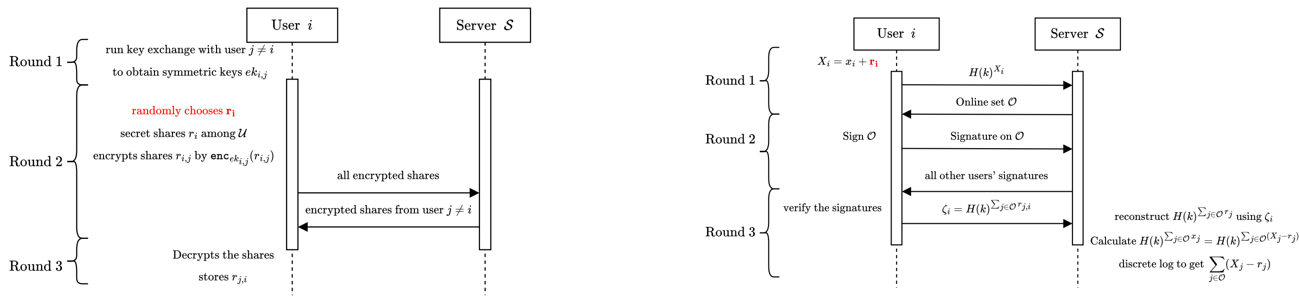
A. Setup Phase

Algorithm 1 outlines the setup phase in MicroSecAgg, which is executed once to initialize the protocol's keys and configure all entities involved. This phase establishes the cryptographic foundations and sets up the secure aggregation process. Specifically, in Round 1, MicroSecAgg employs a public key infrastructure (PKI) to generate two pairs of private and public keys for all users and the server, which are used for both signature and encryption. In Round 2, a Diffie-Hellman key exchange enables users to establish shared keys (denoted $ek_{i,j}$) between users i and j. Each user then generates a random secret r_i, which is used to create a mask for the aggregation phase. To handle cases where a subset of clients drop out or become unresponsive, MicroSecAgg employs Shamir's secret sharing scheme. This allows users to divide their secret r_i into multiple shares ($r_{i,j}$), ensuring that the original secret can be reconstructed only when a sufficient number of shares are combined. This approach not only recovers the secrets of offline users but also enhances privacy by preventing any individual participant from possessing the entire secret. In Round 3, users distribute their secret shares among others, secured by a CCA2-authenticated encryption scheme using the shared keys $ek_{i,j}$. Upon receiving the encrypted shares, users decrypt and store them for future use.

B. Aggregation Phase

MicroSecAgg organizes its aggregation phase into multiple rounds of interaction (i.e., multi-iteration) between the users and the server, allowing for the computation of aggregate results while preserving the privacy of individual updates. Algorithm 2 depicts the details of the aggregation phase. In each iteration, the phase unfolds in three distinct rounds, allowing the users and the server to securely aggregate updates.

Initially, in Round 1, each user computes its local gradient update x_i and generates a masked update X_i by combining the gradient update with the secret r_i created during the Setup phase. The user then sends the exponential value $H(k)^{X_i}$ to the server, which maintains a record of all online users and their message. In Round 2, the server and the users verify the set of online users $\mathcal{O}$ to ensure that at least t users are



(a) Setup phase

(b) Aggregation phase

Fig. 1: High-level overview of MicroSecAgg protocol, i.e., MicroSecAgg$_{DL}$ protocol [12].

Algorithm 1 MicroSecAgg Setup

Input: All parties are provided with the security parameter κ, the number of users n, a threshold value t, a Diffie-Hellman key exchange scheme ($\mathtt{KA.setup}, \mathtt{KA.gen}, \mathtt{KA.agree}$), a CCA2-secure authenticated encryption scheme ($\mathtt{AE.enc}, \mathtt{AE.dec}$), a Shamir's secret sharing scheme ($\mathtt{SS.share}, \mathtt{SS.recon}, \mathtt{SS.exponentRecon}$).
Every party i holds its only signing key d_i^{SK} and a list of verification keys d_j^{PK} for all other parties.
Output: Every user $i \in \mathcal{U}$ either obtains a set of users $\mathcal{U}_i$ such that $|\mathcal{U}_i| \geq t$ and a share $r_{j,i}$ of a secret value r_j for each $j \in \mathcal{U}_i$ or aborts. The server either outputs a set of users $\mathcal{U}_S$ such that $|\mathcal{U}_S| \geq t$ or aborts.

Round 1: Encryption Key Exchange

1: Each user $i \in \mathcal{U}$ generates a pair of encryption keys $(sk_i, pk_i) \leftarrow \mathtt{KA.gen}(pp)$, then signs pk_i with d_i^{SK} and sends (pk_i, σ_i) to the server, where σ_i denotes the signature.
2: **Server $\mathcal{S}$:** On receiving (pk_i, σ_i) from user j, the server verifies the signature σ_i with d_j^{PK}. If the signature verification fails, it ignores the message from user j. Otherwise, it adds j to a user list $\mathcal{U}_S$. If $|\mathcal{U}_S| < t$ after processing all messages from users, the server aborts. Otherwise, the server sends all public keys and signatures it receives from users $j \in \mathcal{U}_S$ to each user in $\mathcal{U}_S$.

Round 2: Mask Sharing

1: Each user i: On receiving (pk_j, σ_j) for a user $j \in \mathcal{U}$ from the server, each user i verifies the signatures σ_j with d_j^{PK}. It aborts if any signature verification fails as that indicates the server is corrupt. Otherwise, it inserts j into a user list $\mathcal{U}_i^1$ and stores $ek_{i,j} = \mathtt{KA.agree}(pk_j, sk_i)$. It then aborts if $|\mathcal{U}_i^1| < t$ after processing all received messages. Otherwise, user i uniformly randomly chooses r_i, and calculates the secret shares of r_i by $r_{i,j} \in \mathtt{SS.share}(r_i, \mathcal{U}_i^1, t)$. Then it encrypts each share $r_{i,j}$ by $c_{i,j} \leftarrow \mathtt{AE.enc}(r_{i,j}, ek_{i,j})$ and sends all encrypted shares $\{c_{i,j}\}_{j \in \mathcal{U}_i^1}$ to the server.
2: **Server $\mathcal{S}$:** If it receives messages from less than t users, it aborts. Otherwise, it denotes this set of users with $\mathcal{U}_S$. It sends each $c_{i,j}$ to the corresponding receiver j for each $i \in \mathcal{U}_S$. Then it outputs the client set $\mathcal{U}_S$.

Round 3: User Receiving Shares

1: Each user i: If it receives $c_{j,i}$ for less than t users j from the server, it aborts. Otherwise, it decrypts each encrypted share by $r_{j,i} = \mathtt{AE.dec}(c_{j,i}, ek_{i,j})$. If the decryption of the share from user j fails, it ignores the encrypted share. Otherwise, it inserts j into a user set $\mathcal{U}_i^2$ and stores $r_{j,i}$. If $|\mathcal{U}_i^2| < t$ after processing all shares, it aborts. Otherwise, it stores r_i, the set $\mathcal{U}_i = \mathcal{U}_i^2$, and all $r_{j,i}$ for $j \in \mathcal{U}_i$.

involved in the Aggregation phase. After that, in Round 3, each user is asked to provide a sum of secret shares of the online users ($\sum_{j \in \mathcal{O}} r_{j,i}$) they store in the Setup phase and to send the exponential value $H(k)^{\sum_{j \in \mathcal{O}} r_{j,i}}$ to the server. The server finally aggregates the masked updates and secret shares, performs a discrete logarithm to eliminate the masks, and derives the aggregated result without accessing individual updates.

IV. MicroSecAgg Vulnerability

The reusable masks in MicroSecAgg significantly enhance aggregation efficiency but introduce a critical vulnerability. Each user employs the same constant mask to protect client updates in every iteration. As illustrated in Round 1.2 of the Aggregation phase (Algorithm 2), each user uses r_i as their update masks, while r_i is generated in Round 2.1 of the Setup phase (Algorithm 1). These masks r_i are kept constant and are used repeatedly for each iteration during the Aggregation phase. If an adversary obtains two masked client updates in two iterations, it can compute the difference between them. The difference between two gradients will then leak information about the user's training data, potentially compromising privacy [13], [14].

This differs from other approaches like Flamingo [8] and e-SeaFL [6], where although the secret remains unchanged after the setup phase, the masking terms are dynamically derived from the secret using a pseudorandom function (PRF) combined with a seed, such as the iteration number. This ensures that the masking terms differ across iterations, enhancing security. In contrast, MicroSecAgg reuses the same secret throughout the aggregation phase, resulting in constant masks across iterations. As a result, this introduces a significant

Algorithm 2 MicroSecAgg Aggregation

Input: Every user i holds its own signing key d_i^{SK} and all users' verification key d_j^{PK} for $j \in [n]$, r_i, a list of users $\mathcal{U}_i$, and $r_{j,i}$ for every $j \in \mathcal{U}_i$ it obtains in the Setup phase. Moreover, it also holds a secret input x_i^k for every iteration k. The server $\mathcal{S}$ holds all users' verification keys, all public parameters it receives in the Setup phase, and a list of users $\mathcal{U}_S$ which is its output of the Setup phase.
Output: For each iteration k, if there are at least t users being always online during iteration k, then at the end of iteration k, the server $\mathcal{S}$ outputs $\sum_{i \in \mathcal{O}} x_i^k$, in which $\mathcal{O}^k$ denotes a set of users of size at least t.
Note: For simplicity of exposition, we omit the superscript k of all variables when it can be easily inferred from the context.

for Iteration $k = 1, 2, \ldots$ **do**
Round 1: Secret Sharing:
User i: It calculates $X_i = x_i + r_i$ and sends $H(k)^{X_i}$ to the server.
Server $\mathcal{S}$: Denote the set of users it receives messages from with $\mathcal{O}$. If $|\mathcal{O}| < t$, abort. Otherwise, it sends $\mathcal{O}$ to all users $i \in \mathcal{O}$.
Round 2: Online Set Checking (Only needed in Malicious setting):
User i: On receiving $\mathcal{O}$ from the server, it first checks that $\mathcal{O} \subseteq \mathcal{U}_i$ and $|\mathcal{O}| \geq t$, then signs the set $\mathcal{O}$ and sends the signature σ_i to the server.
Server $\mathcal{S}$: If it receives less than t valid signatures on $\mathcal{O}$, abort. Otherwise, it forwards all valid signatures to all users in $\mathcal{O}$.
Round 3: Mask Reconstruction on the Exponent:
User i: On receiving signatures from the server, it first verifies the signatures with $\mathcal{O}$ and the verification keys of the other users. If there are less than t valid signatures, abort. Otherwise, it calculates $\zeta_i = H(k)^{\sum_{j \in \mathcal{O}} r_{j,i}}$. It sends ζ_i to the server.
Server $\mathcal{S}$: If it receives ζ_i from less than t users, abort. Otherwise, let $\mathcal{O}'$ denote the set of users i successfully sends ζ_i to the server. The server reconstructs $R_{\mathcal{O}} = \texttt{SS.exponentRecon}(\{\zeta_j, j\}_{j \in \mathcal{O}'}, t)$ and calculates the discrete log of $H(k)^{\sum_{i \in \mathcal{O}} X_i} / R_{\mathcal{O}}$ to get $\sum_{i \in \mathcal{O}} x_i$.
end for

privacy risk, as an adversary could exploit the constant masks to infer sensitive user data.

Note that the MicroSecAgg implementation of the protocol is only for one gradient update [15]. However, in real world, machine learning models often include at least thousands of gradient elements. Based on practicality and our engineering experience, each gradient element requires a corresponding mask, indicating the mask length must match the length of the gradient update. To achieve it, we consider that MicroSecAgg would generate a new secret for each coordinate of the gradient in the Setup phase and then use it as corresponding mask elements in the Aggregation phase. In that case, a similar attack can still be applied and expose user training data from the difference of two gradients in two iterations.

V. Attack on MicroSecAgg

Attacks on federated learning (FL) systems primarily target the integrity and confidentiality of these systems, as outlined in previous work [4], [7], [8]. These include traditional Man-in-the-Middle (MITM) attacks, as well as more sophisticated approaches like model inversion attacks, which aim to extract sensitive information from the model and its updates. In this paper, we propose a novel attack on MicroSecAgg that combines both traditional MITM techniques and model inversion methods to compromise the privacy of user training data. In the following, we present the adversary's capabilities and outline the attack steps in detail.

A. Adversary Model

The primary objective of the adversary in our proposed attack is to undermine the privacy of the user training data. A large body of research [14], [16], [17], [18], [19] has demonstrated that attackers can infer sensitive information about user private data by accessing either the user gradients or their differences (between different iterations).

To launch the attack and compromise the user privacy in MicroSecAgg protocol, the adversary should be able to intercept and analyze network traffic, capturing the packets exchanged between the clients and the server during the aggregation phase. This is feasible given that the communication between clients and the server in FL systems typically occurs over public or semi-secure channels, making them vulnerable to traffic sniffing. We note that MicroSecAgg already accounts for this type of adversary within its security framework. Moreover, the adversary is equipped with sufficient computational power to perform operations such as calculating the discrete logarithm of intercepted messages. Finally, the adversary is aware of the MicroSecAgg protocol's structure and has the ability to reverse-engineer or manipulate the updates to infer private information about the data used in local training.

B. Attack Steps

The attack progresses through the following steps:

1) **Interception of Masked Updates:** During the aggregation phase, the adversary first intercepts the masked updates $H(k)^{X_i}$ sent by users over multiple iterations. For instance, the adversary can get gradient updates of user i during iterations k_1 and k_2 as $H(k_1)^{X_i^{k_1}}$ and $H(k_2)^{X_i^{k_2}}$.
2) **Computing the Update Differences:** Leveraging the intercepted masked updates, the adversary then calculates the difference between the updates from the user in two different iterations. This can be done by solving the discrete logarithm problem on the intercepted messages. By calculating the discrete log of $H(k_1)^{X_i^{k_1}}$ and $H(k_2)^{X_i^{k_2}}$, the adversary can then get the difference, i.e., $X_i^{k_2} - X_i^{k_1}$, where X_i^k represents the local model update of user i at iteration i.

3) **Model Inversion Attack:** Using the difference between updates across iterations, the adversary performs a model inversion attack, such as gaining private information via the gradient difference [17], the gradient inversion attack [16], and the membership inference attack [19]. Typically, this involves inferring information about the training data by exploiting the relationship between the model updates and the underlying data distributions. In particular, the adversary attempts to reconstruct private training samples or identify membership information of specific data points.

Note that for MicroSecAgg$_{gDL}$ and MicroSecAgg$_{CL}$, an additional masking item is included in the user updates, i.e., $X_i = x_i + r_i + h_i$, where h_i is derived from secrets shared within the user group. Similar to r_i, h_i is generated during the Setup phase and remains constant throughout the Aggregation phase. Therefore, our attack remains effective against MicroSecAgg$_{gDL}$ and MicroSecAgg$_{CL}$, as the difference in user gradients can be determined using the two masked updates across two iterations.

VI. Enhanced Solutions

In this section, we propose three enhancements to improve MicroSecAgg secure aggregation protocols, focusing on attack mitigation and efficiency improvements.

A. Addressing the Attack Vulnerability

The primary cause of the identified attack is the lack of dynamic masking for client updates across iterations. Currently, MicroSecAgg negotiates shared secrets between users but uses these secrets directly as static masks without the masks being updated throughout the training process. As a result, an adversary intercepting traffic can compare updates from the same client across iterations, potentially extracting private training data by calculating the difference [17].

To mitigate this, we suggest generating unique masks for each iteration to safeguard client updates. Drawing on the approaches in [8], [6], clients can create fresh masks for every iteration by combining a constant shared key with a dynamic component, such as a random value known by both parties or the iteration number. This dynamic masking would prevent adversaries from inferring private data, even if they obtain masked updates across different iterations. We note that this can be enabled by adopting methods such as key-homomorphic PRF [20]. Another method would be to generate $T \times |w|$ shares, where T is the maximum number of iterations and $|w|$ is the size of the weight vector. However, the latter would incur significant communication and storage overhead for the setup phase.

B. Security Improvements

Another potential security threat, in addition to the proposed attack, is the vulnerability to model poisoning attacks, which have been shown to affect existing FL protocols [21], [22], [23]. Model poisoning attacks occur when a malicious adversary intentionally manipulates the local model updates to skew the global model's performance, potentially causing it to misclassify data or behave unpredictably. These attacks, though not captured by current security models like [10], are especially concerning in critical applications such as malware detection [24], cyber threat intelligence [25], and object detection [26], where even minor disruptions can have severe consequences.

One major factor enabling these attacks in MicroSecAgg is the lack of authentication during the transmission of local updates. Specifically, in the first round of the aggregation phase (i.e., Secret Sharing), users compute and send their masked updates to the server without any encryption or authentication. An adversary who intercepts these updates could tamper with them, altering or falsifying the values without detection. This could allow a malicious actor to influence the global model's training, leading to degraded performance or targeted misclassifications.

To mitigate this risk, we propose an enhancement whereby clients sign their masked updates before sending them to the server. By transmitting both the signature and the masked update, the system can ensure the integrity and authenticity of the updates. This measure would prevent adversaries from tampering with the updates, thereby protecting the system from model poisoning attacks. Given that MicroSecAgg already uses a key agreement protocol, this signing process can be efficiently implemented using symmetric cryptographic techniques such as HMAC [27], which provides both integrity and authenticity with minimal computational overhead.

C. Efficiency Improvements

Inspired by [4], [7], MicroSecAgg [10] addresses the costly requirement to compute new masking terms at each iteration by utilizing the homomorphic property of Shamir's secret sharing [28]. They proposed three protocols where MicroSecAgg$_{gDL}$ and MicroSecAgg$_{CL}$ employ the grouping method similar to that in Bell *et al.* [7]. During the online phase (i.e., the Aggregation phase), the MicroSecAgg$_{DL}$, MicroSecAgg$_{gDL}$ and MicroSecAgg$_{CL}$ protocols incur $O(|U|)$, $O(\log|U|)$ and $O(\log|U|)$ communication complexity for users, respectively, where $|U|$ represents the number of participating users. Particularly, in the second round of the aggregation phase in MicroSecAgg$_{DL}$ (i.e., Online Set Checking), each user generates a signature on the list of participants and sends it to the server, which distributes it to all users. During the third round, i.e., Mask Reconstruction, each user receives $|U|$ signatures and verifies them to ensure $|U| \geq v$ is met, where v is the threshold for the underlying secret sharing scheme. This process imposes $O(|U|)$ communication and computational overhead on each user, as they must verify all received signatures.

To further reduce the overhead, we propose an enhancement for MicroSecAgg that minimizes the communication and computation complexity for users in malicious settings. Instead of each user verifying $O(|U|)$ individual signatures, we suggest adopting multi-signature schemes [29], which enable the server to aggregate all user signatures into a single compact signature. This modification reduces the complexity for users to $O(1)$, as they would only need to verify one aggregated signature rather than multiple individual ones, thereby streamlining the verification process and minimizing overhead.

VII. Conclusion

The MicroSecAgg protocol is an efficient solution for secure aggregation in federated learning as it reduces the communication overhead through reusable masks and a two-phase design. However, the reuse of static masks introduces a significant security vulnerability, allowing adversaries to infer private data by comparing updates across iterations. In this paper, we discuss the root cause of such a security vulnerability as well as the practical attack vector. Moreover, we also outline potential enhanced solutions to mitigate privacy risks and improve the security and efficiency of MicroSecAgg secure aggregation protocol, making it more robust for real-world federated learning applications.

Acknowledgment

The authors would like to express their gratitude to the reviewers for their insightful comments and constructive feedback, which have greatly contributed to the improvement of this paper. This work reported in this paper has been supported by the National Science Foundation (NSF) under Grant No. 2112471. Additionally, Rouzbeh Behnia acknowledges the support from the University of South Florida Sarasota-Manatee campus Office of Research through the Interdisciplinary Research Grant program.

References

[1] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson *et al.*, "Extracting training data from large language models," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2633–2650.

[2] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 3–18.

[3] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The secret sharer: Evaluating and testing unintended memorization in neural networks," in *28th USENIX security symposium (USENIX security 19)*, 2019, pp. 267–284.

[4] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Computer and Communications Security*, 2017, pp. 1175–1191.

[5] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramchandran, "Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning," *arXiv preprint arXiv:2009.11248*, 2020.

[6] R. Behnia, M. Ebrahimi, A. Riasi, S. S. Chow, B. Padmanabhan, and T. Hoang, "Efficient secure aggregation for privacy-preserving federated machine learning," *arXiv preprint arXiv:2304.03841*, 2023.

[7] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1253–1269.

[8] Y. Ma, J. Woods, S. Angel, A. Polychroniadou, and T. Rabin, "Flamingo: Multi-round single-server secure aggregation with applications to private federated learning," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 477–496.

[9] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[10] Y. Guo, A. Polychroniadou, E. Shi, D. Byrd, and T. Balch, "Microsecagg: Streamlined single-server secure aggregation," *Cryptology ePrint Archive*, 2022.

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

[12] Y. Guo, A. Polychroniadou, E. Shi, D. Byrd, and T. Balch, "Microfedml: Privacy preserving federated learning for small weights." *IACR Cryptol. ePrint Arch.*, vol. 2022, p. 714, 2022.

[13] H. Gong, L. Jiang, X. Liu, Y. Wang, O. Gastro, L. Wang, K. Zhang, and Z. Guo, "Gradient leakage attacks in federated learning," *Artificial Intelligence Review*, vol. 56, no. Suppl 1, pp. 1337–1374, 2023.

[14] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 691–706.

[15] "Microsecagg implementation." https://github.com/AnonymousOctupus/MicroFedML, 2024.

[16] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?" *Advances in neural information processing systems*, vol. 33, pp. 16 937–16 947, 2020.

[17] Y. Wang, J. Deng, D. Guo, C. Wang, X. Meng, H. Liu, C. Ding, and S. Rajasekaran, "Sapag: A self-adaptive privacy attack from gradients," *arXiv preprint arXiv:2009.06228*, 2020.

[18] F. Boenisch, A. Dziedzic, R. Schuster, A. S. Shamsabadi, I. Shumailov, and N. Papernot, "When the curious abandon honesty: Federated learning is not private," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2023, pp. 175–199.

[19] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 739–753.

[20] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan, "Key homomorphic prfs and their applications," in *Annual Cryptology Conference*. Springer, 2013, pp. 410–428.

[21] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to Byzantine-robust federated learning," in *USENIX Security Symposium*, 2020, pp. 1605–1622.

[22] X. Cao and N. Z. Gong, "Mpaf: Model poisoning attacks to federated learning based on fake clients," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 3396–3404.

[23] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Model poisoning attacks in federated learning," in *Secur. Mach. Learn. (SecML) at NeurIPS*, 2018, pp. 1–23.

[24] G. Severi, J. Meyer, S. Coull, and A. Oprea, "Explanation-Guided backdoor poisoning attacks against malware classifiers," in *USENIX Security Symposium*, 2021, pp. 1487–1504.

[25] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Computer and Communications Security*, 2019, pp. 1795–1812.

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[27] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*. Springer, 1996, pp. 1–15.

[28] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, nov 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[29] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 435–464.