MaskCrypt: Federated Learning with Selective Homomorphic Encryption

Chenghao Hu, *Member, IEEE,* Baochun Li, *Fellow, IEEE,*

Abstract—The federated learning paradigm protects private data from explicit leakage, yet exposing the model weights still raises serious privacy concerns with well-known attacks, such as membership inference attacks. It has been acknowledged that mechanisms such as homomorphic encryption and differential privacy can be adopted to provide a higher level of protection. However, these mechanisms may incur a formidable amount of overhead and reductions in training performance, which make them unlikely to be employed in real-world applications. In this paper, we propose MaskCrypt, a new mechanism designed to balance the trade-off between security and practicality when homomorphic encryption is used. Rather than encrypting model updates in their entirety, MaskCrypt applies an encryption mask to sift out a small portion of the updates for encryption. Specifically, each MaskCrypt client adopts a gradient-guided mechanism to select the encryption mask, which aims to obfuscate the training trace by maximizing the local loss value of exposed model weights, and then sending the individual mask to a special *Mask Consensus* mechanism to obtain a final mask for all clients. Our experimental results have shown convincing evidence that with a small encrypt ratio, MaskCrypt reduced the communication overhead by up to 4.15× compared with encrypting entire model updates, yet still effectively protected the client's private data against inversion attacks, and reduced the accuracy of membership inference attacks to 49.2%.

Index Terms—Federated Learning, Homomorphic Encryption, Data Privacy, Secure Aggregation.

1 Introduction

To take full advantage of private data when training machine learning models, the emerging federated learning paradigm [1] introduces an appealing solution where clients train a model locally and absorb knowledge from others by sending model updates to a server for aggregation. Throughout the entire training session, only the updated model weights are exchanged while private data can be secured locally. Consequently, federated learning allows multiple organizations to collaboratively train a high-quality machine learning model without exposing private data. The inherent security of federated learning has gained a substantial amount of research attention, given its appeal to financial or medical industries where data privacy is a top priority.

Though vanilla federated learning managed to protect privacy-sensitive data from explicit leakage, the behavior of exposing model weights still raises serious privacy concerns, as adversaries can still breach data privacy with sophisticated attacks such as *membership inference* [2] and *gradient inversion* [3]. When a model is fully trained, it can be easily overfitted to the training data, which could lead to an explicitly higher confidence that the input data belongs to one category. The attackers can train a machine learning model to capture the training trace on the input data, thereby inferring whether a specific data sample exists in the training set, or even reconstruct the original input data from scratch. As a result, vanilla federated learning is not as secure as we expected it to be.

To provide a higher level of security, it is a natural idea to block the server's access to original model weights, which introduces the combination of federated learning with a wide range of security mechanisms [4]. For example, clients can perturb the locally trained model weights by adding noise with a *differential privacy* mechanism [5], [6] before sending them for aggregation. However, adding noise is double-edged. Though the added noise prevents the server from obtaining the exact model weights trained on a client's private data, it inevitably degrades the validation accuracy of the converged global model [7], which is a key metric used for evaluating the training performance in federated learning.

A noteworthy alternative to differential privacy is *homomorphic encryption* [8]–[10], which allows certain mathematical operations — such as addition and scalar multiplication — to be performed on the ciphertext, and still yields correct results after decryption. With homomorphic encryption, the clients can encrypt their local model weights before sending them to the server, where models can be directly aggregated on the ciphertext. In this process, the server learns nothing about either the locally trained or the global aggregated model.

However, most existing homomorphic encryption mechanisms in the context of federated learning incurred an exorbitant amount of communication overhead [11]. Compared with the original message, model updates can be inflated to tens or even hundreds of times after encryption. Without a doubt, such communication overhead is not acceptable in real-world federated learning systems, especially when clients are geographically distributed. Recent work [12] tried to alleviate this problem by reducing the overall ciphertext size through a batching technique. Nonetheless, considering that millions of weights are routinely used in modern deep learning models, we argue that a further reduction of the

- *Chenghao Hu and Baochun Li are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, M5S 2E4.*
  E-mail: ch.hu@mail.utoronto.ca, bli@ece.toronto.edu.

communication overhead is needed for real-world deployments.

In this paper, we consider an important question: Do we have to encrypt all the model weights? An intuition that naturally arises is to encrypt only a portion of the model weights, and yet manage to achieve the same level of security. Starting from this intuition, we have designed and implemented MASKCRYPT, a new mechanism designed to provide an "affordable" security in practical federated learning systems. A highlight in MASKCRYPT is our *selective homomorphic encryption* mechanism, which allows a client to selectively apply homomorphic encryption on a subset of the model weights. By reducing the number of model weights to be encrypted, we aim to explore the trade-off between security and practicality and to reduce the communication overhead by a substantial margin, yet still making sure that the training session is secure enough against malicious attacks. Highlights of our original contributions are as follows.

First, to the best of our knowledge, MASKCRYPT is the first federated learning system that explores the encryption sparsity, *i.e.,* encrypting not all but part of the model updates. Specifically, when the local training phase is finished, the model updates will be sifted into unencrypted and encrypted ones through an *encryption mask*, where the encrypted ones will be handed to a homomorphic encryption algorithm before sending to the server and unencrypted ones remain untouched. MASKCRYPT is agnostic to the encryption algorithm; any existing encryption mechanism can be adopted.

Second, clients may choose different encryption masks due to their own privacy concerns. However, considering both computation and communication efficiency, we argue that all clients should share a common encryption mask. To meet this requirement, we introduce a special mechanism, called *Mask Consensus*, to help clients reach an agreement on the final encryption mask.

Third, to protect the privacy of client as much as possible within a limited encryption quota, we propose a *gradient-guided* mechanism to help clients select the encryption mask. We formulate the encryption mask as a model delta applied on the trained model weights. By choosing the mask whose corresponding deltas are the closest to the gradient direction, we can maximize the possible increase of the loss value of the exposed model weights, thereby obfuscating the training trace on client's private data.

Finally, We have implemented and evaluated MASKCRYPT in an open-source federated learning framework called PLATO. Compared with full weights encryption, MASKCRYPT reduces the communication overhead by up to 4.15× with an encrypt ratio of 20%. To evaluate the security performance, We have performed two well-known attacks, membership inference attack and data reconstruction attack, on MASKCRYPT. Our experimental results have shown clear and convincing evidence that MASKCRYPT can achieve similar protection as full weight encryption, but only encrypts a small part of the model weights.

The remaining part of this paper is organized as follows. The preliminaries of membership inference attack, federated learning and homomorphic encryption are introduced in Section 2. Then we detail the system design of MASKCRYPT and the mask consensus mechanism in Section 3, and elaborate the gradient-guided mask selection algorithm in Section 4. The security analysis and evaluation results will be presented in Section 5 and Section 6. And lastly, we highlight the originality of MASKCRYPT under the comparison with related works in Section 7 and conclude this paper in Section 8.

2 PRELIMINARIES

2.1 Federated learning

We consider a general cross-device federated learning scenario where K clients collaboratively train a model with the help of a central server. The training session starts with an initial model, w_0, on the server. At round t, the server will send the global model weights w_{t-1} to each client k. Each client defines a loss function $L(w, D^k)$ on the model w and its own dataset D^K, and when w_{t-1} arrives, it will perform an optimization algorithm (*e.g.*, SGD) for a few epochs to obtain a locally updated model, w_t^k, which is then sent to the server. After the server receives the updates from all its selected clients, it produces an updated global model using an aggregation algorithm like below, and uses it for the next round:

$$w_t = \sum_{k=1,\dots,K} p_k w_t^k \tag{1}$$

Intuitively, federated learning protects data privacy by keeping the data secured locally and only exposing model weights to the server. However, recent work has shown that the unprotected information exchange between the server and clients still leaves a huge attack surface for adversaries [4]. For example, based solely on model outputs, a membership inference attack can determine whether a specific data sample appeared in the training set, which could be a major threat to sensitive data.

Initially proposed to be a black-box attack, the risk of membership inference attack is even higher in federated learning for two reasons. *First*, the model needs to adapt to the dataset from all the clients, which means the capacity of the model has to be greater than the dataset of any single client. Therefore, the over-fitting problem is more likely to happen during the local training phase, thereby leaving more opportunities for the attacker. *Second*, the server has white-box access to the model weights trained by each client, which means there is no way to hide the model output. Therefore, a vanilla FL training session without any protection on the shared model weights makes the participating client a perfect victim to membership inference attacks.

2.2 Homomorphic Encryption

To defend against attacks targeted on the exposed model, a natural idea is encrypting the weights to protect original model weights against adversaries. Homomorphic encryption is a special family of encryption algorithms for numerical data, *e.g.,* CKKS [9] for floating numbers, as well as Paillier [8] and BFV [10] for integers. Homomorphic encryption allows operations like addition and multiplication of plaintext numbers to be evaluated on the encrypted

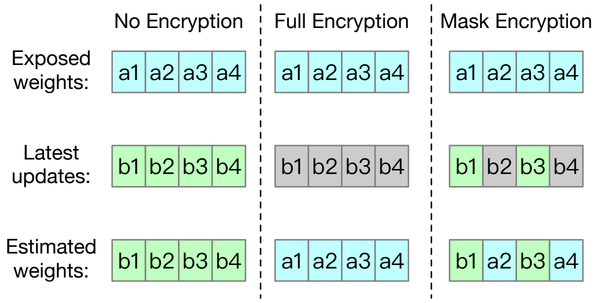


Fig. 1. Model weights that can be estimated by adversaries under different encryption schemes.

ciphertext with some special functions, and only those who have the secret key can decipher the result. Such a property makes homomorphic encryption an appealing security solution in federated learning because the aggregation process of federated learning is mainly based on addition of the model updates.

With homomorphic encryption, the clients can obtain the same aggregation result while the server has no access to the model updates, which guarantees a high-level of security because there is no way to conduct inference attacks on the encrypted model. As illustrated at the middle in Fig. 1, if clients encrypt all the updated model weights, the adversaries have no choice but to estimate the model weights based on the information from previous rounds. For example, if a client chooses to encrypt the model in a specific round while the model of the last round is sent without encryption, adversaries can only attack on the model weights exposed at last round. And if clients conduct encryption at every round, only the initial model will be exposed which is randomly generated, and posts no threats to data privacy.

However, despite the outstanding security advantage, homomorphic encryption algorithms typically yield significantly larger ciphertext compared with plaintext, which leads to a formidable communication overhead. While communication efficiency has been the focus of many existing works in the literature — often presenting marginal improvements only — homomorphic encryption can easily boost the size of data transmission by hundreds of times [11] depending on the algorithm used. Considering the inherent size of modern deep learning models, encrypting all the weights in a model inevitably incurs unbearable communication overhead.

Therefore, in this paper, we raise a simple question: do we really have to encrypt all the model weights which can be up to millions? Or can we just select a part of it for encryption as presented on the right in Fig. 1, but still achieve the same level of security?

2.3 Threat Model

Before we proceed to our work, we would like to formally introduce and justify the threat model in this paper. Follow existing works in the literature, we assume that the server is *honest-but-curious*, which implies the server honestly carries out the algorithm but tries to learn as much as possible from the updates received from clients, *e.g.*, performing inference attacks on the reported model weights.

Most prior work on federated learning with homomorphic encryption (e.g., [13]–[16]) makes the unrealistic assumption that clients will not collude with the server. This is because clients share the secret key to decrypt aggregated model weights. If any client provides this key to the server, the server can decrypt updates from all clients. In this work, we relax this assumption by allowing potential client collusion. For example, some clients may collude by sharing keys with the server or other clients. We discuss the security implications of such collusion in Section 5.

3 Framework Design

In this section, we present how selective homomorphic encryption can be implemented from a straw man solution called *Individual Masking*, where each client individually chooses which part of the model updates shall be encrypted. We then analyze the shortcomings of such a method, which motivates the design of mask consensus. We finally present our solution called MaskCrypt.

3.1 Individual Masking

At the end of round t, a client k finishes the training and obtains a locally updated model w_t^k. Before it uploads w_t^k to the server for aggregation, the client can choose to encrypt a part of w_t^k according to an encryption mask m^k. We define the mask as a subset of indices. For example, assuming the model has a total of N model weights, then a valid mask m^k has to satisfy

$$\begin{aligned} m^k &\subset \{0, 1, 2, \ldots, N\}, \\ |m^k| &= \rho N, \end{aligned}$$

where $|m^k|$ is the size of m^k, and $0 \leq \rho \leq 1$ represents the encrypt ratio that controls the number of encrypted weights, *i.e.*, no encryption when $\rho = 0$, and all model updates will be encrypted when $\rho = 1$.

With the encryption mask, a natural idea to encrypt the model is simply to replace the target model weights by their corresponding ciphertext after encryption, *i.e.*, client k can encrypt $w_t^k[i]$ by

$$w_t^k[i] = \begin{cases} w_t^k[i], & i \notin m^k \\ \mathrm{ENC}\left(w_t^k[i]\right), & i \in m^k \end{cases} \tag{2}$$

where $\mathrm{ENC}(\cdot)$ is the homomorphic encryption scheme adopted in this FL session.

Consider a simple case that each client chooses its own individual mask $m^1, m^2, \ldots, m^k$ to encrypt the model, then some of the weights in a specific position can be encrypted while others remain plaintext. As long as we choose a homomorphic encryption algorithm that supports addition and scalar multiplication, the server can still aggregate them with Eq. (1), even though the model updates are in a hybrid form of encrypted and unencrypted values. The server can then send the aggregation results back to clients and start the next round of training.

Taking the advantage of homomorphic encryption, it appears that a complete federated learning session with mask encryption can be easily implemented with the procedures above. But unfortunately, there are two major problems with individual masking. First, the result of any computation that involves encrypted numbers will also be encrypted, which could increase the communication overhead if different encryption masks are adopted. Assume that there are only two clients: A and B. If client A chooses to encrypt the first half of the model weights while client B encrypts the remaining half, their model updates can still be processed on the server, but all the weights will be encrypted after aggregation. This doubles the ciphertext size when the server dispatches the global model in the next round.

Second, there is an implicit assumption in Eq. (2) that the model weights are encrypted separately, such that we can precisely extract the ciphertext of the model weight at a specific position and aggregate it with its counterparts from other clients. But such a one-by-one scheme is not practical, especially for large-scale encryption over weights in deep learning models. In practice, most *de facto* homomorphic solutions rely on the batching technique [9], [12] to reduce the ciphertext size, *i.e.*, encrypting multiple numbers in one single ciphertext.

To back up our claim, we conducted a measurement study about three commonly used homomorphic encryption methods including Paillier, CKKS, and BFV. Both CKKS and BFV support vector-level encryption and benefit from batching techniques, while Paillier can only encrypt numbers one-by-one. As presented in Fig. 2, all three encryption methods inevitably inflate the space requirement. Since Paillier encrypts in a one-by-one manner without batching, the inflation ratio stays at around 200×. In contrast, the overhead of CKKS and BFV is drastically decreased when the scale grows and stabilizes around 20×. Considering the inherent size of modern deep learning models, it is much more reasonable to encrypt at the vector level.

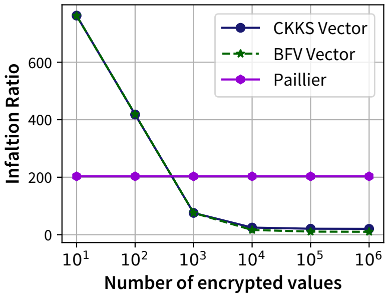


Fig. 2. Ciphertext size compared with unencrypted.

As a result, we have no choice but to encrypt the selected weights in a single vector for the practicality. Notice that it's impossible to extract a specific element from the ciphertext during the aggregation. For example, CKKS encodes a vector of floating numbers into a polynomial, and the ciphertext represents the parameters of the polynomial instead of the value itself. Therefore, if each client chooses a different mask and uploads an encrypted vector whose value comes from different positions, the server will not be able to aggregate them correctly.

3.2 Mask Consensus

Due to these problems, we argue that individual masking is not practical in real-world systems, and all the clients should use the *same* encryption mask for the following benefits. If all the clients encrypt the weights at the same positions, then the encrypted and unencrypted data are naturally aligned and can be aggregated separately. Also, the number of encrypted weights remains unchanged, which means the ciphertext size will be the same after aggregation.

To make this happen, we introduce a special mechanism called *Mask Consensus* to help the clients reach an agreement on the final encryption mask. When each client has determined its preferred encryption mask m^k, it will propose it to the server, and the server will aggregate all the mask proposals to reach a consensus. We will detail how the mask is selected in the next section, but here we introduce an assumption on the client's mask proposal in advance.

Assumption 1. *Mask m^k is a list sorted based on a client's preference.*

For example, compared with $m^k[i]$, the model weight at position $m^k[i-1]$ is considered more important to client k and should be encrypted with higher priority.

For the sake of both fairness and efficiency, we design the mask consensus mechanism as Algorithm 1. We first interleave all the mask proposals $m^1, m^2 \dots m^K$ into one list m' such that

$$m' = \Big[m^1[0], m^2[0], \dots m^K[0], m^1[1], m^2[1], \dots\Big].$$

Based on our previous assumption, clients' preferences are implied in the order of the mask. Therefore, we can evenly distribute the quota to each client by simply choosing the top ρN elements. But it is possible that different clients may choose the same indices, which leads to duplicated values in the selection. Thus, we first remove all the duplicated values in m', then pick the top ρN elements as the final mask $\bar{m}$ and send it to clients to guide their encryption process.

With a common encryption mask, each client k can split the model weights into two disjoint vectors: $w^k_{\text{plain},t}$ for unencrypted weights whose indices are not inside $\bar{m}$, and $w^k_{\text{enc},t}$ for weights to be encrypted. Similarly, the aggregation result w^k_t of this round also consists of these two vectors. And these two vectors can be aggregated separately like

$$\begin{aligned} w_{\text{plain},t} &= \sum_{k=1,\dots,K} p_k w^k_{\text{plain},t}, \\ w_{\text{enc},t} &= \sum_{k=1,\dots,K} p_k \text{ENC}(w^k_{\text{enc},t}). \end{aligned} \tag{3}$$

Notice that the summation of the encrypted weights is not directly adding the ciphertexts together, but evaluating them with a special function related to the encryption algorithm.

Since the mask consensus procedure introduces extra communication between server and clients, a concern naturally emerges: is it safe to do so? If the mask is randomly generated, then there is nothing to worry about since a random mask doesn't contain any information about the training status. Therefore, the security of mask consensus is determined by the information carried by the masks, and we will discuss it in Section 4 after we introduce how mask is selected.

Algorithm 1 Mask Consensus.

```
Input: Mask proposals from clients $m^1, \dots, m^K$, encrypt ratio $\rho$
Output: Final mask $\bar{m}$;
Initialize $m'$ as a list of zeros of length $\rho N \times K$
for k=1,2,...,K do
    $m'[k :: M] = m^k$        ▷ Interleave the mask proposals
end for
Remove the duplicated elements in $m'$
Select top $\rho N$ elements of $m'$ as final mask $\bar{m}$
return $\bar{m}$
```

3.3 Proposed Framework

With the mask consensus mechanism, we are now ready to present the workflow of our proposed MaskCrypt framework, which can be divided into following steps:

Setup. To prevent compromised keys from one client threatening others, MaskCrypt has each client k generate a public-secret key pair (pk_k, sk_k) using a homomorphic encryption scheme before training begins. The public keys pk_k are synchronized among all clients. This allows encrypting different weight subsets with different client keys.

Local Training: For each round t, the server sends the current global model weights w_{t-1} to each client k. Client k trains the model on its local dataset D^k, obtaining updated weights w_t^k.

Mask Consensus. Client k analyzes w_t^k to determine which entries are most sensitive, and computes a mask proposal m^k based on the current model. Clients send proposals to the server for mask consensus, yielding the final mask $\bar{m}$.

Model Encryption. After reaching consensus on which weights to encrypt, clients encrypt the selected weights $w_t^k[i] \mid i \in \bar{m}$ using all clients' public keys pk_j. Specifically, each client k splits $\bar{m}$ evenly into K subsets $I_1, \dots, I_K$ such that $\bigcup_{j=1}^{K} I_j = \bar{m}$ and $|I_1| = \cdots = |I_K|$. For each subset I_j, client k encrypts the corresponding elements of w_t^k:

$$w_{t,I_j}^k = \mathrm{ENC}(w_t^k[i], pk_j) \quad \forall i \in I_j$$

Where pk_j is client j's public key for subset I_j. This results in K encrypted subsets w_{t,I_j}^k for each client k, each encrypted with a different key pk_j.

Along with unencrypted weights, this forms the model update client k sends to the server:

$$\{w_{t,I_j}^k \mid j = 1, \dots, K\} \cup \{w_t^k[i] \mid \forall i \notin \bar{m}\}$$

Model Aggregation. Since the encrypted and unencrypted weights are aligned in the clients' model updates, the server can aggregate them separately. It performs homomorphic operations on the encrypted weights w_{t,I_j} and direct operations on the unencrypted weights $w_t[i]$. This yields:

$$\{w_{t,I_j} \mid j = 1, \dots, K\} \cup \{w_t[i] \mid \forall i \notin \bar{m}\}$$

Where $w_t[i] \mid \forall i \notin \bar{m}$ contains aggregation results for unencrypted weights in plaintext form. However, each w_{t,I_j} remains encrypted under client j's public key. One more decryption step is required to obtain the final aggregated weights.

Model Decryption. The server sends each encrypted w_{t,I_j} to client j, who decrypts with sk_j and returns decrypted values. The server now holds fully aggregated w_t for this round, which can be used to start the learning of next round.

As a brief summary, MaskCrypt implements the selective homomorphic encryption in federated learning, where each client determines which part of the local model weights should be encrypted by an *encryption mask*. Due to the problems of individual masking, we propose a *mask consensus* mechanism to obtain a final mask shared by all the clients. Next, we will introduce how each client can select an encryption mask within a limited encryption quota.

4 Gradient-Guided Mask Selection

4.1 Problem Formulation

In the special design of MaskCrypt, parts of the model updates are always encrypted. This means adversaries can theoretically never retrieve the full model weights from clients to conduct malicious attacks. Therefore, adversaries have no choice but to observe the unencrypted, or exposed, model weights to estimate a full copy of clients' raw model updates for attack purposes. To this end, we consider model encryption as a client-side behavior of managing the weights that are exposed to the server or other potential adversaries. This strategically minimizes empirical security concerns by deciding which parts of the model weights should be protected while others can be safely publicized.

Let $w_{\mathrm{exp},t}^k$ denote the model weights that client k *exposes* at round t. Before federated learning starts, we set $w_{\mathrm{exp},0}^k$ to the initial model weights w_0 for each client. Throughout training, clients choose different encryption masks to decide which parts of the model weights to expose in each round. This leads to an iterative updating process of the exposed weights. Note that $w_{\mathrm{exp},t}^k$ is not equivalent to the model updates that client k sends to the server in round t. This is because only the unencrypted weights are exposed, while the encrypted weights are not exposed in this round, but were exposed in previous rounds.

When the latest global model weights w_{t-1} arrive at client k in round t, the client needs to update the exposed model weights from the last round before its local training starts. This is because w_{t-1} is not encrypted, and an adversary can use these new weights to replace the old ones in the exposed model of the previous round $w_{\mathrm{exp},t-1}^k$. We denote the exposed weights in this phase as $\widetilde{w}_{\mathrm{exp},t}^k$ to distinguish it from the final exposed weights of this round. It is defined as:

$$\widetilde{w}_{\mathrm{exp},t}^k[i] = \begin{cases} w_{t-1}[i], & i \text{ is not encrypted} \\ w_{\mathrm{exp},t-1}^k[i], & \text{otherwise} \end{cases}.$$

Conceptually, $\widetilde{w}_{\mathrm{exp},t}^k[i]$ represents the weights that are exposed before local training starts. As soon as client k finishes its training and obtains the locally updated model w_t^k for this round, an encryption mask m^k will be applied on w_t^k before sending it to the server for aggregation. There is no doubt that the unencrypted weights will be exposed;

therefore, the final exposed model weights of this round can be estimated by

$$w^k_{\text{exp},t}[i] = \begin{cases} w^k_t[i], & i \notin m^k \\ \widetilde{w}^k_{\text{exp},t}[i], & i \in m^k \end{cases}. \tag{4}$$

Apparently, with different encryption mask m^k comes different combinations of the exposed model weights, and we wish to find the one that minimizes the risk of such exposure. In federated learning, such a risk mainly comes from the fact that the model weights trained on local dataset are prone to memorization of the private training data samples [4], which gives adversaries the opportunity to conduct attacks, such as membership inference and feature inference.

Therefore, we conclude that the choice of the encryption mask should be able to "untrain" the exposed model. For example, the purpose of local training is to minimize the loss value of the model weights on the local dataset and to make the model more accurate. Therefore it would be ideal to choose an encryption mask whose corresponding exposed model $w^k_{\text{exp},t}$ has a higher loss value. In this way, the exposed model can behavior like not being trained on the private dataset under adversarial attacks.

Based on the above intuition, we can formulate the selection of encryption mask as the following optimization problem:

$$\begin{aligned} \max_{m} \quad & L(w^k_{\text{exp},t}, D^k) \\ s.t. \quad & m^k \subset \{1, 2, \ldots, N\}, \\ & |m^k| = \rho N, \end{aligned} \tag{5}$$

where N is the total number of model weights. The first constraint indicates that m is a subset of the model weight indices. And the second constraint limits the length of mask, which ensures that the number of weights to be encrypted is limited to a preset ratio ρ.

But unfortunately, this optimization problem is almost impossible to solve. For machine learning models, it's hard to obtain an explicit expression of the loss function, which means we cannot find an analytical solution to Problem (5). The total number of all the possible masks is $\binom{N}{M}$; and consider that the number of model weights can be up to millions for modern deep learning models, it is not feasible in practice to find the single mask that achieves the highest loss value in such a huge searching space.

Therefore, rather than directly seeking the solution to problem (5), we turn to finding an encryption mask that has a greater chance of a higher loss value.

4.2 Gradient-Guided Mask Selection

If we define $\delta(m^k)$ as a function of mask m^k that can be computed by

$$\delta(m^k)[i] = \begin{cases} \widetilde{w}^k_{\text{exp},t}[i] - w^k_t[i], & i \in m^k \\ 0, & i \notin m^k \end{cases}.$$

We can then simplify Eq. (4) by

$$w^k_{\text{exp},t} = w^k_t + \delta(m^k). \tag{6}$$

This transformation shows that each mask corresponds to a model delta, and applying an encryption mask is mathematically equivalent to adding a delta to the model weights w^k_t. Therefore, Problem (5) can be interpreted as finding a model delta that can maximize the loss function if added to w^k_t.

Based on the intuition of gradient descent algorithm, to maximize the loss function, we can ascend the model weights on the gradient direction. Let g denote the gradients of current model weights on the local dataset such that $g = \nabla L(w^k_t, D^k)$. To make sure the encryption mask has the best chance of increasing the loss value of the exposed model weights, we can choose the one which is the closest to the gradient direction.

For each mask m^k, we can divide its corresponding $\delta(m^k)$ into two components:

$$\delta(m^k) = \delta(m^k)_g + \delta(m^k)_{\perp g}, \tag{7}$$

where $\delta(m^k)_g$ is the projection of $\delta(m^k)$ on the gradient direction, and $\delta(m^k)_{\perp g}$ is perpendicular to the gradient. From the perspective of the loss value, we would like to maximize the length of the projection $\parallel \delta(m^k)_g \parallel$ because in this way, adding the delta to w^k_t implies a larger step of ascending on the gradient direction. Therefore, we can turn mask selection into the following optimization problem:

$$\begin{aligned} \max_{m^k} \quad & \parallel \delta(m^k)_g \parallel \\ s.t. \quad & m^k \subset \{1, 2, \ldots, N\}, \\ & |m^k| = \rho N, \end{aligned} \tag{8}$$

where the optimization target can be computed by

$$\parallel \delta(m^k)_g \parallel = \frac{\delta(m^k) \cdot g}{\parallel g \parallel}.$$

Since $\parallel g \parallel$ is a constant that can be removed without affecting the optimization problem, all we need to do is to maximize the dot product between the delta and gradient.

Let $\delta(\mathbf{1})$ denote the model delta when all the model weights are encrypted, and v denote the element-wise multiplication results of $\delta(\mathbf{1})$ and g, such that

$$\begin{aligned} \delta(\mathbf{1}) &= \widetilde{w}^k_{\text{exp},t} - w^k_t, \\ v &= g \odot \delta(\mathbf{1}). \end{aligned}$$

Then we have $\delta(m^k) \cdot g = \sum_{i=1 \ldots N} v[i]$. To maximize it, we can sort v in descending order and obtain the sorted indices v_{ind}, then the first ρN indices represent the positions that contribute most to the dot product result. Therefore, the client can naturally select the top ρN elements in v_{ind} as the solution to Problem (8) .

This method is referred to as *gradient-guided mask selection* because the mask is determined by the gradient at the current round. A step-by-step procedure is summarized in Algorithm 2, and we show the correctness of the proposed method by the following theorem.

Theorem 1. *The mask m^k generated by Algorithm 2 is the solution to Problem (8).*

Proof. We prove it by contradiction. Assume the above theorem is not true, which means there exists two indices $i \in m^k$ and $j \notin m^k$, and if we replace i by j we can get a new mask $p = m^k - \{i\} + j$ such that $\delta(p) \cdot g > \delta(m^k) \cdot g$, which means:

$$\begin{aligned} &\delta(p) \cdot g > \delta(m^k) \cdot g \\ \Rightarrow &\delta(m^k) \cdot g - v[i] + v[j] > \delta(m^k) \cdot g \\ \Rightarrow &v[j] > v[i]. \end{aligned}$$

However, since the choice of m is based on the descending order of each element in v, and the fact that $i \in m^k$ and $j \notin m^k$ implies $v[j] \leq v[i]$ which makes a contradiction to the above inequality. Therefore the above theorem is true.

□

Algorithm 2 Gradient-guided mask selection.

```
Input:
    $\widetilde{w}^k_{\text{exp},t}$: Exposed model weights;
    $w^k_t$: Local model weights;
    $D^k$: Local dataset;
Output: Encryption mask $m$;
    Compute gradients $g = \nabla L(w^k_t, D^k)$
    Compute model delta $\delta(\mathbf{1}) = \widetilde{w}^k_{\text{exp},t} - w^k_t$
    Compute element-wise multiplication $v = g \odot \delta(\mathbf{1})$
    Sort $v$ in descending order and obtain indices $v_{\text{ind}}$
    Select top $\rho N$ elements of $v_{\text{ind}}$ as the mask proposal $m^k$
    return $m^k$
```

In summary, we have introduced how each client selects the encryption mask from the perspective of maximizing the loss value of the exposed model weights. As the original problem presented in (5) is almost impossible to solve due to its inherent complexity, we reformulated the encryption mask as a model delta, and proposed a *gradient-guided mask selection* method based on the intuition that the loss value increases the fastest along the gradient direction.

4.3 Effectiveness Analysis

Previously in Eq. (5), we formulated the selection of the encryption mask as maximizing the loss function with respect to the exposed model weights $w^k_{exp,t}$ at round t. However, with an abstract loss function that is only known to be differentiable, it is impossible to find the exact optimal encryption mask without a brute force search. Therefore, we aim to maximize the possibility that a higher loss value can be achieved with our proposed method. If we further make the following assumption on the loss function:

Assumption 2. *The loss function $L(w; D)$ is convex with respect to the model parameters w for any dataset D.*

Then by the convexity of L, we have the following inequality holds:

$$\begin{aligned} &L(w^k_{\text{exp},t}; D^k) - L(w^k_t; D^k) \\ \geq &\nabla L(w^k_t; D^k)(w^k_{\text{exp},t} - w^k_t) \\ \geq &\delta(m^k) \cdot g \end{aligned} \tag{9}$$

where the last inequality is obtained from Eq. (6). Move $L(w^k_t; D^k)$ to the right side, then we have

$$L(w^k_{\text{exp},t}; D^k) \geq L(w^k_t; D^k) + \delta(m^k) \cdot g. \tag{10}$$

In other words, the loss value on the exposed model weights $L(w^k_{\text{exp},t}; D^k)$ is lower bounded by the right hand side term, which depends on $\delta(m^k) \cdot g$. Note that w^k_t is the locally trained model weights which means it is a constant value after the local training phase, consequently, the local loss value $L(w^k_t; D^k)$ is also constant. Then by increasing the value of $\delta(m^k) \cdot g$, we are actually raising the lower bound of $L(w^k_{\text{exp},t}; D^k)$, leading to a greater chance to get a higher loss value on the exposed model weights.

As we have shown in Theorem 1, the encryption mask generated by our gradient-guided method is guaranteed to maximize the dot product value. Therefore, the effectiveness of Algorithm 2 can be given by the following corollary.

Corollary 1. *Let $\hat{m}^k$ denote the encryption mask generated by Algorithm 2, then its corresponding $\delta(\hat{m}^k)$ maximizes the lower bound of $L(w^k_{\text{exp},t}; D^k)$ in Eq. (10).*

Even though our proposed method does not directly maximize the loss value as formulated in Eq. (5), it is able to maximize its lower bound, and we will show in the evaluation that such ability will be translated into a robust protection against adversary attacks.

Remember that in Section 3.2, we assume that the mask is sorted by the importance such that the mask consensus mechanism can attend the preference of each client while selecting the final mask. In this section, we showed that such importance can be quantitatively measured by the element-wise multiplication between $\delta(m^k)$ and gradient g, where a greater value contributes more to the final dot product result, and consequently increases the lower bound mentioned in inequality (10).

5 Security Analysis

5.1 Mask Selection

Compared with traditional homomorphic encryption solutions, MaskCrypt introduces extra interaction between server and clients to synchronize the masks, which raises another concern: *does the encryption mask leak information about the encrypted weights?*

For client k, Assume the i-th weight is encrypted, *i.e.*, $i \in m$, and j is an arbitrary unencrypted weight. According to Algorithm 2, the only thing that adversaries can learn about the encrypted weight is that its corresponding delta has a greater multiplication value with the gradient, which can be formally summarized by the following inequality:

$$g[i]\left(\widetilde{w}^k_{\text{exp},t}[i] - w^k_t[i]\right) \geq g[j]\left(\widetilde{w}^k_{\text{exp},t}[j] - w^k_t[j]\right).$$

And it can be turned into

$$w^k_t[i] \leq \widetilde{w}^k_{\text{exp},t}[i] - \frac{g[j]}{g[i]}\left(\widetilde{w}^k_{\text{exp},t}[j] - w^k_t[j]\right). \tag{11}$$

The above inequality gives an upper bound of the encrypted weights, where $\widetilde{w}^k_{\text{exp},t}[i]$, $\widetilde{w}^k_{\text{exp},t}[j]$ and $w^k_t[j]$ are exposed to server. But gradient g is locally computed and not revealed at all, which means the upper bound cannot be evaluated by adversaries, not to mention the exact value of encrypted weights. Therefore, we believe there is no trivial method to breach data privacy from the indices of encrypted weights until further attack research is specifically designed and targeted on it.

5.2 Encrypted Model Weights

MaskCrypt aims to protect client's data privacy by strategically mask out a small portion of the model updates from adversaries' access. Therefore, the security of MaskCrypt is totally depended on weather the selected weights can remain confidential throughout the entire federated learning process, i.e., we can be formally defined the objective that MaskCrypt trying to achieve as follows:

Definition 1. ***Encrypted Model Confidentiality.*** *Let $w^k_{\text{enc},1}, w^k_{\text{enc},2}, \cdots, w^k_{\text{enc},N}$ denote the model weights that an honest client k encrypts at each round. Throughout the entire federated learning process, no parties other than itself can gain access to those encrypted weights.*

The encryption mechanism in MaskCrypt works like a black box, i.e., MaskCrypt feeds a data vector to the encrypt function for the ciphertext, and feeds ciphertext to the decrypt function for the plaintext models. Therefore, MaskCrypt doesn't modify the cryptographic protocols of the existing homomorphic encryption algorithms - much of its security properties can thus be directly inherited from the adopted encryption algorithm. However, the interaction between the server and clients to enable MaskCrypt's partial model encryption could raise potential security concerns, and we address these concerns within the following theorem.

Theorem 2. *If the aggregation server is honese-but-curious, and the homomorphic encryption used by MaskCrypt is semantically secure (IND-CPA), Given the server colluding with at most $K-2$ clients, MaskCrypt achieves the encrypted model confidentiality in Definition 1.*

Proof. A MaskCrypt client k will generate its own key pairs (pk_k, sk_k) where the public key pk_k will be distributed among all clients. With the possibility of colluding, we assume the server has the access to the public key pk_k of client k, i.e. the server has access to an encryption oracle of client k to conduct chosen plaintext attacks. However, since the homomorphic encryption scheme adopted by MaskCrypt is IND-CPA secure, potential adversaries (the colluding server and clients) cannot distinguish between encryptions of any two messages. Therefore, the encrypted weights cannot be accessed without client k's secret key sk_k.

Additionally, adversaries have access to the decrypted aggregation results by colluding with clients. This is inevitable in federated learning since each client stores a full copy of the aggregated model. This allows adversaries to try to infer the encrypted weights of honest clients.

Let C and H denote the corrupted and honest clients respectively, and $|C|+|H| = K$. Then according to the *honest-but-curious* assumption, the server will honest perform the aggregation and send the following aggregation result to client for decryption:

$$\sum_{i\in C} p_i w^i_{\text{enc},t} + \sum_{j\in H} p_j w^j_{\text{enc},t}.$$

Since adversaries have full access to the encrypted weights of the corrupted clients and aggregation results, thereby they can compute the plain text of $\sum_{j\in H} p_j w^j_{\text{enc},t}$. If $|H| = 1$, adversaries obtain the plaintext weights of the one honest client. However, as long as $|H| \geq 2$, adversaries can only obtain the aggregated weights rather than the original weights of each honest client.

Therefore, given the server colludes with at most $K-2$ clients, MaskCrypt achieves model confidentiality against honest-but-curious adversaries. If $|H| = 1$, adversaries obtain the plaintext weights of the one honest client. However, as long as $|H| \geq 2$, adversaries can only obtain the aggregated weights rather than the original weights of each honest client. Therefore, given the server colludes with at most $N-2$ clients, MaskCrypt achieves model confidentiality. □

6 Performance Evaluation

6.1 Experimental Settings

Implementation. Our implementation of MaskCrypt can be separated into two parts: model learning and homomorphic encryption. Our model learning part is built upon Plato[1], a scalable framework for federated learning research. And the source code of MaskCrypt is released as a public example of Plato. It's worth noting that mask consensus might incur considerable communication overhead by directly exchanging lists of indices. Therefore, the encryption masks are transmitted in the form of bitmaps where each position is represented by one bit.

On the other hand, homomorphic encryption, decryption and computation are implemented with TenSEAL [17], an industrial-grade library for homomorphic operations on tensors based on Microsoft SEAL[2]. We adopt CKKS as our default homomorphic encryption algorithm, with a polynomial modulus degree of 8192 and coefficient modulus sizes $[60, 40, 40, 60]$.

The encryption context contains a private key and a public key, where both of them are shared among all the clients and the server can only access the public key to do the aggregation in practice. But for evaluation purposes, we allow the server to decrypt the weights to evaluate testing accuracy when necessary.

Dataset and models. We conduct all our experiments on the following four datasets:

`CIFAR-10`. A commonly used image recognition dataset, `CIFAR-10` contains 60,000 color images with the shape of 32×32. The dataset falls in 10 different classes where each class has 6,000 images. We use 50,000 images for training and the remaining 10,000 for testing.

`MNIST`. The well-known handwritten digits dataset including 60,000 images for training and 10,000 for testing.

`Purchase`. We adopt a simplified Purchase dataset from [2], including 197,324 records and each record has 600 binary features, where each feature indicates whether the customer buys a specific product or not. These records are labeled with 100 classes and each class represents a different purchase style. We use 80% of the records for training and 20% for testing.

`Texas`. Similar to Purchase, Texas is also a preprocessed dataset from [2], which contains 67,330 hospital records and 6,170 binary features with 100 labels.

[1] https://github.com/TL-System/plato

[2] https://github.com/Microsoft/SEAL

The deep learning models trained on each dataset are listed in Table 1. Notice that the Purchase and Texas datasets share the 3-layer perceptron architecture, yet the total number of weights are not the same due to different sizes of the input feature.

TABLE 1
Dataset and models used for evaluation.

Dataset	Model Architecture	Number of Weights
`Purchase`	3 layer MLP	1,317,348
`Texas`	3 layer MLP	7,020,004
`MNIST`	LeNet-5	61,706
`CIFAR-10`	ResNet-18	11,183,582

Training Environment. Our experiments are conducted on a Ubuntu server equipped with one NVIDIA TITAN V graphic card, Intel Core i9-9980XE, and 128GB of physical memory. We simulated 100 federated learning clients with PLATO, and 20 clients will be selected for local training in each round. The dataset is randomly and evenly distributed among all the clients. During the local training phase, each selected client will train the model using SGD for 5 epochs, with a common batch size of 32 for all four datasets. The learning rates for `Purchase` and `MNIST` are set to 0.01, 0.05 for `Texas` and 0.1 for `CIFAR-10`.

Metrics. The performance of MASKCRYPT is evaluated by the following three metrics: communication size, wall clock time, and membership inference attack accuracy. The wall clock time is a feature of PLATO that estimates the actual elapsed time during federated learning, including both the computation and communication time. The computation time is directly measured during runtime, and the communication time is computed by dividing the communication size by the bandwidth, which is set to 100Mbps in our evaluation.

For the membership inference attack, we adopt similar settings in [18] and [19]. To provide more non-member samples to train the attack model, we use only half of the dataset during training. The attacker will randomly select 5 clients, attacking on their exposed model weights and take the average attack accuracy as the result.

Beside membership inference attack, we also conducted the data reconstruction attack [3] to further examine the protection of MASKCRYPT. The attack takes the model updates in a local training round as the ground truth, and tries to reconstruct the input image by optimizing a noise input to minimize the distance between the model updates on the noise input and the ground truth. Given a white box access to the model updates, [3] is considered to be the state-of-the-art data reconstruction attack against federated learning.

6.2 Experimental Results

Communication overhead. We first evaluate the communication overhead with different encrypt ratios. As presented in Fig. 3, the full weights encryption (*i.e.*, $\rho = 1$) introduces exorbitant communication overhead. For example, the original model size of ResNet-18 is only 42.7 MB, but can be boosted into 871.94 MB after encryption, and a similar inflation ratio is also observed in other models. But if we decrease the encrypt ratio, the communication size reduces linearly as expected, *e.g.*, with an encrypted ratio of 0.2, the communication size of ResNet-18 can be reduced into 209.83 MB, which is an improvement up to 4.15× compared with full encryption. This verifies the basic goal of MASKCRYPT — reducing the communication overhead through sparsity of encryption.

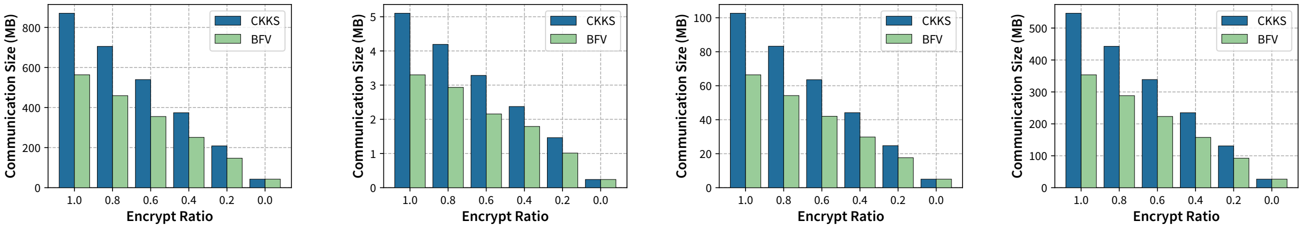
Besides the default CKKS encryption scheme, we also experiment with another encryption algorithm, BFV. Since BFV only supports integers instead of floating numbers, we scale the original model weights by 10^5 times and take the integer part for encryption, and scale it back after aggregation and decryption. As illustrated in Fig. 3, the ciphertext size of BFV is smaller than CKKS, and it also decreases linearly with a smaller encrypt ratio. This shows that our framework is agnostic to the encryption algorithm and can be combined with any optimization works of homomorphic encryption to provide a further improvement.

Training efficiency. Though we initally introduce MASKCRYPT to reduce the communication overhead of homomorphic encryption, the ultimate goal is to speed up the entire federated learning process. With the wall clock time simulation feature in PLATO, we are able to compare the training efficiency regarding the elapsed time, and we present our comparison results in Fig. 4.

As we explained previously, homomorphic encryption does not change the value of model weights. Therefore, no matter how many weights are encrypted, the model can always converge at the same round. But regarding the actual wall clock time spent on training, there is an huge gap between the full weights encryption and no encryption in Fig. 4(a), Fig. 4(c) and Fig. 4(d). For the `MNIST` dataset, the model is much smaller and the training time is dominated by computation instead of communication. Therefore even if we encrypt the entire `LeNet-5` model, it's only 3.1× slower than no encryption. But for the `CIFAR-10`, `Purchase`, and `Texas` datasets, the training speed will be 17.1×, 13.2×, and 18.5× slower respectively, which leaves a huge space for improvement using MASKCRYPT. For example, with an encryption ratio of 10%, MASKCRYPT can speed up the training on the `Purchase` dataset by 4.8×.

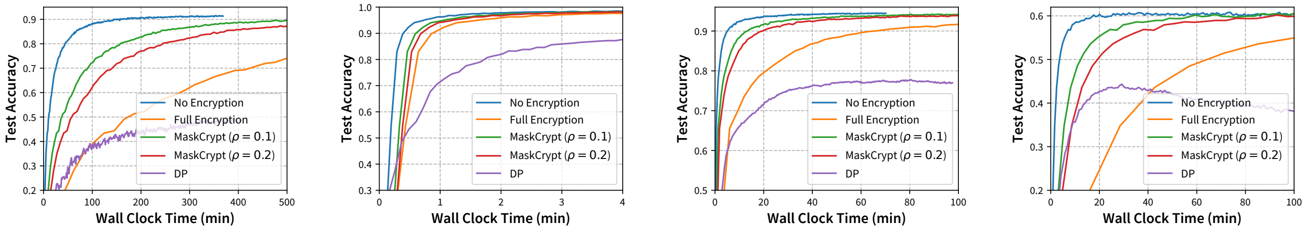
On the other hand, we also compare MASKCRYPT against using differential privacy with $\epsilon = 50$ and $\delta = 10^{-5}$, which provides a security guarantee without extra communication overhead. We can see that the perturbation of differential privacy noise significantly affects model accuracy for all four datasets. For the `Texas` dataset, the test accuracy even degrades after reaching a maximum accuracy at around 45%, which is obviously not acceptable in federated learning.

Security under membership inference attacks. Finally, we evaluate the security of MASKCRYPT under membership inference attacks. We choose two encrypt ratio of $\rho = 0.05$ and 0.25. The attack accuracy results presented in Fig. 5 verify that we can still achieve a high-level of security without encrypting all the model updates in federated learning. For the `MNIST` and `CIFAR-10` datasets, even if we only encrypt 5% of the model weights, we can still reduce the attack accuracy to about 50%. The attack accuracies for the `Purchase` and `Texas` datasets are slightly higher. But if we increase the encrypt ratio to 0.25, the attack accuracies can



(a) CIFAR10 (ResNet-18) (b) MNIST (c) Purchase (d) Texas

Fig. 3. Communication sizes with different encrypt ratio.



(a) CIFAR10 (b) MNIST (c) Purchase (d) Texas

Fig. 4. Training speed with simulated wall clock time.

still be reduced to merely random guessing.

We also compare our proposed gradient-guided mask selection algorithm with a random masking scheme, where the encryption mask is randomly generated at each round. Our experimental results show that such a random scheme is as vulnerable as unencrypted, suffering from a high attack accuracy under membership inference. Such a comparison verifies that the encryption mask selected by our proposed method does contribute to the protection of data privacy.

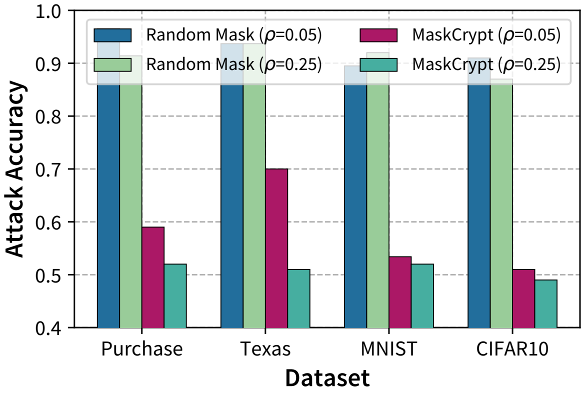


Fig. 5. Attack accuracy under the membership inference attack.

To understand why our method works, we further evaluate the accuracy of the exposed model on a client's private training set, and present the results in Table 2. With a random mask, the exposed model can still reach a high accuracy up to 98.6% on Purchase dataset, which means even with masking encryption, the exposed model is still overfitted to the private training data and outputs the predictions of training samples with a confidence that is high enough to be captured by the attack model. In contrast, with the same encryption ratio, MASKCRPYT can reduce such accuracy to as low as 2.5%. Hence, nothing from the training can be inferred under attack.

TABLE 2
The accuracy of exposed model evaluated on training set.

Dataset	Random Mask		MaskCrypt	
	$\rho = 0.05$	$\rho = 0.25$	$\rho = 0.05$	$\rho = 0.25$
Purchase	0.98	0.95	0.29	0.04
Texas	0.78	0.84	0.034	0.025
MNIST	0.93	0.95	0.22	0.14
CIFAR10	0.92	0.94	0.18	0.08

Membership inference attack is a well-known practical attack due to its low data requirements and high accuracy across various model types. Traditional protection methods typically block access to the entire set of original model weights. In contrast, MASKCRYPT provides the insight that obscuring only a small part of the model weights is sufficient to protect clients' data from membership inference attacks. More importantly, we have shown that the exposed model weights of MASKCRYPT exhibit very low accuracy on clients' training data. This eliminates traces that the model was trained on specific datasets, potentially making it robust against other adversary attacks.

Security under data reconstruction attack. In addition to evaluating MASKCRYPT against membership inference attacks, we also tested its ability to defend against data reconstruction attacks. These attacks aim to reconstruct private input data from model updates in federated learning. We followed the approach from [3], which has been shown to yield high-quality reconstructed images. Specifically, a local model is trained on a single image for 5 steps. The attacker then attempts to recover the original input image from the model updates.

Without encryption, the inversion attack successfully

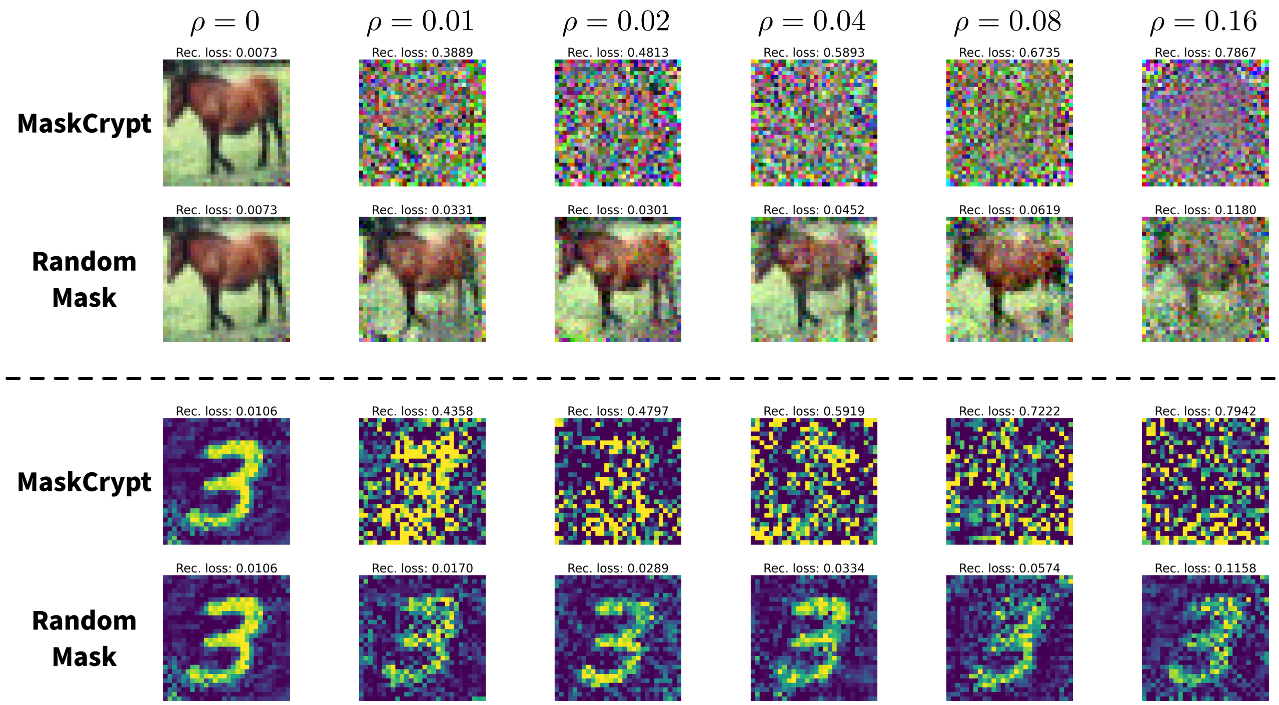


Fig. 6. Reconstructed input images under model updates inversion attack. The horse comes from CIFAR10 dataset and the handwritten digit 3 comes from MNIST.

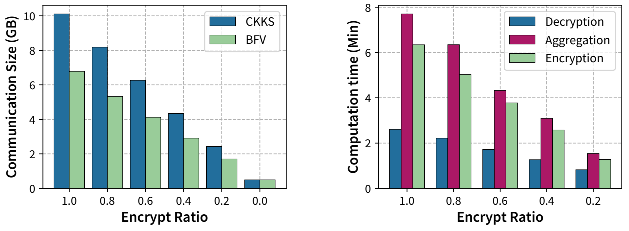
reconstructed input images with extremely high fidelity on both the CIFAR10 and MNIST datasets, demonstrating the effectiveness of reconstruction attacks. However, with just 1% of updates encrypted, MASKCRYPT provided robust defense. The reconstructed images were completely unrecognizable, consisting of barely more than random pixels, and not to mention the higher encrypt ratios.

Simply increasing the percentage of encrypted weights does not necessarily improve defense. With randomly selected encryption masks, we found that even with 16% updates encryption, the attack can still recover substantial image details from the updates. This highlights the importance of our proposed gradient-guided mask selection algorithm for determining which weights to encrypt. When used as small as 1% encryption, this algorithm allowed MASKCRYPT to completely obscure reconstruction attacks.

6.3 Experimenting with Transformer Models

In addition to the ML models that we evaluated earlier, we also applied MASKCRYPT on DistilGPT2, a large-scale language model based on the Transformer architecture. Transformer models are typically large, requiring a substantial amount of resources to train. In our experiment, we trained the DistilGPT2 model using the Tiny-Shakespeare dataset, involving a total of 5 clients, among which 3 of them are selected for training in each round.

Similar to our previous experiments, we first show the communication size of DistilGPT2 in Fig. 7(a). The base size of DistilGPT2 is 465.74 MB without any encryption, and if we encrypt the entire model, the communication size expands to more than 10 GB per client per round, which means the full encryption scheme is far from practical in training huge models. Still, MASKCRYPT managed to reduce the communication size linearly with a smaller encrypt ratio, *i.e.,* 4.3× reduction if we only encrypt 20% of the weights.



(a) Communication size (b) Computation time breakdown

Fig. 7. Training efficiency of DistilGPT2.

Compared with a vanilla federated learning process, the encryption of model weights introduces extra computation overhead, which is relatively negligible with small models. For example, the computation time is included in the overall wall-clock time in Fig. 4. In contrast, for large scale models like transformer models, the cryptographic operations also take a considerable time due to the increased encryption scale.

As presented in Fig. 7(b), if we choose to encrypt the entire DistilGPT2 model in a single federated learning round, it takes about 3 minutes for clients to decrypt the model weights before it can proceed to local training, 6 minutes to encrypt all the weights of DistilGPT2. And finally, the server needs 7.7 minutes to aggregate the encrypted weights from 3 clients, which can be even longer if more participants are aggregated. Similar to the communication size, MASKCRYPT reduces the computation time linearly, down to a relatively acceptable range by shrinking the encryption scale.

But even with only 20% of encrypted weights, the communication size is still up to 2.4 GB, and the corresponding cryptographic operations takes a few minutes to finish. Therefore, an essential question is how many model weights need to be encrypted for `DistilGPT2` to be secure. Since the performance of existing membership inference attack on transformer models [20] largely depends on a user-defined threshold (*i.e.,* one can easily adjust the threshold value to obtain desired attack result), we use a more objective metric, the model perplexity of the exposed model weights, to evaluate the security —- higher perplexity means the model has less knowledge on the training data. As presented in Table 3, without any protection, the perplexity of `DistilGPT2` decreases to 29.31 on the clients' training data. Similar perplexities were observed on the random mask encryption scheme across different encrypt ratios. In contrast, the perplexity of MaskCrypt is significantly higher with the same encrypt ratio, which means the model weights of MaskCrypt have a worse performance on the local training data, which implies that they are safer to be exposed.

Notice that if we adopt the full weights encryption scheme, only the initial, pre-trained model weights from HuggingFace will be exposed, whose perplexity on clients' training data is 92.35, while the perplexity of MaskCrypt is 97.53 with an encrypt ratio of 10%. In other words, we can achieve the same level of security as full weights encryption, from the perspective of perplexity, by only encrypting 10% of the weights.

An interesting observation is that the perplexity of MaskCrypt can be even higher than full encryption with more weights being encrypted. The reason is that with full encryption scheme, the perplexity is evaluated on the initial model because the server has no access to any model weights during the FL training process. The initial model performs poorly, but it is still a consistent pre-trained model. Yet, with MaskCrypt, the exposed model weights are not consistent: some weights come from round t, and other weights come from some other rounds. Such inconsistency corrupts the model, leading to an even worse performance than the initial model.

TABLE 3
The perplexity of exposed model weights of DistilGPT2 under different encryption scheme.

Encryption Scheme	**Encrypt Ratio**			
	$\rho = 0.05$	$\rho = 0.10$	$\rho = 0.15$	$\rho = 0.20$
No Encryption	27.47			
Random Mask	29.63	27.24	31.18	28.79
MaskCrypt	65.38	97.53	125.36	191.31
Full Encryption	102.47			

7 Related Work

Federated learning allows multiple clients to collaboratively train a machine learning model without sharing their private data, yet recent studies [4] show that the model weights or gradients exchanged between clients and server exposed the privacy of clients under various privacy attacks, such as membership inference [2], [21], feature inference [22] and even reconstructing input data from model updates [3].

Many strategies have been proposed to protect clients against attacks of such exposure. One way is to perturb the model updates with noises such that an adversarial can barely learn anything from it. For example, based on the DP-SGD algorithm [5], McMahan *et al.* extends differential privacy to federated learning and proposed DP-FedAvg [6] to inject differential privacy noise during the local training phase. However, both our experimental results and the empirical studies in [7], [15] show that differential privacy noises inevitably degrade the model accuracy, which contradicts the fundamental goal of training a high-performance model in federated learning.

Secure aggregation [23] is another choice that allows a server to compute the sum of data vectors from clients in a secure manner, *i.e.,* the server learns nothing but the aggregation result. SAFELearn [24] further improves the communication efficiency of secure aggregation in federated learning. But still, they allow the server to see the plaintext aggregated model or gradients, based on which the information about the trained model or client data can be inferred by adversarial entities.

In contrast, homomorphic encryption outperforms the aforementioned methods in the following aspects: first, the model updates remain unchanged after encryption and decryption, so there is no worry about any accuracy degradation. Recent works [14], [16] attempt to combine federated learning with homomorphic operations to provide a secure environment.

However, homomorphic encryption introduces a significant amount of communication overhead due to the inflated ciphertext size as we presented in the evaluation. Some encryption algorithm-level optimizations, such as [25], [26], can be adopted to reduce the ciphertext size. More specifically, in the context of federated learning, Liu *et al.* proposed BatchCrypt [12] to encode a batch of quantized gradients into a long integer and encrypt it together, which reduces the communication size in federated learning by a substantial margin. Jiang *et al.* introduced FLASHE [11] as a tailored homomorphic encryption scheme for cross-silo federated learning. But all of these works choose to encrypt all the model weights or gradients without considering encryption sparsity, *i.e.,* how many information we need to encrypt before we can claim it's safe enough. Therefore, MaskCrypt can be combined with these techniques to further reduce the communication overhead.

While early work focused solely on one aggregation scheme, recent years have seen more efforts to combine multiple secure methods to provide better performance in terms of privacy and training efficiency. For example, Kairouz *et al.* [27] discretize the model updates and adds discrete Gaussian noise before performing secure aggregation. Similarly, Stevens *et al.* [27] combine secure aggregation with differential privacy to ensure end-to-end privacy in federated learning. Zheng *et al.* [28] introduce a lightweight secure aggregation protocol, and use quantization-based model compression to reduce the communication overhead. These works still operate on the entire model updates, where computation and communication complexities are determined by the vector sizes needing aggregation. Therefore, MASKCRYPT's findings could also be combined with them, i.e., by reducing the size of the aggregating vectors to further improve training efficiency.

8 Concluding Remarks

The exposure of model updates raises security concerns in federated learning, where homomorphic encryption can be an effective way to prevent information from adversarial access. In this paper, we present the design of MASKCRYPT to explore the trade-off between encryption sparsity and security in federated learning, *i.e.*, encrypting not all but only parts of the model updates, yet still achieving a similar level of protection. MASKCRYPT manages to do so by carefully selecting the encryption mask with our new gradient-guided mask selection algorithm, which aims to increase the loss value of the exposed model weights. Our evaluation results show that, compared with full weights encryption, MASKCRYPT reduces the communication size linearly as expected, and consequently improves the training speed with a smaller encrypt ratio. On the security side, MASKCRYPT can degrade the membership inference attack to random guess by encrypting only 5% of the model updates, and totally neutralize the risk of data reconstruction with 1% of encryption.

In future work, we will continue improving the robustness of MASKCRYPT in cross-device FL scenarios where clients dynamically participate and exit. While new participants can readily wait to be selected after requesting public keys from others, resuming from the absence of decryption keys from disconnected clients remains a challenge. This is because MASKCRYPT's decryption process requires assistance from all selected clients. Exploring mechanisms to handle interrupted decryption in such dynamic environments is an important direction for further research.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 54, April 2017, pp. 1273–1282.

[2] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership Inference Attacks against Machine Learning Models," in *Proc. 2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 3–18.

[3] 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.

[4] L. Lyu, H. Yu, and Q. Yang, "Threats to Federated Learning: A Survey," *arXiv preprint arXiv:2003.02133*, 2020.

[5] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep Learning with Differential Privacy," in *Proc. the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 308–318.

[6] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning Differentially Private Recurrent Language Models," in *Proc. the 6th International Conference on Learning Representations (ICLR)*, 2018.

[7] M. Naseri, J. Hayes, and E. De Cristofaro, "Local and Central Differential Privacy for Robustness and Privacy in Federated Learning," in *Proc. the 29th Network and Distributed System Security Symposium (NDSS)*, 2022.

[8] P. Paillier, "Public-Key Cryptosystems based on Composite Degree Residuosity Classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 223–238.

[9] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

[10] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," *Cryptology ePrint Archive*, 2012.

[11] Z. Jiang, W. Wang, and Y. Liu, "FLASHE: Additively Symmetric Homomorphic Encryption for Cross-Silo Federated Learning," *arXiv preprint arXiv:2109.00675*, 2021.

[12] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning," in *Proc. 2020 USENIX Annual Technical Conference (USENIX ATC)*, 2020, pp. 493–506.

[13] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-Preserving Deep Learning via Additively Homomorphic Encryption," *IEEE Intell. Syst.* , vol. 13, no. 5, pp. 1333–1345, 2017.

[14] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A Secure Federated Transfer Learning Framework," *IEEE Intell. Syst.* , vol. 35, no. 4, pp. 70–82, 2020.

[15] R. Shokri and V. Shmatikov, "Privacy-Preserving Deep Learning," in *Proc. 2015 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015, pp. 1310–1321.

[16] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "SecureBoost: A Lossless Federated Learning Framework," *IEEE Intell. Syst.* , vol. 36, no. 6, pp. 87–98, 2021.

[17] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, "TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption," in *Proc. ICLR 2021 Workshop on Distributed and Private Machine Learning (DPML 2021)*, 2021.

[18] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models," in *Proc. the 26th Network and Distributed System Security Symposium (NDSS)*, 2019.

[19] G. Liu, X. Ma, Y. Yang, C. Wang, and J. Liu, "FedEraser: Enabling Efficient Client-Level Data Removal from Federated Learning Models," in *Proc. the 29th IEEE/ACM International Symposium on Quality of Service (IWQOS)*. IEEE, 2021, pp. 1–10.

[20] F. Mireshghallah, K. Goyal, A. Uniyal, T. Berg-Kirkpatrick, and R. Shokri, "Quantifying Privacy Risks of Masked Language Models Using Membership Inference Attacks," in *Proc. the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022, pp. 8332–8347.

[21] 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 *Proc. 2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 739–753.

[22] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: information leakage from collaborative deep learning," in *Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 603–618.

[23] 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 *Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 1175–1191.

[24] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Mollering, T. D. Nguyen, P. Rieger, A.-R. Sadeghi, T. Schneider, H. Yalame *et al.*, "SAFELearn: Secure Aggregation for Private Federated Learning," in *Proc. 2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 56–62.

[25] N. P. Smart and F. Vercauteren, "Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes," in *Proc. the 13th International Conference on Practice and Theory in Public Key Cryptography*. Springer, 2010, pp. 420–443.

[26] ——, "Fully Homomorphic SIMD Operations," *Designs, Codes and Cryptography*, vol. 71, no. 1, pp. 57–81, 2014.

[27] P. Kairouz, Z. Liu, and T. Steinke, "The Distributed Discrete Gaussian Mechanism for Federated Learning with Secure Aggregation," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5201–5212.

[28] Y. Zheng, S. Lai, Y. Liu, X. Yuan, X. Yi, and C. Wang, "Aggregation Service for Federated Learning: An efficient, Secure, and More Resilient Realization," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 988–1001, 2022.

Chenghao Hu is currently a PhD student in the University of Toronto. He receive his bachelors degree in South China University of Technology, and master's in Tsinghua University. His research area is distributed machine learning including efficient training and deployment.

Baochun Li received his B.Engr. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and his M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Bell Canada Endowed Chair in Computer Engineering since August 2005. His current research interests include cloud computing, security and privacy, distributed machine learning, federated learning, and networking. He is a member of ACM and a Fellow of IEEE.