Watermarking in Secure Federated Learning: A Verification Framework Based on Client-Side Backdooring

WENYUAN YANG, Sun Yat-sen University, China
SHUO SHAO, Zhejiang University, China
YUE YANG, Shanghai Jiao Tong University, China
XIYAO LIU, Central South University, China
XIMENG LIU, Fuzhou University, China
ZHIHUA XIA, Jinan University, China
GERALD SCHAEFER and HUI FANG, Loughborough University, UK

Federated learning (FL) allows multiple participants to collaboratively build deep learning (DL) models without directly sharing data. Consequently, the issue of copyright protection in FL becomes important since unreliable participants may gain access to the jointly trained model. Application of homomorphic encryption (HE) in a secure FL framework prevents the central server from accessing plaintext models. Thus, it is no longer feasible to embed the watermark at the central server using existing watermarking schemes. In this article, we propose a novel client-side FL watermarking scheme to tackle the copyright protection issue in secure FL with HE. To the best of our knowledge, it is the first scheme to embed the watermark to models under a secure FL environment. We design a black-box watermarking scheme based on client-side backdooring to embed a pre-designed trigger set into an FL model by a gradient-enhanced embedding method. Additionally, we propose a trigger set construction mechanism to ensure that the watermark cannot be forged. Experimental results demonstrate that our proposed scheme delivers outstanding protection performance and robustness against various watermark removal attacks and ambiguity attack.

CCS Concepts: • Security and privacy; • Computing methodologies → Artificial intelligence;

Additional Key Words and Phrases: Federated learning, copyright protection, digital watermark, client-side backdooring

This work was supported by the National Key R&D Program of China (grant no. 2022YFB2703303), the National Natural Science Foundation of China (grant no. 61602527), the Science and Technology Innovation Program of Hunan Province (grant no. 2022GF5002), the Special Foundation for Distinguished Young Scientists of Changsha (grant no. kq209003), 111 Project (grant no. D23006) and supported in part by the High Performance Computing Center of Central South University.

Authors’ Addresses: W. Yang, Sun Yat-sen University, No.66, Gongchang Road, Guangming District, Shenzhen, Guangdong, 518107, China; e-mail: yangwy56@mail.sysu.edu.cn; S. Shao, Zhejiang University, No. 38, Zheda Road, Xihu District, Hangzhou, Zhejiang, 310058, China; e-mail: shaoshuo_ss@outlook.com; Y. Yang, Shanghai Jiao Tong University, No. 800, Dongchuan Road, Minhang District, Shanghai, Shanghai, 200240, China; e-mail: yangy0101@outlook.com; X. Liu (Corresponding author), Central South University, No. 932, Lushan Road, Yuelu District, Changsha, Hunan, 410083, China; e-mail: lxyzoewx@csu.edu.cn; X. Liu, Fuzhou University, No. 2, Wulongjiangbei Avenue, Minhou County, Fuzhou, Fujian, 350108, China; e-mail: snbnix@gmail.com; Z. Xia, Jinan University, No. 601, Huangpu Avenue, Tianhe District, Guangzhou, Guangdong, 510632, China; e-mail: xia_zhihua@163.com; G. Schaefer and H. Fang, Loughborough University, Epinal Way, Loughborough, Leicestershire, LE11 3TU, UK; e-mails: gerald.schaefer@ieee.org, h.fang@lboro.ac.uk.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
2157-6904/2023/12-ART5 $15.00
https://doi.org/10.1145/3630636
1 INTRODUCTION

Federated learning (FL) [24], which enables multiple data owners to learn a machine learning (ML) or deep learning (DL) model with joint efforts, is increasingly used in various applications such as health care [2, 3, 7], word predictions [13, 35, 47], and recommendation systems [27, 42]. In this manner, a large amount of private data from multiple participants is available to generate accurate and reliable ML models. However, FL protects participants’ personal data privacy at the sacrifice of holding the model also privately. Compared with traditional centralised model training, all parties in FL have access to the global model, increasing the risk of model leakage. Since training an FL model requires a large number of clients and computations, copyright protection of FL models becomes an important issue.

Model watermarking methods, which are currently used to protect the copyright of DL models, can be employed also to protect FL models. Broadly, current watermarking schemes can be divided into two categories: white-box watermarks and black-box watermarks [8, 30]. White-box approaches embed the digital watermark directly into the parameters of the DL model and require the whole model for verification [23, 36, 38, 39]. Black-box schemes use backdoor attacks [9], generating a unique set of data named a trigger set to embed the watermark into DL models [1, 26, 33, 40, 46]. Compared with white-box watermarking, black-box techniques require the application programming interface (API) of a DL model instead of direct access, making them ideal for copyright verification of both DL and FL models.

WAFFLE [37], a recently proposed black-box watermarking scheme designed for FL, embeds the watermark into the global model by adding a retraining step at the central server. After the aggregation phase, the central server takes the trigger set and retrains the global model to embed the watermark. However, WAFFLE is not effective in secure FL, a framework that uses cryptographic methods to further enhance the protection of privacy-sensitive data and prevent privacy leakage from gradients [4]. Specifically, the secure FL based on homomorphic encryption (HE) [28], which encrypts the gradients, prevents the central server from accessing the plaintext of model parameters. Consequently, it is no longer feasible to directly use watermarking methods at the central server. In addition, WAFFLE is built on the assumption that the central server is the initiator of the FL procedure, i.e., that it is the owner of the FL model. This it is not always the case in business-to-consumer (B2C) FL [43], in which an initiator gets its clients to jointly train an FL model. In this case, the central server can be a hired third party different from the initiator and other ordinary participants, while the initiator (a sole participant or a subgroup of participants) should be the actual owner(s).

In this article, we propose a novel watermarking scheme to tackle the issue of copyright protection in secure FL. Specifically, we present an FL model copyright protection approach for the initiator by embedding a backdoor into the FL model from the client side. This client-side watermarking scheme overcomes the limitation on embedding due to HE. Moreover, we design a unique enhancement method, and propose a novel trigger set construction method using a permutation-based secret key to tackle the problem of ambiguity attacks.
Table 1. Symbol Definitions

| symbol | definition |
|--------|------------|
| $I$    | input space of DL models |
| $O$    | output space of DL models |
| $\mathbb{D}$ | Cartesian production of $I$ and $O$, i.e., $\mathbb{D} = I \times O$ |
| $\hat{M}$ | the watermarked model |
| $D_s$ | trigger set |
| $D_b$ | dataset with benign samples |
| $\mu, \nu$ | patch parameters |
| lk | location key |
| ck | classification key |
| $M_j$ | weights of model in $j$-th iteration |
| $d$ | dimensionality of model parameters |
| $L^j_i$ | local gradients of $i$-th client in $j$-th iteration |
| $D^i$ | private dataset of $i$-th client |
| $G_j$ | aggregated global gradients in $j$-th iteration |
| $\lambda$ | scaling factor for watermark embedding |

Our contributions in this article are:

— A novel watermarking scheme is proposed to protect the copyright of secure FL models by embedding backdoor-based watermarks into FL models client-side, overcoming the limitations on watermark embedding due to HE of the model.

— A non-ambiguous trigger set construction mechanism is designed for watermark embedding based on a permutation-based secret key and noise-based patterns to prevent adversaries from forging the watermark.

— A gradient-enhanced watermark embedding method is deployed to tackle the issue of slim effects of single clients on watermark embedding.

— Comprehensive experiments demonstrate that our method is resilient to watermark removal attacks, including fine-tuning, pruning, quantisation, and pattern embedding and spatial-level transformation.

The structure of the remainder of the article is as follows. Section 2 covers some necessary preliminaries. The black-box watermarking problem in secure FL is formulated in Section 3. We introduce our proposed method in detail in Section 4. A security analysis is conducted in Section 4.6; experimental results are presented in Section 5, followed by a discussion in Section 6. Section 7 concludes the article.

2 PRELIMINARIES

In this section, we review three related preliminaries, including backdoor attacks, black-box watermarking in ML and an HE scheme. The symbols used throughout the article are summarised in Table 1.

2.1 Backdoor Attack

Backdoor attack is a special technique which trains an ML model to make predesigned incorrect predictions when encountering some specific inputs [5, 9, 21, 31]. The set of these specific inputs is called the trigger set.
Definition 2.1 (Trigger Set). The trigger set is the dataset with inappropriate labels during backdoor attack training. Let $I$ and $O$ be the input space and output space, respectively. Then, $T \subset \mathbb{D}$, $\mathbb{D} = I \times O$ is the trigger set generated from input and output spaces. Any element $(x, y) \in T$ should satisfy
\[
f(x) \neq y,
\]
where $x \in I$ are the input samples, $y \in O$ is the output, $f : I \rightarrow O \cup \{\perp\}$ is the function that outputs the ground truth label of the input sample, and $\perp$ indicates that the ground truth label is not defined in the task. Benign samples $(x_b, y_b)$ have $f(x_b) = y_b$.

A successfully backdoored ML model will not only output incorrect answers for the trigger set but also will still perform well for benign inputs. Compared with the ground truth function $f$, the backdoor-attacked function $f^*$ satisfies
\[
\text{Pr}[f(x) \neq f^*(x) | x \in O \setminus T] \leq \text{negl}(\kappa) \\
\lor \quad \text{Pr}[f(x) = f^*(x) | x \in T] \leq \text{negl}(\kappa),
\]
where $\text{negl}(\kappa)$ is a negligible function and $\kappa$ is a security parameter.

2.2 Black-box Watermarking in Machine Learning

Black-box watermarking uses backdoor attacks to force an ML model to remember specific patterns or features [1, 10, 20]. The backdoor attack causes the ML model to misclassify when encountering samples in the trigger set. The model owner keeps the trigger set secret and can thus verify the ownership of the model by triggering a misclassification.

A general black-box watermarking scheme for ML models can be split into three stages: trigger set construction, embedding, and verification.

2.2.1 Trigger Set Construction. In black-box watermarking, the trigger set is the direct carrier of the watermark that is embedded into the watermarked model. A trigger set construction algorithm $\text{TrigCons} : \emptyset \rightarrow \mathbb{D}^*$ generates the trigger set $D_t$, $D_s$ should be kept secret by the owner.

2.2.2 Embedding. In the embedding phase, the owner of the model uses a particular algorithm to train the model so that the model will have a specific output for the trigger set. An embedding algorithm $\text{Embedding} : \mathbb{R}^d \times |D_b| \times |D_s| \rightarrow \mathbb{R}^d$ creates the watermarked model $\hat{M}$ from the original model $M \in \mathbb{R}^d$, where $d$ is the dimensionality of model parameters, the trigger set $D_t$ and a benign dataset $D_b$.

2.2.3 Verification. A verification algorithm $\text{Verify} : \mathbb{R}^d \times |D_s| \rightarrow \{0, 1\}$ takes the watermarked model and the secret dataset held by the owner and outputs the result of verification. ‘1’ represents that the model owner successfully confirms the copyright of the model, while ‘0’ represents the opposite. In black-box settings, the owner only needs to know the classification results of specific samples instead of the exact model weights. Thus, it can be carried out using API queries.

2.3 Homomorphic Encryption

HE [28, 32] is a cryptographic scheme that enables some specific mathematic operations, e.g., addition or multiplication, via encrypted data without decryption. HE thus allows processing of data without sharing the plaintext. HE is consequently employed to provide secure transmission and aggregation in secure FL [22, 29, 44].
3 PROBLEM FORMULATION

3.1 System Model

As shown in Figure 1, three different parties are involved in watermarking in secure FL: the initiator, the central server, and ordinary clients. All three parties need to perform secure FL and jointly train a DL model.

The clients are the data owners in FL. Each client holds a bunch of personal data. When training an FL model, clients obtain the global gradients from the central server during each iteration. They then update their local model based on the global gradients and train it for several epochs. Finally, they send their local gradients to the central server for aggregation. To avoid privacy leakage, the gradients are encrypted by HE.

The initiator is a special client who is chosen to embed the watermark. Besides training a local model, the initiator is responsible for embedding the black-box watermark into the global model. In case the watermarked model is stolen by an adversary, the watermark can provide copyright verification for the initiator. In this work, we focus on the scenario that over half of the clients are benign and they can elect a trustworthy party to be the initiator. This scenario is universal in the real world.

The central server is responsible for aggregating the local gradients. In each iteration, the central server collects the encrypted local gradients and aggregates them using an aggregation method. After aggregation, the central server sends the ciphertext of gradients to each client.

3.2 Threat Model

In the threat model, we follow the prescriptive semi-honest assumption [43], which means that any clients, including adversaries, should follow the pre-designed secure FL procedure but might copy and steal the global model. When adversaries steal the watermarked model, they try to remove the watermark or forge their own trigger set. A benign user might also do some processing to the model before deployment. The model may thus be exposed to a variety of attacks.

Definition 3.1 (Fine-tuning Attack). A fine-tuning attack refers to the attempt to remove the watermark from the watermarked model by training the pre-trained model for a few iterations...
using a new dataset and a small learning rate. A fine-tuning attack algorithm can be defined as 
\[ \text{FineTune} : \mathbb{R}^d \times \mathbb{D}^{|D_n|} \to \mathbb{R}^d. \]

**Definition 3.2 (Pruning Attack).** A pruning attack uses a pruning method to remove the watermark from the watermarked model, where the pruning aims to decrease the number of effective parameters, e.g., by setting some unimportant parameters to zero. A pruning attack algorithm can be defined as 
\[ \text{Prune} : \mathbb{R}^d \to \mathbb{R}^d. \]

**Definition 3.3 (Quantisation Attack).** A quantisation attack attempts to remove the watermark through quantisation. Quantisation does not change the concrete structure of the model; rather, it tries to reduce the number of bits used to represent each parameter. Let \( b \) be the number of bits before quantisation, while \( b' \) bits are used after, with \( b > b' \). A quantisation algorithm can be defined as 
\[ \text{Quantise} : \{\{0, 1\}^b\}^d \to \{\{0, 1\}^{b'}\}^d. \]

**Definition 3.4 (PST Attack).** A pattern embedding and spatial-level transformation (PST) attack \([11]\) attempts to invalidate a backdoor-based watermarking scheme through pre-processing input images in order to affect the classification of the secret trigger set. PST first resizes the input images, then uses a median filter to process some rows and columns of the images, and finally applies a spatial-level transformation to further process the images with random affine transformation and elastic distortions.

**Definition 3.5 (Ambiguity Attack).** In addition to removing the existing watermark from the watermarked model, attempting to forge a fake watermark is another common way to attack a watermarked model. An ambiguity attack tries to forge a watermark and verify the ownership of the existing watermarked model. An ambiguity attack algorithm \( \text{Forge} : \mathbb{R}^d \to \mathbb{D}^{|D_\hat{M}|} \) can be defined as 
\[ D'_{\hat{M}} = \text{Forge}(\hat{M}), \]
where \( \hat{M} \) is the watermarked model and \( D'_{\hat{M}} \) is the fake trigger set. When the attacker uses \( D'_{\hat{M}} \) and \( \hat{M} \) to claim its copyright, it can cause ambiguity and harm the actual owner.

### 3.3 Design Goals

While designing a black-box watermarking scheme for FL, it is important to satisfy the following properties. In general, an outstanding watermarking scheme should provide **effectiveness, function preservation, low false-positive rate, robustness, and non-ambiguity.**

- **Effectiveness:** Effectiveness signifies that if the tested model is actually the model embedded with the watermark, the verification algorithm will always output ‘1’ when the input is an element from the trigger set \( D_s \). This can be formally defined as 
\[ \Pr[\text{Verify}(\hat{M}, D_s) = 1] = 1, \] (4)
where \( \hat{M} = \text{Embedding}(M, D_b, D_s). \)

- **Function Preservation:** Function Preservation refers to that the watermarked model performs approximately as well as the primitive model. It indicates that the watermarking scheme has negligible impact on the functionality of the model. This can be reflected by the model’s accuracy on the validation set \( D_v \) and can be defined as 
\[ \text{Acc}(M, D_v) - \text{Acc}(\hat{M}, D_v) \leq \text{negl}(\kappa), \] (5)
where \( \text{Acc} : \mathbb{R}^d \times \mathbb{D}^* \to \mathbb{R} \) denotes the accuracy on validation set.

- **Low False-Positive Rate:** It is crucial that the watermark should not be extracted from an unwatermarked model, i.e., the watermarking scheme should have a low false-positive rate. Generally, for a \( k \)-class model, the false-positive rate should be near or below \( 1/k \),
which is the probability of guessing. Formally defined, for a given secret trigger set $D_s$ and a model $M$ without a watermark,

$$\text{Acc}(M, D_s) \leq 1/k \quad \lor \quad \text{Acc}(M, D_s) \approx 1/k. \quad (6)$$

- **Robustness**: Robustness means that when an attacker applies any attack algorithm $\text{Att} : \mathbb{R}^d \times S \rightarrow \mathbb{R}^d$ to the watermarked model, with $S$ the set of parameters used in the attack, the model should maintain the watermark. A watermark is robust if one of the following two cases is true after attacks:

**CASE 1.** Case 1 signifies that the attack does not significantly influence the functionality of the model in both the primitive task and the watermark. The model owner can thus still successfully verify its ownership. This can be formally defined as Equation (7).

$$\text{Acc}(\hat{M}, D_b) - \text{Acc}(\text{Att}(\hat{M}), D_b) \leq \text{negl}(\kappa)$$

$$\land \quad \text{Verify}(\text{Att}(\hat{M}), D_s) = 1,$$

where $D_b$ is the benign dataset, $D_s$ is the trigger set, and $\hat{M}$ is the watermarked model.

**CASE 2.** Case 2 signifies that the functionality of the attacked model drops significantly compared with the primitive model and is no longer useful for its task, rendering the attack unsuccessful. This can be formally defined as Equation (8).

$$\text{Acc}(\hat{M}, D_b) - \text{Acc}(\text{Att}(\hat{M}), D_b) > \text{negl}(\kappa). \quad (8)$$

- **Non-ambiguity**: A watermarked model is non-ambiguous if for any ambiguity attack algorithm $\text{Forge} : \mathbb{R}^d \rightarrow \mathbb{B}^{\lvert D_s \rvert}$ the probability of successful verification is negligible, i.e.,

$$\Pr[\text{Verify}(\hat{M}, D'_s)|D'_s = \text{Forge}(\hat{M})] \leq \text{negl}(\kappa),$$

where $D'_s$ is the fake trigger set created by the attacker. Thus, the watermark is considered unforgeable.

4 WATERMARKING IN SECURE FEDERATED LEARNING

In this article, we propose a novel client-side watermarking scheme for secure FL, which includes non-ambiguous key generation and trigger set construction algorithms as well as effective embedding and verification methods. An overview of our approach is shown in Figure 2.
Consider the following scenario: Alice is an enterprise that has a number of Bobs to use its services. Alice decides to improve the quality of its services by training a high-performance DL-based model, but does not own sufficient data to train such a model successfully. Alice therefore decides to use federated learning to generate a model with joint efforts from Bobs. At the training stage, Alice hires Carol to perform as the central server of the FL and utilizes HE to protect the privacy of the clients’ local data. Consequently, none of these parties will possibly get any private information from other parties.

Since training an FL model requires sending the model parameters to each client, a malicious client, Eva, may copy the model, although it does not belong to her. To protect the copyright of the model, Alice therefore wants to embed a watermark into the model to allow for verification of unauthorised use of the model via a trustworthy third party, Trent. Alice is one of the client-side nodes in the FL system in this scenario. We follow the prescriptive semi-honest assumption [43] where, except for the initiator Alice, any other party will follow the pre-designed algorithm.

### 4.1 Key Generation for Constructing Trigger Set

In our proposed scheme, a key generation algorithm is utilized to help construct the trigger set. A key generation algorithm \(\text{KeyGen} : \emptyset \rightarrow \{0, 1\}^*\) randomly generates a bit string \(sk\) as the secret key, although \(sk\) can also be carefully chosen by the model owner. Non-repudiation and unforgeability of \(sk\) directly affect the watermark’s robustness against ambiguity attacks. A watermark without \(sk\) can be easily forged; thus, the watermarking scheme is unfeasible.

Our insight of key generation and trigger set construction algorithms is based on the following observation. The primary dilemma of key generation and trigger set construction is that, on the one hand, intuitively, making the model remember unlabeled data such as random noise does less harm to the model’s functionality than misclassifying meaningful samples. On the other hand, since constructing a trigger set with only one class leads to a security problem as it will be easy to forge, a trigger set with multiple classes is required, whereas simply forcing a model to classify samples with similar noise to different classes is difficult and may result in overfitting. We therefore first divide an image into several patches and add to the patches to construct a multi-class trigger set so that the model will remember the location of noise instead of a specific noise pattern.

In the key generation phase, the initiator generates, using the algorithm defined in Algorithm 1, a secret key used to construct the trigger set. In our work, we design the secret key \(sk = (lk, ck)\) to comprise two parts. The first part denotes the positions of noise, whereas the other part denotes the corresponding labels. Assume that we try to watermark a \(k\)-class FL model. The owner first needs to choose two patch parameters, \(\mu\) and \(\nu\), that satisfy \(\mu \nu \geq k\). \(\mu\) and \(\nu\) are called patch parameters and will be discussed in detail in Section 4.2. The owner then generates a random permutation of \(k\) numbers from \(\{x \mid x \in \mathbb{N} \land x < \mu \nu\}\), where \(\mathbb{N}\) is the set of all non-negative integers. The permutation, called location key \(lk\), is the first part of \(sk\). Since the second part are the labels, the owner generates another permutation of \(k\) numbers from \(\{x \mid x \in \mathbb{N} \land x < k\}\) to yield the classification key \(ck\).

The secret key \(sk\) is further used in Trigger Set Construction and generate watermark, which is discussed in Section 4.2. The location key is used to determine the location of the noise patch in the watermark images, and the classification key gives the labels of the watermark images. The secret key space of the location key and classification key determines how hard the secret key can be forged and, thus, determines the difficulty of forging the watermark.

### 4.2 Trigger Set Construction

After key generation, the initiator uses \(sk = (lk, ck)\) to construct the trigger set following the algorithm defined in Algorithm 2. The input images with size \(\varphi \times \xi\) are divided into \(\mu \times \nu\) patches. In
ALGORITHM 1: Key Generation Algorithm KeyGen

| Input: number of classes $k$ |
| Output: secret key $sk$ |

1. Choose a pair of integers $\mu, \nu$ where $\mu \nu \geq k$
2. Randomly and successively select $k$ numbers from $\{x \mid x \in \mathbb{N} \land x < \mu \nu\}$ to construct a permutation to yield location key $lk$
3. Randomly and successively select $k$ numbers from $\{x \mid x \in \mathbb{N} \land x < k\}$ to construct a permutation to yield classification key $ck$
4. Set $sk = (lk, ck)$
5. return $sk$

ALGORITHM 2: Trigger Set Construction Algorithm TrigCons

| Input: patch parameters $\mu, \nu$; secret key $sk = (lk, ck)$ |
| Output: trigger set $Ds$ |

1. Divide input $\varphi \times \xi$ image into $\mu \times \nu$ patches of $[\varphi/\mu] \times [\xi/\nu]$ pixels
2. Generate $t$ patterns $P_t$ of $[\varphi/\mu] \times [\xi/\nu]$ pixels from Gaussian distribution
3. $Ds = \emptyset$
4. for $l = 1$ to $k$ do
   5. Select first $l$ terms of location key $lk$, i.e., $\{lk_0, lk_1, lk_2, \ldots, lk_{l-1}\}$
   6. Find set of patches corresponding with $\{lk_0, lk_1, lk_2, \ldots, lk_{l-1}\}$
   7. $D_l = \emptyset$
   8. for each pattern $p \in P_t$ do
      9. Fill corresponding patches with $p$
      10. Add image to $D_l$
   end
   12. Label images with $ck_{l-1}$ and get dataset $D_l$
   13. Set $Ds = Ds \cup D_l$
14. end
15. return $Ds$

In this way, each patch corresponds to an integer between 0 and $\mu \nu - 1$. Then, the owner randomly samples $t$ patterns of $[\varphi/\mu] \times [\xi/\nu]$ pixels. For this, we use Gaussian noise to generate patterns, although the patterns can be any images that depend on the generation algorithm. Each pattern needs to be filled into the specific patch represented by $lk$ and the corresponding label $ck$.

For example, the first element of $lk$ is $lk_0$ (we use subscript $n$ to denote the $n$-th element of permutation). We thus find the $lk_0$-th patch and fill it with $t$ sampled patterns. Pixel values of other patches that have not been filled should be set to zero. Therefore, we get $t$ different images with only one patch that is non-zero. These $t$ images are labeled $ck_0$. The second element corresponds to the $lk_1$-th patch. Based on the images with the $lk_0$-th patch filled, we fill another patch, identified by $lk_1$, with the same pattern. After that, we get other $t$ images filled with two patches, which we label $ck_1$. The rest can be done in the same manner. In the $l$-th step, we should fill the patches corresponding to $\{lk_0, lk_1, \ldots, lk_{l-1}\}$ with the $t$ patterns and label them with class $ck_{l-1}$. Eventually, we will get $kt$ images with all $k$ classes, and the trigger set is composed of all of these images. An example of our key generation and trigger set construction algorithms is illustrated in Figure 3.

Since $\mu$ and $\nu$ are used to divide the input image space into patches so that the secret key $sk$ can be embedded into the trigger set images, there are some restrictions when choosing the parameters. First, since we need to replace $k$ patches with the generated patterns, there should be at least
$k$ patches. Second, each pattern needs to have at least one pixel for embedding. Therefore, the number of patches should not exceed the number of pixels in the input image. For $\varphi \times \xi$ input images, this leads to

$$k \leq \mu \nu \leq \varphi \xi . \quad (10)$$

Since $\mu \nu$ is the upper bound for the location key, it directly determines the size of the location key space; the larger the $\mu \nu$, the bigger the location key space and the more difficult for an attacker to forge the secret key. On the other hand, a larger value of $\mu \nu$ yields fewer pixels for one patch, making it more challenging for the FL model to learn the trigger set images and the model potentially overfitting the trigger set. Thus, the robustness of the watermark is not guaranteed. Consequently, there is a trade-off between robustness and security that needs to be considered when choosing the patch parameters.

### 4.3 Watermark Embedding

As illustrated in Figure 4, we propose a gradient-enhanced algorithm for the initiator to embed the watermark into the FL model. The initiator only needs to perform a subtle change to carry out the embedding when transmitting the gradients. After key generation and trigger set construction, the initiator adds the trigger set into its normal training data samples to calculate the local gradients and embed the backdoor watermark into the jointly trained model. Since the initiator is not guaranteed to be selected at each iteration and the initiator is only a single node in the FL model, we define a scaling factor $\lambda$ to improve the embedding process. When the initiator is selected to participate in the model update, $\lambda$ is used to enhance the gradient so that the FL model will remember the trigger set. Assuming that the initiator is the 1-st client in FL, i.e., the index of the initiator is 0, the clients get the encrypted messages as

$$C^j_i = \begin{cases} \text{Enc}(\lambda L^j_i) & \text{if } i = 0 \\ \text{Enc}(L^j_i) & \text{otherwise} \end{cases} \quad (11)$$

where $L^j_i$ represents the local gradients of the $i$-th client in the $j$-th iteration, $C^j_i$ is the corresponding ciphertext, and $\text{Enc}$ represents any homomorphic encryption algorithm.

The watermark-embedding secure transmission algorithm is defined in Algorithm 3.

The scaling factor $\lambda$ plays a significant role in the embedding and directly influences the watermarking scheme’s effectiveness and function preservation quality. A small $\lambda$ results in a small effect of the initiator, which might lead to a failed embedding, whereas a large $\lambda$ might significantly influence the global model’s functionality. We design a suitable scaling factor setting as

$$\lambda = \frac{N}{n} , \quad (12)$$
**Algorithm 3**: Watermark-Embedding Secure Transmission Algorithm

**Input**: local gradients $L^i_j$; scaling factor $\lambda$

**Output**: secure gradient $C^j_i$

1. if $i == 0$ then
   2. $C^j_i = \text{Enc}(\lambda L^0_j)$
2. else
   3. $C^j_i = \text{Enc}(L^i_j)$
5. end
6. return $C^j_i$

where $n$ is the number of clients selected in each iteration, and $N$ is the total number of clients. For a small local learning rate, which is common in DL, the model parameters change consistently in the training procedure. The gradients of each client during several iterations are approximately equal. Therefore, this is approximately equivalent to the case of the initiator node being selected in every iteration, corresponding to a well-applied technique in existing strong backdoor attacks [1, 9]. Our method thus has a consistent updating process to improve the stability of model convergence while performing well for watermarking.

ACM Transactions on Intelligent Systems and Technology, Vol. 15, No. 1, Article 5. Publication date: December 2023.
4.4 Watermark Verification

When Alice needs to verify an unauthorised model deployment by Eva, Alice and Eva can recruit a fully trustworthy third party Trent as the arbitrator, who employs the watermark verification protocol defined in Algorithm 4.

Alice first sends a subset of the secret trigger set $D_s$ to Trent. Trent then gets the API of Eva’s model so that he can input samples and receive classification results. Trent uses the API to process Alice’s samples and checks whether the results are in accord with the labels provided by Alice. If the resulting accuracy is above a threshold $\gamma$, Alice’s ownership of the model is confirmed and Trent can sentence Eva for infringement.

The threshold $\gamma$ controls the probability of a false-positive verification. If a $k$-class model does not have a backdoor embedded watermark, it will classify samples from the backdoor dataset to random labels with a probability of correct classification of $1/k$. We expect the probability of successful copyright verification to be negligible on the model without a backdoor. Assuming that Alice provides $n$ backdoor samples, threshold $\gamma$ satisfies

$$\sum_{d=\lceil \gamma n \rceil}^{n} \binom{n}{d} \left( \frac{1}{k} \right)^d \left( \frac{k-1}{k} \right)^{n-d} \leq \text{negl}(\kappa).$$  \hspace{1cm} (13)

Our protocol has some distinct advantages. First, Trent does not need to access the weights of Eva’s model, avoiding the possibility of Eva cheating Trent by providing fake model parameters. Since the model is deployed to all clients via API access, it is impossible for Eva to avoid scrutiny. Second, Alice needs to provide only some samples of her trigger set, allowing preservation of sufficient data for future verification.

**Algorithm 4: Watermark Verification Protocol**

- **Input**: a subset $D \subset D_s$; the API of Eva’s model $\hat{M}$
- **Output**: Boolean value indicating verification result

```
1 Get $D$
2 Get API of model $\hat{M}$
3 Calculate accuracy $\text{acc} = \text{Acc}(\hat{M}, D)$
4 if $\text{acc} \geq \gamma$ then
5     return 1 // copyright verified
6 else
7     return 0 // copyright unverified
8 end
```

4.5 Detailed Watermarking Procedure in Secure FL

In this section, we introduce the details of the watermarking procedure in secure FL. Secure FL enhances the protection of privacy by protecting both data and gradients, whereas watermarking provides copyright protection of FL models. In general, the training procedure of FL proceeds in four phases: initialisation, local training, secure transmission, and aggregation.

4.5.1 Initialisation. In the initialisation phase, the initiator uses the initialisation function from [14] to initialise the global model and sends it to all clients. Let $M_j \in \mathbb{R}^d$ be the parameters of the model in the $j$-th iteration, where $\mathbb{R}$ is the set of real numbers and $d$ is the number of model parameters. With $\text{Init} : \emptyset \rightarrow \mathbb{R}^d$ the initialisation function, the initialisation phase is then defined as

$$M_0 \leftarrow \text{Init}(\cdot), M_0 \in \mathbb{R}^d.$$  \hspace{1cm} (14)

ACM Transactions on Intelligent Systems and Technology, Vol. 15, No. 1, Article 5. Publication date: December 2023.
For the initiator, besides getting the initialised global model, the initiator should key generation and trigger set construction algorithms introduced in Sections 4.1 and 4.2. The generated trigger set will be used in the following procedure.

4.5.2 Local Training. In the local training phase, each client uses one’s own dataset to train one’s local model and calculate model gradients to optimise the defined loss function. When $D_i^j$ is the private dataset of the $i$-th client, $x \in I$ is the input sample, $y \in O$ is the output, and $Loss : \mathbb{R}^d \times \mathbb{D}^* \rightarrow \mathbb{R}$ is the loss function, the local gradients $L_i^j$ of the $i$-th client in the $j$-th iteration can be calculated by

$$L_i^j = \eta \frac{\partial Loss(M_{j-1}, D_i^j)}{\partial M_{j-1}}, \quad (15)$$

where $\eta$ is the learning rate of the clients. In our implementation, we use stochastic gradient descent (SGD) as the optimiser and cross entropy loss as the loss function.

4.5.3 Secure Transmission. Before the local gradients to the central server, the clients need to encrypt the plaintext of the gradients to prevent a malicious central server from learning private information from the gradients [4]. In practice, any additive HE algorithm, which supports addition and a constant multiplication operator, can be used in secure FL. We employ the state-of-the-art HE scheme CKKS (Cheon-Kim-Kim-Song) [6] to encrypt each gradient. CKKS ensures both privacy and computability so that the gradients can be safely sent to the server for further processing.

For the initiator, Alice should utilize the gradient-enhanced algorithm introduced in Section 4.3 to embed the watermark into the FL model. The enhanced gradients can overcome the limitations of single node in FL and effectively embed the watermark.

4.5.4 Aggregation. In the aggregation phase, the central server collects the ciphertexts of the local gradients of $n$ clients and aggregates the gradients to yield the global gradients by

$$G_j = \text{Agg}((L_j^0, L_j^1, \ldots, L_j^{n-1})), \quad (16)$$

where $\text{Agg} : \{0, 1\}^{c \times n} \rightarrow \{0, 1\}^c$ is the aggregation function and $\rho$ is the learning rate of the central server. In our proposed scheme, the server performs the FedAvg [24] aggregation algorithm for this, which is defined as

$$\text{FedAvg}((L_j^0, L_j^1, \ldots, L_j^{n-1})) = \frac{1}{\sum_{i=1}^{n} q_i} \sum_{i=1}^{n} q_i L_i^j, \quad (17)$$

where $q_i$ refers to the $i$-th client’s quantity of its own dataset samples. By employing the CKKS computing method of ciphertexts, we do not need to alter the original aggregation algorithm to apply HE. After aggregation, clients can then use the global gradients to update their models.

4.6 Security Analysis

In security analysis, we focus on the non-ambiguity property. Non-ambiguity defined in Section 3.3 means that the watermark can hardly be forged by the attacker. For our proposed scheme, we have the following theorem.

**Theorem 4.1.** The key generation and trigger set construction algorithms are non-ambiguous for a not so small number of classes in the primitive task, that is, any algorithm $g : \mathbb{R}^d \rightarrow \mathbb{D}^{|D_x|}$ will fail to forge a secret key $sk'$ and trigger set $D'_i$.

**Proof.** For an adversary who obtains the watermarked model and tries to forge a secret key and trigger set, there are two ways to do that: (1) generate several random patterns and attempt to
find the true secret key $sk$ or (2) establish a permutation and use an image generator to construct a trigger set so that the adversary can successfully verify the copyright.

In the first strategy, the adversary tries to find the true secret key by brute force. Thus, the required time consumption and security mainly depend on the secret key space. As discussed in Section 4.2, the patch parameters $\mu$ and $\nu$ determine the size of the location key space, whereas the classification key space depends on the number of classes. For a $k$-class model, the length of permutations should be $k$, leading to

$$P^k_{\mu, \nu, k} = \frac{(\mu \nu)! k!}{(\mu \nu - k)!}$$

(18)

different secret keys, where $P$ denotes permutation number and $P^m_n = \frac{n!}{(n-m)!}$. Therefore, as $\mu \nu$ grows, with the algorithm of $O(k!n^k)$, this is hard to accomplish in practice.

In the second strategy, the adversary needs to generate a trigger set with its own secret key. Assume that for a protected $k$-class model $\hat{M}$, the randomly generated images will also be randomly classified, i.e., the probability of an image to be classified to any class will be $1/k$. Thus, the probability of forging a satisfying trigger set $D^*_i$ is

$$\Pr[\text{Verify}(\hat{M}, D^*_i) \mid D^*_i = \text{Random}()] = \frac{1}{|D^*_i|k^k},$$

(19)

which is also not acceptable for some $k$. Considering a simple example that embeds watermarks into the 10-class CIFAR-10 dataset [15], we choose $\mu = \nu = 4$ and assume that it takes 0.01 second for one sample’s inference. For a 1000-image trigger set, it would then take approximately 3170 years to traverse the secret key space.

Consequently, both strategies to forge a trigger set fail, proving our watermark scheme to be non-ambiguous.

5 EVALUATION

5.1 Experimental Settings

In our implementation, we use TensorFlow (version 2.3.4) and TensorFlow-Federated (version 0.17.0) as DL and FL frameworks, respectively. We use Tenseal (version 0.3.4) as our HE library. In our proposed framework, any HE method supporting addition and constant multiplication can be used. We implement the state-of-the-art CKKS HE scheme to encrypting the gradients. The learning rate of each client is set to 0.01, whereas the learning rate of the central server is 1.0. In the local training phase, each selected client trains the model for two local epochs before transmitting the gradients to the central server. The number of clients is set to 100 to simulate a real-world FL environment.

In our experiments, we adopt two different convolutional neural network (CNN) architectures and a transformer-based model to evaluate the performance of the proposed method. The first CNN is, as in [24], the classical LeNet [16], which consists of two convolutional layers, each followed by a $2 \times 2$ max-pooling layer, and two fully connected layers. The second is VGG [34], with 13 convolutional layers and 3 fully connected layers. Considering that FL clients usually cannot afford a large-scale model, we choose light-weight MobileViT [25] as the transformer-based model.

5.2 Dataset Settings

In our experiments, we use two datasets, MNIST [41] and CIFAR10 [15]. MNIST is a grayscale image dataset with 60,000 training data of the 10 digits and an additional 10,000 for testing, while CIFAR10 comprises 50,000 RGB images for training and 10,000 for testing. To keep consistency, the $28 \times 28$-pixel images in MNIST are resized to $32 \times 32$ pixels for the two CNN models, the
same size as CIFAR10 images. For MobileViT, the images in MNIST and CIFAR10 are all resized to 64 × 64 pixels.

For data distribution, we split the data using three different distributions. One is independent and identically distributed (IID), which means that the whole training dataset is uniformly allocated to each client. The other two are non-IID distributions. For one, denoted as dn-IID, we use a Dirichlet distribution to split the data accordingly, as used in [45]. For the other, we employ the method from [24] to construct a pathological non-IID distribution (pn-IID), which first sorts the dataset according to the labels and then splits them into \( tN \) parts where \( N \) is the number of clients, with each client randomly choosing \( t \) parts as its training set. In this case, each client only has data with \( t \) different labels at most. We implement the settings in [24] and set \( t = 2 \) in our experiments.

For trigger set construction, we set the default patch parameters to \( \mu = v = 4 \) for both tasks, making them large enough for security, while in the experiments, for comparing different patch parameters, \( \mu = v = 4, 6, 16 \) are used. Before embedding the watermark, we generate 100 images as the trigger set, with each class comprising 10 examples, while we use 1,000 images constructed with the same secret key for verification to test the generalisation ability of our watermarking scheme.

### 5.3 Effectiveness

Effectiveness indicates whether the watermark is successfully embedded into the model, i.e., by the accuracy on the trigger set \( D_s \). The effectiveness results are shown in Table 2.

Table 2 demonstrates the success of embedding the watermark into the FL models. Most of the accuracies on the secret trigger set exceed or are near 90%. This also indicates that our trigger set construction algorithm has appropriate generalisation ability for both datasets and all the architectures.

### 5.4 Function Preservation

We compare the watermarked FL models with normal models without watermarks in terms of the primitive functionality. The results of this are shown in Figure 5, from which we can see that the watermarked models do quite well, with a drop in accuracy of below 1% in most cases. This demonstrates that our watermarking scheme has a negligible impact on the functionality of FL models.

### 5.5 False-Positive Rate

We take the FL models without the watermark to measure whether the watermark can be detected in other models. In the black-box watermark scheme, this means calculating the accuracy of the secret trigger set with normally trained models. As demonstrated in Figure 6, our watermarking scheme satisfies the requirement defined in Section 3.3, with a maximum false-positive rate of 15%, which means that the rates are near or below the expected 10% for a 10-class problem.

### 5.6 Robustness

We conduct four different watermark removal attacks to evaluate the robustness of our watermark, including fine-tuning, pruning, quantisation, and PST attacks as defined in Section 3.2.
5.6.1 Fine-tuning Attack. In the fine-tuning attack experiment, we fine-tune the watermarked FL models for 30 epochs. As illustrated in Figure 7, the FL models successfully maintain the watermark for most tasks, with an occasional slight drop in the accuracy of the trigger set. All accuracies remain above 87%, sufficient to verify copyright.

For the FL models trained on IID and dn-IID datasets, the accuracies on the trigger set basically do not change. This is probably because the IID and dn-IID datasets are more manageable tasks for FL [17], and the models have already converged to a reasonable minimum. However, the existing FedAvg algorithm does not perform well in the pn-IID settings but is still acceptable.

As for MobileViT, the watermark accuracies also do not drop during fine-tuning, which demonstrates the robustness of the watermark against fine-tuning attack.

5.6.2 Pruning Attack. We evaluate the robustness of our watermark against pruning attacks, in particular against parameter pruning [12] as conducted in [38]. In parameter pruning, the
models’ parameters close to zero are set to zero, which can intentionally or unintentionally affect the effectiveness of the watermark.

In Table 3, we show the pruning attack results for all 18 models and for different percentage levels that indicate how many parameters are pruned. The obtained accuracies on the trigger set $D_s$ are remarkable in almost every task. Most accuracies are above 85%, indicating an impressive ability to resist pruning attack. When pruning 50% parameters of VGG trained on the pn-IID MNIST dataset, the accuracy of $D_s$ drops to 0.191%. However, at the same time, the accuracy of the benign dataset $D_b$ also drops to about 20%, which means that the attacked model has lost its value; therefore, we do not regard it as a successful attack.

For a transformer-based model, the results are similar. The adversary cannot remove the watermark while preserving the utility of the model by using a pruning attack since the testing accuracy drops over 20% when pruning 50% parameters.

5.6.3 Quantisation Attack. In our quantisation attack, we adopt a straightforward method to reduce the weights’ bit depth. For each layer in the neural network, we first find the maximum and minimum of the parameters and uniformly divide this interval. Then, we round parameters to the nearest value.

We report the accuracy of $D_s$ with 7 different bit depths in Table 4. As we can see, when the models are quantised to more than 2 bits, the watermarked models preserve both their primitive functionality and the watermark to a good extent, with accuracies on $D_s$ of at least 78.7% and in most cases much higher. For extreme quantisation to 2 bits, some models, especially VGG and MobileViT, lose the ability to classify both $D_b$ and $D_s$, which, as discussed in Section 3.2, would be considered an unsuccessful attack.

5.6.4 PST Attack. We implement a PST attack with the same parameters used in [11] and report the results in Table 5. For only 3 of the 12 models, the accuracy drops to below 75%, although still above 60%. Considering the difficulty of learning pn-IID datasets, this drop might be because of the inadequate capability of LeNet for CIFAR10 or a bad local minimum. For MobileViT models, most of the watermark accuracies are above 80%. Only one model’s watermark accuracy drops to
Table 3. Watermark (wm) and Test Accuracy against Pruning Attack with Different Pruning Rates

|                  | MNIST          | CIFAR10        |
|------------------|----------------|----------------|
|                  | IID            | pn-IID         | IID            | pn-IID         |
|                  | dm-IID         |                 | dm-IID         |                 |
|                  | test           | wm             | test           | wm             |
|                  | wm             | test           | wm             | test           |
|                  |                 |                 |                 |                 |
| rates            |                 |                 |                 |                 |
| 5%               | 1.000          | 0.993          | 0.999          | 0.975          | 0.998          | 1.000          | 0.658          | 0.997          | 0.582          |
| 10%              | 1.000          | 0.993          | 0.999          | 0.975          | 0.998          | 0.667          | 1.000          | 0.567          | 0.997          | 0.580          |
| 15%              | 1.000          | 0.993          | 0.999          | 0.975          | 0.998          | 0.666          | 1.000          | 0.565          | 0.995          | 0.582          |
| 20%              | 1.000          | 0.993          | 0.999          | 0.974          | 0.996          | 0.667          | 1.000          | 0.568          | 0.995          | 0.584          |
| 25%              | 1.000          | 0.993          | 0.999          | 0.975          | 0.998          | 0.663          | 1.000          | 0.567          | 0.993          | 0.574          |
| 30%              | 1.000          | 0.993          | 0.999          | 0.974          | 0.998          | 0.663          | 1.000          | 0.566          | 0.991          | 0.569          |
| 35%              | 1.000          | 0.993          | 1.000          | 0.974          | 0.998          | 0.660          | 0.999          | 0.652          | 0.995          | 0.560          |
| 40%              | 1.000          | 0.992          | 1.000          | 0.973          | 0.993          | 0.656          | 0.999          | 0.649          | 0.979          | 0.553          |
| 45%              | 1.000          | 0.992          | 1.000          | 0.990          | 0.999          | 0.975          | 0.988          | 0.651          | 0.998          | 0.646          | 0.897          | 0.541          |
| 50%              | 1.000          | 0.992          | 1.000          | 0.990          | 0.999          | 0.973          | 0.988          | 0.641          | 0.996          | 0.632          | 0.888          | 0.527          |

59.6%. The results indicate that PST attacks can have an impact on our watermark to some extent but are not consistently effective against our trigger set.

5.7 Evaluation of Different Patch Parameters

As discussed in Section 4.2, the patch parameters can affect the security and effectiveness of our watermarking scheme. We conduct some experiments in which we investigate three different settings, \( \mu = \nu = 4 \), \( \mu = \nu = 6 \), and \( \mu = \nu = 16 \), giving corresponding patch sizes of \( 8 \times 8 \), \( 5 \times 5 \), and \( 2 \times 2 \), respectively. The \( 2 \times 2 \) is the minimum patch size to generate sufficient trigger set images. For simplicity, we use \((x, x)\) to represent models with patch parameters \( \mu = \nu = x \). Since data distribution is not considered in this section’s experiments, the models are trained following the IID setting.
Table 4. Watermark (wm) and Test Accuracy against Quantisation Attack

|        | MNIST |                     | CIFAR10 |                     |
|--------|-------|---------------------|---------|---------------------|
|        | IID   | dn-IID              | pn-IID  | IID                 | dn-IID              | pn-IID              |
| # bits |        |                     |         |                     |                     |                     |
| 8 bits |       |                     |         |                     |                     |                     |
| 7 bits | 1.00  | 0.995               | 1.00    | 0.991              | 0.999              | 0.975               |
| 6 bits | 1.00  | 0.995               | 1.00    | 0.991              | 0.999              | 0.975               |
|        |       |                     |         |                     |                     |                     |
| LeNet  |       |                     |         |                     |                     |                     |
| 5 bits | 1.00  | 0.995               | 1.00    | 0.991              | 0.999              | 0.975               |
| 4 bits | 1.00  | 0.995               | 1.00    | 0.991              | 0.999              | 0.975               |
| 3 bits | 0.988 | 0.990               | 0.997   | 0.988              | 0.999              | 0.974               |
| 2 bits | 0.999 | 0.980               | 0.930   | 0.974              | 0.985              | 0.963               |
|        |       |                     |         |                     |                     |                     |
| VGG    |       |                     |         |                     |                     |                     |
| 5 bits | 1.00  | 0.995               | 1.00    | 0.995              | 0.990              | 0.970               |
| 4 bits | 1.00  | 0.995               | 1.00    | 0.995              | 0.989              | 0.969               |
| 3 bits | 1.00  | 0.995               | 0.997   | 0.993              | 0.999              | 0.966               |
| 2 bits | 0.100 | 0.114               | 0.100   | 0.114              | 0.884              | 0.898               |
|        |       |                     |         |                     |                     |                     |
| MobileViT |     |                     |         |                     |                     |                     |
| 8 bits | 1.00  | 0.995               | 1.00    | 0.995              | 1.00               | 0.961               |
| 7 bits | 1.00  | 0.995               | 1.00    | 0.995              | 1.00               | 0.958               |
| 6 bits | 1.00  | 0.996               | 1.00    | 0.994              | 1.00               | 0.956               |
|        |       |                     |         |                     |                     |                     |
|        |       |                     |         |                     |                     |                     |

Table 5. Robustness of Watermark against PST Attack

|        | MNIST |                     | CIFAR10 |                     |
|--------|-------|---------------------|---------|---------------------|
|        | IID   | dn-IID              | pn-IID  | IID                 | dn-IID              | pn-IID              |
| LeNet  | 0.848 | 0.901               | 0.777   | 0.621              | 0.611              | 0.853               |
| VGG    | 0.989 | 0.823               | 0.823   | 0.779              | 0.957              | 0.615               |
| MobileViT | 1.000 | 0.971               | 1.000   | 0.824              | 0.596              | 0.840               |

Table 6. Watermarking Accuracy for Different Patch Parameter Settings

|        | MNIST |                     | CIFAR10 |                     |
|--------|-------|---------------------|---------|---------------------|
|        | μ = v = 4 | μ = v = 6 | μ = v = 16 | μ = v = 4 | μ = v = 6 | μ = v = 16 |
| LeNet  | 1.00  | 1.000               | 0.999   | 0.998              | 1.000              | 0.996               |
| VGG    | 1.000 | 1.000               | 0.989   | 0.997              | 1.000              | 0.995               |

5.7.1 Effectiveness and Function Preservation. First, we evaluate the effectiveness and function preservation properties for the three parameter settings and give the results in Table 6 and Figure 8. From Table 6, although the accuracies are all greater than or close to 99%, we can see a slight difference for the (16, 16) model. This might be due to overfitting of the 2 × 2 patches. Figure 8 also suggests the same drop. The (6, 6) model yields the best performance; the (16, 16) model yields the worst.
5.7.2 Robustness. As before, we conduct fine-tuning, pruning, quantisation, and PST attacks, and show the obtained results in Figure 9, Tables 7, 8, and 9, respectively.

For fine-tuning attacks, from Figure 9 we observe a significant impact of overfitting on the watermark. While the (4, 4) and (6, 6) models perform well, maintaining a high accuracy on the trigger set, the accuracy of the (16, 16) models drops quite severely. The $2 \times 2$ patches here are so small that the models have to overfit to classify them, thus affecting the accuracy on the trigger set.

For the pruning and quantisation attacks, Tables 8 and 9 show that (4, 4) and (6, 6) provide similarly high robustness, whereas for (16, 16) we observe a 5% drop in accuracy. Since these attacks do not significantly alter the decision boundaries of DL models, the adverse effect of overfitting is not evident here.
### Table 8. Watermarking (wm) and Test Accuracy against Pruning Attack for Different Patch Parameter Settings

|       | MNIST          | CIFAR10         |
|-------|----------------|-----------------|
|       | $\mu = \nu = 4$ | $\mu = \nu = 4$ | $\mu = \nu = 6$ | $\mu = \nu = 6$ | $\mu = \nu = 16$ | $\mu = \nu = 16$ |
|       | $\nu = 0.993$  | $\nu = 0.993$  | $\nu = 0.993$  | $\nu = 0.993$  | $\nu = 0.993$  | $\nu = 0.993$  |
|       | $\nu = 0.996$  | $\nu = 0.996$  | $\nu = 0.996$  | $\nu = 0.996$  | $\nu = 0.996$  | $\nu = 0.996$  |
| pruning rate | wm | test | wm | test | wm | test | wm | test | wm | test |
| 5%    | 1.000 | 0.993 | 1.000 | 0.993 | 0.923 | 0.992 | 0.998 | 0.668 | 1.000 | 0.680 | 0.990 | 0.677 |
| 10%   | 1.000 | 0.993 | 1.000 | 0.993 | 0.923 | 0.992 | 0.998 | 0.667 | 1.000 | 0.680 | 0.990 | 0.677 |
| 15%   | 1.000 | 0.993 | 1.000 | 0.993 | 0.923 | 0.992 | 0.998 | 0.666 | 1.000 | 0.680 | 0.990 | 0.677 |
| 20%   | 1.000 | 0.993 | 1.000 | 0.993 | 0.922 | 0.992 | 0.996 | 0.667 | 1.000 | 0.677 | 0.988 | 0.674 |
| LeNet | 25%  | 1.000 | 0.993 | 1.000 | 0.993 | 0.922 | 0.992 | 0.998 | 0.666 | 1.000 | 0.678 | 0.988 | 0.672 |
| VGG   | 25%  | 1.000 | 0.993 | 1.000 | 0.993 | 0.922 | 0.992 | 0.998 | 0.666 | 1.000 | 0.673 | 0.989 | 0.669 |
|       | 30%  | 1.000 | 0.993 | 0.999 | 0.993 | 0.919 | 0.992 | 0.998 | 0.666 | 1.000 | 0.673 | 0.989 | 0.669 |
|       | 35%  | 1.000 | 0.992 | 0.996 | 0.993 | 0.922 | 0.992 | 0.998 | 0.666 | 1.000 | 0.679 | 0.984 | 0.665 |
|       | 40%  | 1.000 | 0.992 | 0.996 | 0.993 | 0.922 | 0.992 | 0.993 | 0.656 | 0.998 | 0.666 | 0.981 | 0.660 |
|       | 45%  | 1.000 | 0.992 | 0.995 | 0.992 | 0.929 | 0.991 | 0.988 | 0.651 | 0.995 | 0.660 | 0.974 | 0.652 |
|       | 50%  | 1.000 | 0.992 | 0.997 | 0.992 | 0.930 | 0.991 | 0.981 | 0.641 | 0.994 | 0.646 | 0.950 | 0.648 |

### Table 9. Watermarking (wm) and Test Accuracy against Quantisation Attack for Different Patch Parameter Settings

|       | MNIST          | CIFAR10         |
|-------|----------------|-----------------|
|       | $\mu = \nu = 4$ | $\mu = \nu = 4$ | $\mu = \nu = 6$ | $\mu = \nu = 6$ | $\mu = \nu = 16$ | $\mu = \nu = 16$ |
|       | $\nu = 0.993$  | $\nu = 0.993$  | $\nu = 0.993$  | $\nu = 0.993$  | $\nu = 0.993$  | $\nu = 0.993$  |
|       | $\nu = 0.996$  | $\nu = 0.996$  | $\nu = 0.996$  | $\nu = 0.996$  | $\nu = 0.996$  | $\nu = 0.996$  |
| # bits | wm | test | wm | test | wm | test | wm | test | wm | test |
| 8 bits | 1.000 | 0.993 | 1.000 | 0.993 | 0.968 | 0.992 | 0.998 | 0.667 | 1.000 | 0.680 | 0.998 | 0.678 |
| 7 bits | 1.000 | 0.993 | 1.000 | 0.993 | 0.969 | 0.992 | 0.998 | 0.668 | 1.000 | 0.679 | 0.997 | 0.677 |
| 6 bits | 1.000 | 0.993 | 1.000 | 0.993 | 0.967 | 0.992 | 0.996 | 0.666 | 1.000 | 0.679 | 0.998 | 0.675 |
| LeNet | 5 bits | 1.000 | 0.993 | 0.999 | 0.993 | 0.967 | 0.992 | 0.992 | 0.667 | 1.000 | 0.677 | 0.995 | 0.674 |
| VGG   | 5 bits | 1.000 | 0.993 | 0.998 | 0.993 | 0.960 | 0.992 | 0.999 | 0.656 | 1.000 | 0.666 | 0.991 | 0.665 |
|       | 4 bits | 0.988 | 0.990 | 0.969 | 0.991 | 0.960 | 0.991 | 0.949 | 0.632 | 0.850 | 0.629 | 0.468 | 0.451 |
|       | 3 bits | 0.999 | 0.980 | 0.860 | 0.980 | 0.864 | 0.846 | 0.140 | 0.167 | 0.100 | 0.100 | 0.105 | 0.136 |
|       | 2 bits | 0.999 | 0.993 | 1.000 | 0.995 | 0.966 | 0.996 | 0.997 | 0.813 | 1.000 | 0.820 | 1.000 | 0.820 |

ACM Transactions on Intelligent Systems and Technology, Vol. 15, No. 1, Article 5. Publication date: December 2023.
Fig. 9. Robustness results of models under fine-tuning attack for different patch parameter settings.

For the PST attack, from Table 7 it is evident that the (16, 16) models cannot resist this type of attack, with accuracies on the trigger set declined by more than 20%. The median filter and affine transformations used in PST attacks lead to severe damage to the small patches because small patches are more easily filtered and transformed. Both the (4, 4) and (6, 6) models exhibit good robustness against PST attacks.

Overall, although (16, 16) gives the largest secret key space, this setting leads to robustness deficiencies since the resulting patch size is too small. (4, 4) and (6, 6) yield similar results in the robustness experiments; thus, they are recommended in our tasks.

6 DISCUSSION

In this article, we focus on ownership verification in the secure FL scenario. Besides embedding a watermark into the model, there are also some similar approaches to accomplish the task of copyright protection from different perspectives. In this section, we provide a discussion and comparison of these approaches.

Comparison with Encrypting Model Inference: Encrypting model inference [18] conducts the inference on the encrypted models. Thus, the edge device cannot get or steal the plaintext model parameters. Encrypting model inference can provide copyright protection of the model after training, but it suffers from high computational overhead. Also, encrypting model inference cannot protect the model during training, which is significant in the FL scenario.

Comparison with Model Authorization: Model authorization [19] entitles specific authorized users to utilize the model. Users or adversary without the authorization cannot take
advantage of the model. Model authorization can actively prevent the occurrence of illegal usage, while ownership verification is a passive posterior scheme. Model authorization is another perspective of copyright protection and can be combined with ownership verification scheme to achieve better copyright protection. We will treat the combination of ownership verification and model authorization in secure FL as future work.

7 CONCLUSIONS

In this article, we have proposed a novel client-side watermarking scheme for homomorphic-encryption-based secure federated learning. To the best of our knowledge, this is the first scheme to embed the watermark to models in a secure FL environment. The advantages of our scheme are as follows: (1) Using the gradient enhancement method, a client side can embed a backdoor-based watermark into the secure FL model; (2) The proposed non-ambiguous trigger set construction mechanism means that an adversary cannot forge the watermark and claim the copyright of the model; (3) The proposed gradient-enhanced watermark embedding method tackles the issue of slim effects of a single client on watermark embedding in the FL environment; (4) Using our proposed scheme, the FL model meets the requirements of effectiveness, function preservation, low false-positive rate, and resistance to typical watermark removal attacks. In future work, we plan to deploy our watermarking framework into real-life secure FL applications, such as hospitals and credit systems.

REFERENCES

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In Proceedings of 2018 USENIX Security Symposium. 1615–1631.

[2] Mohammed Adnan, Shivam Kalra, Jesse C. Cresswell, Graham W. Taylor, and Hamid R. Tizhoosh. 2022. Federated learning and differential privacy for medical image analysis. Scientific Reports 12, 1 (2022), 1–10.

[3] Rodolfo Stoffel Antunes, Cristiano André da Costa, Arne Küderle, Irmara Abdullahi Yari, and Björn Eskofier. 2022. Federated learning for healthcare: Systematic review and architecture proposal. ACM Transactions on Intelligent Systems and Technology (TIST) 13, 4 (2022), 1–23.

[4] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shihou Moriai, et al. 2017. Privacy-preserving deep learning via additively homomorphic encryption. IEEE Transactions on Information Forensics and Security 13, 5 (2017), 1333–1345.

[5] Eugene Bagdasaryan, Andreas Veit, Yuqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to backdoor federated learning. In Proceedings of 2020 International Conference on Artificial Intelligence and Statistics. 2938–2948.

[6] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In Proceedings of 2017 International Conference on the Theory and Application of Cryptology and Information Security. 409–437.

[7] Trung Kien Dang, Xiang Lan, Jianshu Weng, and Mengling Feng. 2022. Federated learning for electronic health records. ACM Transactions on Intelligent Systems and Technology (TIST) (2022).

[8] Alaa Fkirin, Gamal Attiya, Ayman El-Sayed, and Marwa A. Shouman. 2022. Copyright protection of deep neural network models using digital watermarking: A comparative study. Multimedia Tools and Applications (2022), 1–15.

[9] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. BadNets: Identifying vulnerabilities in the machine learning model supply chain. arXiv preprint arXiv:1708.06733 (2017).

[10] Jia Guo and Miodrag Potkonjak. 2018. Watermarking deep neural networks for embedded systems. In Proceedings of 2018 IEEE/ACM International Conference on Computer-Aided Design, 1–8.

[11] Shangwei Guo, Tianwei Zhang, Han Qiu, Yi Zeng, Tao Xiang, and Yang Liu. 2021. Fine-tuning is not enough: A simple yet effective watermark removal attack for DNN models. In Proceedings of 2021 International Joint Conference on Artificial Intelligence.

[12] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In Advances in Neural Information Processing Systems, Vol. 28.

[13] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloë Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604 (2018).
[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In Proceedings of 2015 IEEE International Conference on Computer Vision. 1026–1034.

[15] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. Tech. Rep. (2009).

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In Proceedings of the IEEE. Vol. 86. 2278–2324.

[17] Qinbin Li, Yaqun Diao, Quan Chen, and Bingsheng He. 2021. Federated learning on non-IID data silos: An experimental study. arXiv preprint arXiv:2102.02079 (2021).

[18] Quushi Li, Ju Ren, Xinglin Pan, Yuezhi Zhou, and Yaoxue Zhang. 2022. ENIGMA: Low-latency and privacy-preserving edge inference on heterogeneous neural network accelerators. In 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS’22). IEEE, 458–469.

[19] Quushi Li, Ju Ren, Yuezhi Zhou, and Yaoxue Zhang. 2022. Privacy-preserving DNN model authorization against model theft and feature leakage. In ICC 2022-IEEE International Conference on Communications. IEEE, 5633–5638.

[20] Zheng Li, Chengyu Hu, Yang Zhang, and Shaoqing Guo. 2019. How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of DNN. In Proceedings of 2019 Annual Computer Security Applications Conference. 126–137.

[21] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. 2020. Reflection backdoor: A natural backdoor attack on deep neural networks. In Proceedings of 2020 European Conference on Computer Vision. 182–199.

[22] Jing Ma, Si-Ahmed Naas, Stephan Sigga, and Xixiang Lyu. 2022. Privacy-preserving federated learning based on multi-key homomorphic encryption. International Journal of Intelligent Systems (2022).

[23] Pratyush Maini, Mohammad Yaghini, and Nicolas Papernot. 2021. Dataset inference: Ownership resolution in machine learning. In Proceedings of 2021 International Conference on Learning Representations.

[24] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In Proceedings of 2017 International Conference Artificial Intelligence and Statistics. 1273–1282.

[25] Sachin Mehta and Mohammad Rastegari. 2021. MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer. In International Conference on Learning Representations.

[26] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. 2020. Adversarial frontier stitching for remote neural network watermarking. Neural Computing and Applications 32, 13 (2020), 9233–9244.

[27] Khalil Muhammad, Qinquin Wang, Diarmuid O’Reilly-Morgan, Elias Tragos, Barry Smyth, Neil Hurley, James Geraci, and Aonghus Lawlor. 2020. Fedfast: Going beyond average for faster training of federated recommender systems. In Proceedings of 2020 ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 1234–1242.

[28] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In Proceedings of 1999 International Conference on the Theory and Applications of Cryptographic Techniques. 223–238.

[29] Jaehyoung Park and Hyuk Lim. 2022. Privacy-preserving federated learning using homomorphic encryption. Applied Sciences 12, 2 (2022), 734.

[30] Francesco Regazzoni, Paolo Palmieri, Fethullah Smailbegovic, Rosario Cammarota, and Ilia Polian. 2021. Protecting artificial intelligence IPs: A survey of watermarking and fingerprinting for machine learning. CAAI Transactions on Intelligence Technology 6, 2 (2020), 182–191.

[31] Phillip Riegler, Thien Duc Nguyen, Markus Miettinan, and Ahmad-Reza Sadeghi. 2022. DeepSight: Mitigating backdoor attacks in federated learning through deep model inspection. In Proceedings of 2022 Network and Distributed System Security Symposium. DOI: https://doi.org/10.14722/ndss.2022.23156

[32] Ronald L. Rivest, Len Adleman, Michael L. Dertouzos, et al. 1978. On data banks and privacy homomorphisms. Foundations of Secure Computation, 11 (1978), 169–180.

[33] Masoumeh Shafeinejad, Nils Lukas, Jiaqi Wang, Xindu Li, and Florian Kerschbaum. 2021. On the robustness of backdoor-based watermarking in deep neural networks. In Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security. ACM, Virtual Event Belgium, 177–188. DOI: https://doi.org/10.1145/3457880.3460401

[34] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

[35] Karan Singhal, Hakim Sidahmed, Zachary Garrett, Shanshan Wu, John Rush, and Sushant Prakash. 2021. Federated reconstruction: Partially local federated learning. Advances in Neural Information Processing Systems 34 (2021).

[36] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N. Asokan. 2021. DAWN: Dynamic adversarial watermarking of neural networks. In Proceedings of 2021 ACM International Conference on Multimedia. 4417–4425.

[37] Buse G. A. Tekgul, Yuxi Xia, Samuel Marchal, and N. Asokan. 2021. WAFFLE: Watermarking in federated learning. In Proceedings of 2021 International Symposium on Reliable Distributed Systems. 310–320.
[38] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin’ichi Satoh. 2017. Embedding watermarks into deep neural networks. In Proceedings of 2017 ACM International Conference on Multimedia Retrieval. 269–277.

[39] Tianhao Wang and Florian Kerschbaum. 2021. RIGA: Covert and robust white-box watermarking of deep neural networks. In Proceedings of 2021 Web Conference. 993–1004.

[40] Mingfu Xue, Shichang Sun, Yushu Zhang, Jian Wang, and Weiqiang Liu. 2022. Active intellectual property protection for deep neural networks through stealthy backdoor and users’ identities authentication. Applied Intelligence (2022), 1–15.

[41] Y. LeCun, C. Cortes, and C. Burges. 2010. MNIST Handwritten Digit Database. (2010). http://yann.lecun.com/exdb/mnist/

[42] Liu Yang, Ben Tan, Vincent W. Zheng, Kai Chen, and Qiang Yang. 2020. Federated recommendation systems. In Federated Learning. Springer, 225–239.

[43] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology 10, 2 (2019), 1–19.

[44] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. 2019. Federated learning. Synthesis Lectures on Artificial Intelligence and Machine Learning 13, 3 (2019), 1–207.

[45] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. 2019. Bayesian nonparametric federated learning of neural networks. In Proceedings of 2019 International Conference on Machine Learning. 7252–7261.

[46] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting intellectual property of deep neural networks with watermarking. In Proceedings of 2018 Asia Conference on Computer and Communications Security. 159–172.

[47] Xinghua Zhu, Jianzong Wang, Zhenhou Hong, and Jing Xiao. 2020. Empirical studies of institutional federated learning for natural language processing. In Proceedings of 2020 Conference on Empirical Methods in Natural Language Processing: Findings. 625–634.

Received 12 December 2022; revised 10 July 2023; accepted 25 September 2023