mathematics

MDPI

Article

TraceGuard: Fine-Tuning Pre-Trained Model by Using Stego Images to Trace Its User

Limengnan Zhou [1,†], Xingdong Ren [2,†], Cheng Qian [2] and Guangling Sun [2,*]

[1] School of Electronic and Information Engineering, University of Electronic Science and Technology of China, Zhongshan Institute, Zhongshan 528402, China; lmnzhou@zsc.edu.cn
[2] School of Communication and Information Engineering, Shanghai University, Shanghai 200444, China; dong0412@shu.edu.cn (X.R.); shuqc@shu.edu.cn (C.Q.)
* Correspondence: sunguangling@shu.edu.cn
† These authors contributed equally to this work.

Abstract: Currently, a significant number of pre-trained models are published online to provide services to users owing to the rapid maturation and popularization of machine learning as a service (MLaaS). Some malicious users have pre-trained models illegally to redeploy them and earn money. However, most of the current methods focus on verifying the copyright of the model rather than tracing responsibility for the suspect model. In this study, TraceGuard is proposed, the first framework based on steganography for tracing a suspect self-supervised learning (SSL) pre-trained model, to ascertain which authorized user illegally released the suspect model or if the suspect model is independent. Concretely, the framework contains an encoder and decoder pair and the SSL pre-trained model. Initially, the base pre-trained model is frozen, and the encoder and decoder are jointly learned to ensure the two modules can embed the secret key into the cover image and extract the secret key from the embedding output by the base pre-trained model. Subsequently, the base pre-trained model is fine-tuned using stego images to implement a fingerprint while the encoder and decoder are frozen. To assure the effectiveness and robustness of the fingerprint and the utility of fingerprinted pre-trained models, three alternate steps of model stealing simulations, fine-tuning for uniqueness, and fine-tuning for utility are designed. Finally, the suspect pre-trained model is traced to its user by querying stego images. Experimental results demonstrate that TraceGuard can reliably trace suspect models and is robust against common fingerprint removal attacks such as fine-tuning, pruning, and model stealing. In the future, we will further improve the robustness against model stealing attack.

Keywords: intellectual property protection; tracing pre-trained model; fine-tuning pre-trained model; steganography network; stego image; fingerprint removal attack

MSC: 94A08; 68T20

Citation: Zhou, L.; Ren, X.; Qian, C.; Sun, G. TraceGuard: Fine-Tuning Pre-Trained Model by Using Stego Images to Trace Its User. *Mathematics* **2024**, *12*, 3333. https://doi.org/10.3390/math12213333

Academic Editor: Antanas Cenys

Received: 20 September 2024
Revised: 15 October 2024
Accepted: 22 October 2024
Published: 24 October 2024

1. Introduction

Due to its powerful representation learning capability, without requiring an extensive number of labeled examples, the self-supervised learning (SSL) paradigm has been pervasively applied in a variety of vision tasks, including facial recognition [1], object detection [2], and object pose estimation [3]. The typical and prevalent SSL algorithms, for instance, SimCLR [4], MoCo v2 [5], and MAE [6], are used to achieve pre-trained models, and the models are fine-tuned by a much smaller amount of data to adapt to diverse downstream tasks. Apparently, the pre-trained models have great values.

As the SSL intensively relies on data and computing resources, only tech giants can deploy well-trained pre-trained models in the cloud and provide MLaaS (machine learning as a service) [7] through charging users (e.g., Google Cloud and Microsoft Azure). Consequently, infringement leads to economic loss for the model owners. One case is that

someone steals the pre-trained model by using stealing attacks and then re-deploys the models to obtain illegal profit. Accordingly, a watermarking-based strategy for protecting the model's copyright is proposed: the model is watermarked, and once the model is suspected, the watermark will be extracted from the model to verify whether it is a pirate model [8–10]. The other case is that some individuals are authorized to use the pre-trained models; however, some of them unlawfully release the model or resell the model for illegal profit, as shown in Figure 1. Therefore, once a suspect model is discovered, it is necessary to trace the source of the model; more specifically, if the suspect model is pirated, it is necessary to attribute it to a specific authorized user who may be the pirate. Unfortunately, the method previously proposed to verify pirated models cannot trace the model, because the watermark is not uniquely embedded into individual models. This indicates that each model must be fingerprinted to link it with each authorized user. In addition, to reliably trace the model, the fingerprint must be robust to pruning, fine-tuning, and even model stealing attacks launched by the malicious user.

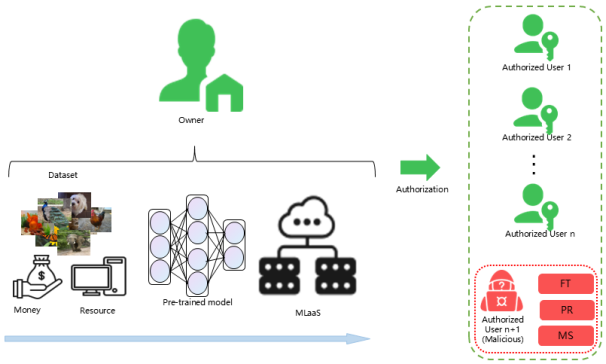

Figure 1. Diagram of tracing a pre-trained model. The model owner spends significant money and resources to obtain and deploy pre-trained models as an MLaaS (machine learning as a service) to attain legal profit. The model owner can also authorize users to legally use their model. However, some malicious users conduct a secondary deployment of the model to earn an illegal profit. To conceal evidence of piracy, malicious users may launch various fingerprint removal attacks on pirate models, such as fine-tuning (FT), pruning (PR), and model stealing (MS), for which it is challenging for the owner to trace the suspect model.

In this paper, TraceGuard is proposed. To the best of our knowledge, it may be the first framework for tracing a pre-trained model to its user. One base aspect is that each individual authorized user is allocated a unique secret key. TraceGuard consists of three components: an encoder, decoder, and pre-trained model. The encoder and decoder apply a set of images, a set of secret keys, and steganography to implement the fingerprint. During the tracing pipeline, the encoder, decoder, and pre-trained model closely interact. First, the encoder and decoder learn to embed secrets and extract them again with a high enough accurate rate while the base pre-trained model is frozen; then, the base pre-trained model is fingerprinted through fine-tuning by using stego images embedded with a unique secret key. Due to the linking between the secret key and the authorized user, each fingerprinted model is related to each authorized user. Finally, stego images embedded with all secret keys are queried, and the secret key with the top extraction accuracy rate is searched to determine the most likely user or whether the model is independent. The model and code can be available at https://github.com/dong-0412/TraceGuard.git (accessed on 15 October 2024).

The main contributions of this work are summarized as follows:

- TraceGuard is proposed. It is the first framework for tracing a pre-trained model to its user, ascertaining which authorized user illegally released the pirate model or verifying that the suspect model is independent.

- A novel fingerprinting strategy is introduced. By using stego images, fine-tuning for unique secret key extraction, model stealing simulations, and fine-tuning for self-supervised learning are alternately executed to balance the effectiveness and robustness of the fingerprint and the utility of the fingerprinted pre-trained model. Experimental results confirm that TraceGuard has generated a traceable pre-trained model with performance well preserved.
- Extensive evaluations on benchmark datasets demonstrate that TraceGuard is robust against fingerprint removal attacks such as model fine-tuning, pruning, and model stealing attack.

2. Related Works

2.1. Copyright Verification and Piracy Attribution for Suspect Model

For copyright verification for suspect models, watermark-based methods are popular. White-box watermarks are embedded directly into a model's structure [8,11,12] or weights [13–15], with extraction achieved through model parameters during copyright verification. In contrast, black-box watermarks do not allow access to parameters; verification depends on specific outputs from designed input examples [9,10,16,17]. Gray-box watermarks combine both methods, using white-box techniques for embedding and black-box for verification [18]. Box-free watermarking does not require a model, relying instead on extracting watermarks from images generated by the suspect model, typically used in image processing tasks [19]. For piracy attribution for suspect models, Chen et al. first introduced model fingerprinting, enabling the traceability of each deployed model [20]. Sun et al. embedded secret keys into fingerprint images, assigning each user a unique image; fine-tuning links these images to specific categories, enabling piracy tracing when the matched image produces a predefined output [21]. Yu et al. embedded secret keys in GANs, allowing owners to trace suspect GANs through images they generate [22]. Li et al. developed an adversarial text generation model for labeling and tracing generated texts [23]. Liu et al. embedded backdoor watermarks in diffusion models, allowing ownership verification even after fine-tuning [24]. Li et al. proposed a method using handcrafted filters and a directional feature learning network to trace AI-generated images, achieving effective results on both GANs and diffusion models [25]. The key information from related works has been summarized in Table 1. Different from the existing works, the objective of our work is to provide copyright protection for pre-trained models, and in particular, to trace the user of a suspect pre-trained model once an infringement is discovered.

Table 1. The literature most related to our work.

Authors and Publication Year	Title	Objective
Chen et al. (2018) [20]	Deepmarks: A digital fingerprinting framework for deep neural networks	Tracing user of classification model
Sun et al. (2021) [21]	Protecting intellectual property of deep neural networks with an additional class and stego images	Tracing user of classification model
Yu et al. (2022) [22]	Responsible disclosure of generative models using scalable fingerprinting	Tracing the source of generated images
Li et al. (2022) [23]	Generating traceable adversarial text examples by watermarking in semantic space	Tracing the source of generated text
Liu et al. (2024) [24]	Lazy Layers to Make Fine-Tuned Diffusion Models More Traceable	Tracing ownership of diffusion models
Li et al. (2024) [25]	Are handcrafted filters helpful for attributing AI-generated images?	Tracing the source of generated images

2.2. Steganography

Steganography is a traditional technique for concealing information into various carriers including text, audio, image, video, and others. With the rapid development of deep learning, steganography techniques have been promoted based on deep neural networks (DNNs). Yao et al. developed a method combining DWT and GAN, enhancing invisibility and data extraction accuracy through dynamic embedding and multi-scale attention [26]. Yu et al. introduced the CRoSS framework, using diffusion models for secure image steganography that allows hiding and retrieving images without extra training, even under various degradations [27]. Bui et al. presented RoSteALS, which utilizes pre-trained autoencoders for secret embedding, ensuring robustness and enabling coverless steganography by generating carrier images from noise or text prompts [28].

2.3. Model Stealing Attack

The purpose of a model stealing attack is to infer the private information of the target black-box model, such as the structure [29] or the parameters [30]. According to the data used by the adversary, model stealing attacks are primarily categorized into data-based and data-free attacks. Karmakar et al. introduced MARICH, a query-efficient model extraction attack that uses public data to create a distributionally equivalent replica of a target model, achieving high accuracy with minimal queries [31]. Regarding the data-free attack, Kariyappa et al. and Truong et al. fed the generated data into the victim and pirate model, and imitated the gradient of the black-box model by zero-order gradient estimation to update the parameters of the victim and pirate model [32,33].

3. Proposed Approach

3.1. Threat Model

The threat model involves an adversary and a defender. The adversary attains illegal benefits by pirating the pre-trained model. The defender is the owner of the suspect pre-trained model, who authorizes one user to legally use the pre-trained model, and expects to trace the responsibility once the suspect model is discovered.

3.1.1. Adversary's Capability and Knowledge

Normal users can obtain authorization from the owner of a pre-trained model and have the right of use for the model. However, a malicious user becomes an adversary once they conduct secondary deployment for the model to obtain illegal benefits. Additionally, to better conceal evidence of piracy, the adversary may perform fingerprint removal attacks such as fine-tuning, pruning, and model stealing attacks on the victim model. Note that only the stealing of the original pre-trained model is considered in our threat model, and not the re-stealing of the pirate model.

3.1.2. Defender's Capability and Knowledge

The defender owns the copyright of the pre-trained model, namely, they own the training data, the structure, and the parameters of the model. Once the defender suspects a model is an infringement on their own model, the defender can trace the suspect model through a black-box query.

3.2. Design Overview

In this section, the pipeline of the proposed TraceGuard is illustrated in Figure 2. All notations used are defined in Table 2. Firstly, Figure 2a shows the learning of the stenography network, composed of an encoder and decoder pair, by using D_f ($D_f \in D_s$). The encoder leans to embed the secret keys into cover images, and the decoder learns how to extract the secret keys from the embedding provided by a base pre-trained model. During the learning of the stenography network, the base pre-trained model is frozen. Secondly, Figure 2b shows how to fingerprint each base pre-trained model using D_s and secret keys. Before fingerprinting, each authorized user is designated a unique secret key.

Then, the steganography network in the previous step is frozen, and the base pre-trained model is fine-tuned by using stego images such that one secret key among all the secret keys can be extracted with a lowest error rate. Such fine-tuning is accomplished for each secret key so that the base pre-trained model is fingerprinted for each authorized user and can be traced to the corresponding user. Finally, Figure 2c shows how to trace the suspect model by querying stego images. The owner of the pre-trained model embeds the secret key designated to each authorized user into D_t and feeds the stego images into the suspect pre-trained model. By comparing the error rate between the secret key extracted from the embedding provided by the suspect model and the embedded secret key, the owner can trace the responsibility of the authorized user holding the secret key with the minimum error rate, provided that the error rate is lower than the pre-set threshold T. Otherwise, the suspect model is determined to be an independent model.

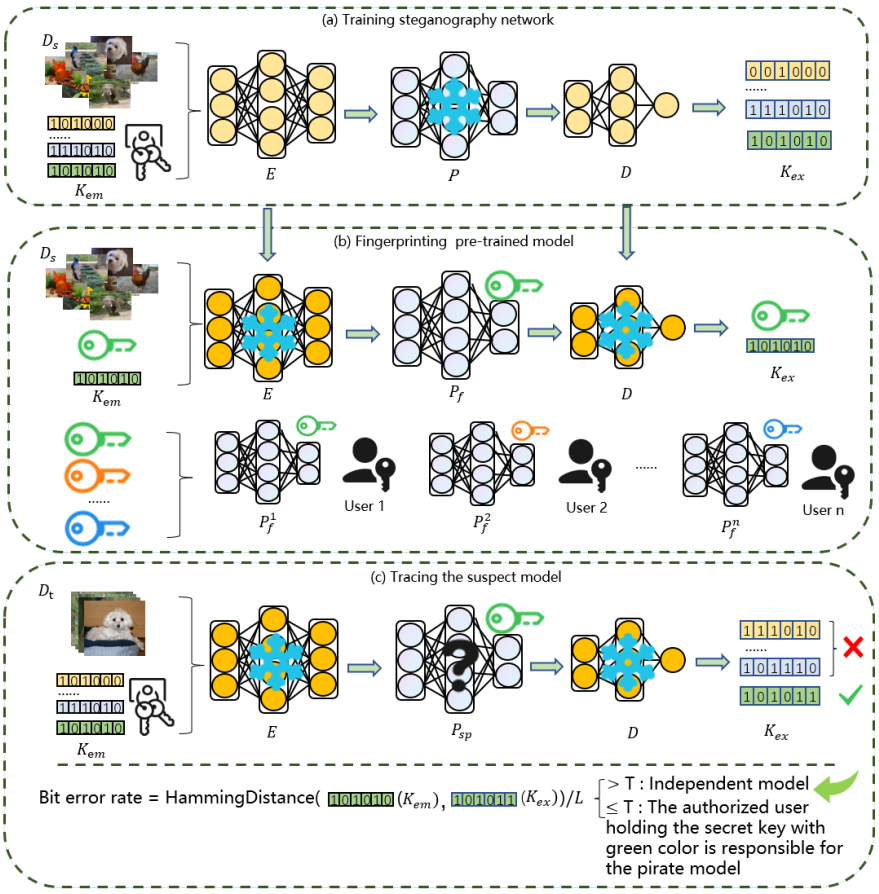

Figure 2. Overview of TraceGuard. (**a**) The steganography network composed of encoder and decoder is learned by using a base pre-trained model, D_s, and a set of secret keys. (**b**) The base pre-trained model is fine-tuned to be individually fingerprinted by using the steganography network, D_f, and secret key designated to each authorized user. (**c**) The suspect pre-trained model is traced by querying stego-images, which are generated by using D_t and steganography network. The authorized user allocated for the secret key with the lowest error rate between the embedded and extracted and below the pre-set threshold T is responsible for the suspect pre-trained model. For a concise description, we only show a single extracted key. In effect, we use the query set to obtain a set of extracted keys so as to obtain a stable error rate. See details in Section 3.5.

Table 2. Definitions of used notation.

Notation	Definition
E, D	encoder and decoder of the steganography networks
P, P_{sd}, P_f, P_{sp}	pre-trained model/shadow pre-trained model/ fingerprinted pre-trained model/suspect model
K_{em}, K_{ex}	embedded key/extracted key
I_{co}, I_{st}	cover images/stego images
D_s D_f, D_t	dataset for training steganography network/ dataset for fingerprinting/tracing pre-trained model

3.3. Training Steganography Network

As shown in Figure 3, encoder E, decoder D, and pre-trained model P participate in the training of the steganography network. $D_s = \{I_{co}^i\}_{i=1}^N$ denotes the image set for training the steganography network, where $I_{co}^i \in \mathbb{R}^{C\times H\times W}$, and C, H, W denote the channel number, height, and width of the image, respectively. $K_{em} \in \{0,1\}^L$ is the binary secret key of length L. The encoder E learns to embed K_{em} into I_{co} and obtain the stego image I_{st}. Next, I_{st} is fed into the pre-trained model to obtain the embedding. Finally, the decoder D learns to extract the secret key K_{ex} from the embedding. The structure of D is composed of two fully connected layers and an activation function ReLU, which can compress the embedding and transform it into K_{ex} with the same length as K_{em}, in which each element is normalized between 0 and 1 using the sigmoid function.

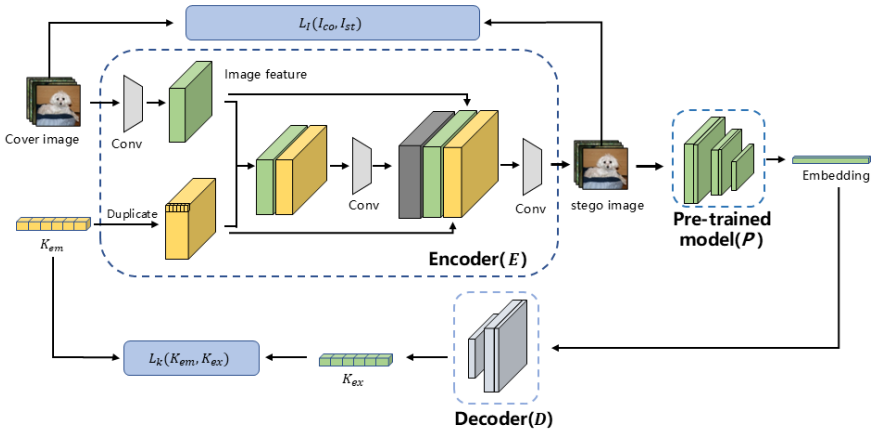

Figure 3. Flowchart of training steganography network: encoder (E), decoder (D), pre-trained model (P), and loss function.

The loss function consists of two items: the secret key loss and the image loss. To evaluate the extraction error of the secret key, the loss is defined. To constrain the visual similarity between I_{st} and I_{co}, the loss is defined. During training, the steganography network P is frozen. Then, E and D are alternately learned by using the combination of the two loss items:

$$\begin{aligned} \mathcal{L}_{ste} &= \lambda_k \mathcal{L}_k(K_{em}, K_{ex}) + \lambda_I \mathcal{L}_I(I_{co}, I_{st}) \\ \mathcal{L}_k(K_{em}, K_{ex}) &= \mathrm{BCE}(K_{em}, K_{ex}) \triangleq \|K_{em} - K_{ex}\|_2^2 / L \\ \mathcal{L}_I(I_{co}, I_{st}) &= \mathrm{MSE}(I_{co}, I_{st}) \triangleq \|I_{co} - I_{st}\|_2^2 / CHW \end{aligned} \tag{1}$$

where λ_k and λ_I are weights to control the balance between the two loss items.

The procedure for training the steganography network is shown in Algorithm 1.

Algorithm 1 Training Steganography Network

Input: base pre-trained model P, D_s, secret key set S_k, training epochs N_t, learning rate η
Output: encoder (E), decoder (D)

for $n = 1$ in N_t **do**
  while $\{I_{co}\}^i \neq NULL$ **do**
    Select $\{I_{co}\}^i \in D_s$, $K_{em}^i \in S_k$ /* $\{I_{co}\}^i$ is a batch of samples */
    $\{I_{st}\}^i = E_\theta(\{I_{co}\}^i, K_{em}^i)$ ▷ embed key
    $\{y\}^i = P(\{I_{st}\}^i)$ ▷ obtain embedding from stego image
    $\{K_{ex}\}^i = D_\theta(\{y\}^i)$ ▷ extract key
    $\mathcal{L} = \lambda_k \mathrm{BCE}(\{K_{ex}\}^i, K_{em}^i) + \lambda_I \mathrm{MSE}(\{I_{co}\}^i, \{I_{st}\}^i)$
    $\theta_{i+1} = \theta_i - \eta \bigtriangledown_\theta \mathcal{L}$ ▷ update the parameters of encoder and decoder
  end while
end for

3.4. Fingerprinting Pre-Trained Model

After learning the steganography network, the pre-trained model is fingerprinted, as shown in Figure 4. The pre-trained model is fine-tuned to implement the fingerprinting. Generally, the fingerprinting is designed to satisfy three requirements. First is the uniqueness of the fingerprint. This requirement is ensured by slightly altering the internal parameters of the pre-trained model to extract a unique key, excluding other keys from images embedded with secret keyst. Second is the robustness against model stealing attack. The model stealing attack has seriously weakened the effectiveness of model tracing. Thus, such an attack is simulated by equipping a shadow model P_{sd} during the fingerprinting. Third is the utility of the fingerprinted model. The requirement indicates that the performance of the fingerprinted model should be well maintained. To meet the requirements, two types of fine-tuning pre-trained model and model stealing simulation towards three objectives are alternately conducted. During the fingerprinting, D_s is used to accomplish the fingerprinting.

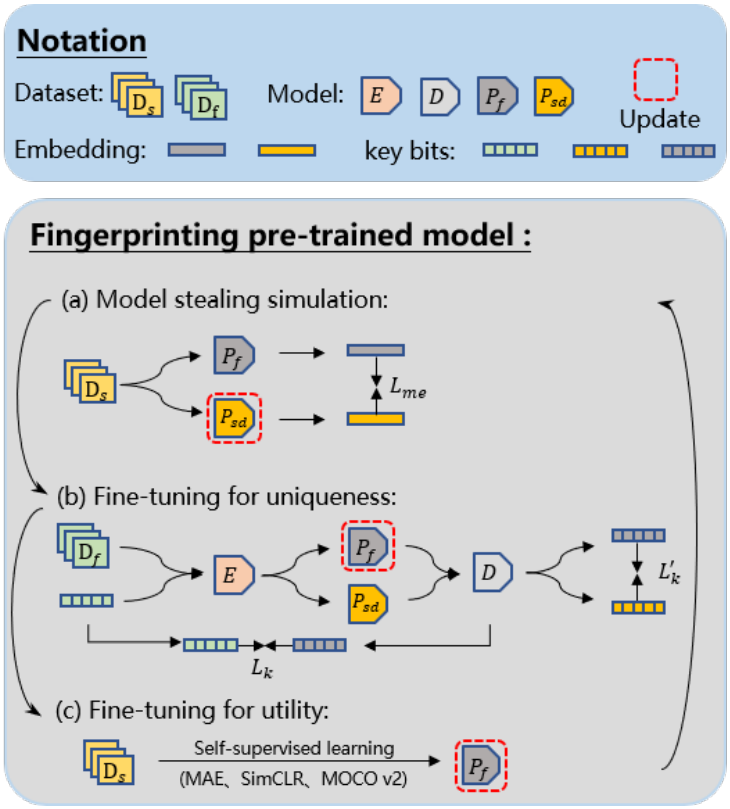

Figure 4. Flowchart of fingerprinting pre-trained model.

3.4.1. Model Stealing Simulation

This step is conducted to achieve robustness against model stealing attacks. In a real scenario, the adversary usually attempts to launch a fingerprint removal attack, just like model stealing, to cloak the evidence of piracy. Thus, a shadow model P_{sd} to follow the function of the fingerprinted model is learned. The purpose of learning the shadow model is to mimic a model stealing attack during fingerprinting. Concretely, a stego image I_{st} is fed into P_f and P_{sd}, and the embeddings given by P_f and P_{sd} are used to evaluate the similarity between P_f and P_{sd}. Thus, the shadow model P_{sd} is learned and updated by minimizing the following loss:

$$\mathcal{L}_{ms} = 1 - cos(P_f(I_{st}), P_{sd}(I_{st})) \tag{2}$$

where $P_f(I_{st})$ and $P_{sd}(I_{st})$ refer to the embedding output by P_f and $P_{sd.}$

3.4.2. Fine-Tuning for Uniqueness

This step is conducted to achieve uniqueness of the fingerprint. D_f, as a subset of D_s, is selected from D_s. We select $K_{\mathrm{em}}{}^{\mathrm{j}}$ from $K_{em} \in \{0,1\}^{\mathrm{L}}$, designated for the j-th authorized user. E receives $I_{co} \in D_f$ and embeds $K_{\mathrm{em}}{}^{\mathrm{j}}$ to generate the stego image I_{st}. Then, I_{st} is sent to P_f and P_{sd}, and embedding is obtained accordingly. Finally, the two embeddings are fed into D to extract $K_{\mathrm{ex}}{}^{\mathrm{j}}$ and $K'_{\mathrm{ex}}{}^{\mathrm{j}}$, respectively.

Two secret key extraction loss items compose the total loss for fine-tuning. One loss item, L_k, is designed to evaluate the discrepancy between $K_{\mathrm{em}}{}^{\mathrm{j}}$ and $K_{\mathrm{ex}}{}^{\mathrm{j}}$, which are extracted from P_f, and the other loss item, L'_k, is designed to evaluate the discrepancy between $K_{\mathrm{em}}{}^{\mathrm{j}}$ and $K'_{\mathrm{ex}}{}^{\mathrm{j}}$, which are extracted from P_{sd}. The incorporation of L'_k aims to enhance the robustness of P_f against model stealing attacks. Thus, the pre-trained model is fine-tuned for uniqueness by minimizing the following loss:

$$\begin{aligned} \mathcal{L}_{\mathrm{uni}} &= \lambda_{P_f}\mathcal{L}_k(K_{\mathrm{em}}{}^{\mathrm{j}}, K_{\mathrm{ex}}{}^{\mathrm{j}}) + \lambda_{P_{sd}}\mathcal{L}'_k(K'_{\mathrm{ex}}{}^{\mathrm{j}}, K^j_{\mathrm{em}}) \\ \mathcal{L}_k(K^j_{\mathrm{em}}, K^j_{\mathrm{ex}}) &= \mathrm{BCE}(K^j_{\mathrm{em}}, K^j_{\mathrm{ex}}) \\ \mathcal{L}'_k(K'_{\mathrm{ex}}{}^{\mathrm{j}}, K^j_{\mathrm{em}}) &= \mathrm{BCE}(K'_{\mathrm{ex}}{}^{\mathrm{j}}, K^j_{\mathrm{em}}) \end{aligned} \tag{3}$$

where λ_{P_f} and $\lambda_{P_{sd}}$ are the weights of the two loss items.

3.4.3. Fine-Tuning for Utility

This step is conducted to ensure the utility of the fingerprinted model. Due to the fine-tuning for uniqueness, the parameters of the fine-tuned pre-trained model have definitely deviated from their original positions, and the utility of the pre-trained model may be more or less damaged. To restore the utility of the fingerprinted model, fine-tuning for utility has to be conducted. In this step, three self-supervised learning algorithms are used, namely SimCLR, MoCov2, and MAE (see details in experiment). The three steps are successively and alternately executed, as shown in Algorithm 2.

Algorithm 2 Fingerprinting Pre-trained Model

Input: base pre-trained model P, steganography network including encoder (E) and decoder (D), D_s, D_f, K_{em}^j, fingerprinting epochs N_f, learning rate η_1, η_2, η_3
Output: fingerprinted model P_f

$P_f \leftarrow P$
for $n = 1$ **to** N_f **do**
  /* model stealing simulation */
  while $\{I_{co}\}^i \neq NULL$ **do**
    select $\{I_{co}\}^i \in D_s$
    $L_{me} = 1 - \cos(P_f(\{I_{co}\}^i), P_{sd}(\gamma)(\{I_{co}\}^i))$
    $\gamma_{i+1} = \gamma_i - \eta_1 \nabla_\gamma L_{me}$ /* *update the parameters of* P_{sd} */
  end while
  /* end of model stealing simulation */
  /* fine-tuning for uniqueness */
  while $\{I_{co}\}^i \neq NULL$ **do**
    select $\{I_{co}\}^i \in D_f$
    $\{I_{st}\}^i = E(\{I_{co}\}^i, K_{em}^j)$
    $\{y\}^i = P_f(\beta)(\{I_{st}\}^i)$
    $\{y'\}^i = P_{sd}(\{I_{st}\}^i)$
    $\{K_{ex}\}^i = D(\{y\}^i)$
    $\{K'_{ex}\}^i = D(\{y'\}^i)$
    $L_{uni} = \lambda_{pf}\text{BCE}(\{K_{ex}\}^i, K_{em}^j) + \lambda_{psd}\text{BCE}(\{K'_{ex}\}^i, K_{em}^j)$
    $\beta_{i+1} = \beta_i - \eta_2 \nabla_\beta L_{uni}$ /* *fine-tune the parameters of* P_f */
  end while
  /* end of fine-tuning for uniqueness */
  /* fine-tuning for utility */
  while $\{I_{co}\}^i \neq NULL$ **do**
    select $\{I_{co}\}^i \in D_s$
    $\beta_{i+1} = \beta_i - \eta_3 \nabla_\beta L_{ssl}(\{I_{co}\}^i)$ /* *fine-tune the parameters of* P_f */
  end while
  /* end of fine-tuning for utility */
end for

3.5. Tracing Suspect Model

The objective of tracing the suspect model is to ascertain which authorized user has released the suspect model with the highest probability. D_t refers embedded secret keys used to form stego images, which are queried. During tracing, all designated keys are checked to search for the key with the minimum extraction error rate. To calculate the extraction error rate, the d-dimension of the extracted key is converted into a d-bit string, in which the value between 0 and 1 of each dimension is converted to 0 if the value is larger than 0.5 or 1 otherwise. The extraction error rate is defined as the Hamming distance between the embedded and extracted key divided by the length of key L. In addition, the threshold T is set to exclude the possibility of an independent model if the minimum extraction error rate is larger than T.

Given the pre-trained model, we can deduce that the demanded time during tracing is linear with N and M, in which N denotes the number of user and M denotes the number of images in the query set. The dominant total time during tracing is approximated as $N \times M \times (T_{ex-emd} + T_{ex-key})$, where T_{ex-emd} and T_{ex-key} denote, for one query, the time for extracting embedding consumed by the pre-trained model and the time for extracting the secret key consumed by the decoder in the stenography network, respectively. Since the generation of stego images that embed all secret keys into cover images is finished in advance, the time for embedding the secret keys should not be considered in the tracing phase.

The procedure for tracing a suspect model is shown in Algorithm 3.

Algorithm 3 Tracing Suspect Model

Input: suspect model P_{sp}, steganography network including encoder E and decoder D, D_t, designated key set S_d, threshold T
Output: tracing result

for $K_{em}^i \in S_d$ **do**
  $\{K_{ex}\}^i = \text{BINARY}(D(P_{sp}(E(D_t, K_{em}^i))))$ /* *embed and extract key* */
  $\{hd\}^i = \textit{Hamming Distance}(K_{em}^i, K_{ex}^i)$
  $T_i = \text{ave}(\{hd\}^i)/L$ /* T_i *is the average extraction error rate for* K_{em}^i */
end for
$T_n = \min(T_1, T_2, \ldots, T_N)$
if $T_n \leq T$ **then**
  The suspect model is pirated by the n-th authorized user
else
  The suspect model is an independent model
end if

4. Experiments

4.1. Experiment Setup

4.1.1. Datasets

ImageNet [34] and STL-10 [35] are used in our experiment. For ImageNet, 50 categories are randomly selected for training a pre-trained model. In addition, 12 categories are randomly selected as the fine-tuning/testing dataset for the downstream task. For STL-10, it contains 100,000 unlabeled examples, 5000 training labeled examples, and 8000 testing labeled examples. The pre-trained model is trained with unlabeled examples, and the model is fine-tuned/tested with labeled examples for the downstream task.

4.1.2. Learning Algorithm and Structure for Pre-Trained Model

SimCLR, MoCo v2, and MAE are selected as the self-supervised learning (SSL) algorithms for our experiments, based on their demonstrated effectiveness and relevance to the research goals. SimCLR and MoCo v2 are well-established contrastive learning methods, known for their ability to generate high-quality image representations, which are essential for tasks like image classification and retrieval. On the other hand, MAE (masked autoencoder) offers a different approach, focusing on reconstructing missing parts of the input data, providing complementary insights into self-supervised learning. For the model structures, SimCLR and MoCo v2 were applied to train convolutional models, including ResNet-18 (RN-18), ResNet-34 (RN-34), and DenseNet121 (DN-121). Meanwhile, MAE was used to train vision transformer (ViT) models, including both the base version (ViT-B) and the large version (ViT-L). These diverse models allow us to comprehensively evaluate the performance of these leading SSL approaches across a range of architectures.

4.1.3. Hyperparameters

PyTorch 1.0 is used on a single RTX 3090 GPU, and epoch numbers for learning pre-trained model, for training steganography network, and for fingerprinting pre-trained model are set 200, 100, and 50, respectively. The batch size is set to 128, and the learning rate is set to 0.0001.

4.2. Capacity of Tracing

The capacity of tracing refers to the number of distinct keys that TraceGuard can accommodate without crosstalk between two different keys. The capacity of TraceGuard is directly related to the length of binary secret key L and the threshold D for the code distance (Hamming distance) of two different keys $C(K_{\text{em}}^i, K_{\text{em}}^j)$, $i \neq j$. The value of D indicates that the code distance of any two keys is at least D bits, namely $C(K_{\text{em}}^i, K_{\text{em}}^j) \geq D$. However,

the relationship between L and D is complex. A longer L can accommodate more users but permits fewer errors for the extraction of the key. A larger D permits more errors in the extraction of the key but can accommodate fewer users. In other words, the capacity and the reliability of tracing are trade-offs. To determine the appropriate L and D, this experiment is designed. The secret key K_{em}^{i} is chosen, and the pre-trained model is fingerprinted. During testing, the extraction error rate for both K_{em}^{i} and K_{em}^{j} : $(C(K_{\text{em}}^{i}, K_{\text{em}}^{j}) = D)$ is calculated. The result shows that TraceGuard can trace the fingerprinted model only if K_{em}^{i} does not crosstalk with K_{em}^{j}, meaning that the extraction error rate of K_{em}^{i} is less than that of K_{em}^{j}. As shown in Figure 5, L varies within the range {10, 20, 30, 40, 50, 60} and D varies within the range {1, 2, 3, 4, 5}. It is found that as L increases, the extraction error rate of K_{em}^{i} also increases. The extraction error rate of K_{em}^{j} also has a broader distribution, and the influence of D diminishes. In terms of a full evaluation, $L = 30$ and $D = 4$ are chosen as the optimal option, and TraceGuard can accommodate approximately 1.6×10^7 authorized users. This option is maintained for all experiments. In Figure 5, FP, NFP-MIN, and NFP-MAX are used to represent the extraction error rate for K_{em}^{i}, minimum extraction error rate, and maximum extraction error for K_{em}^{j}, respectively. Notice that the "Bit error rate" in all the following figures denotes the extraction error rate.

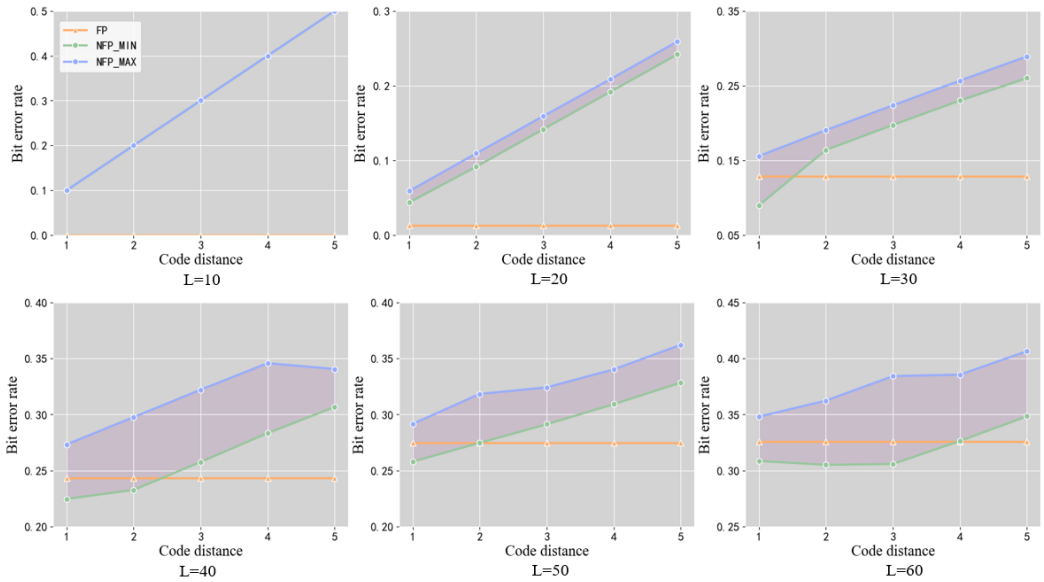

Figure 5. The impact of the length L and minimum code distance D on extraction error rate. The orange line represents the extraction error rate (FP) of K_{em}^{i}, where K_{em}^{i} corresponds to the i-th fingerprinted model. The blue line and the green line represent the maximum extraction error rate (NFP-MAX) and the minimum extraction error rate (NFP-MIN) for K_{em}^{j} such that $C(K_{\text{em}}^{i}, K_{\text{em}}^{j}) = D$. The purple area represents the extraction error rate distribution for extracting K_{em}^{j}.

4.3. Effectiveness of TraceGuard

4.3.1. Uniqueness

In TraceGuard, the secret key does not need to be perfectly extracted. Instead, it is sufficient if FP is smaller than NFP-MIN, as TraceGuard identifies one as the pirate holding the key with the lowest extraction error rate. The secret key K_{em}^{i} is chosen to fingerprint the pre-trained model. During testing, 100 K_{em}^{j} : $(C(K_{\text{em}}^{i}, K_{\text{em}}^{j}) \geq 4, L = 30)$ is randomly selected. Two datasets (STL-10 and ImageNet) and three SSL algorithms (SimCLR, MoCo v2, and MAE) for five models (RN-18, RN-34, DN-121, ViT-B, ViT-L) are evaluated. The results listed in Table 3 demonstrate that FP is consistently lower than NFP-MIN, which implies that TraceGuard can successfully trace the suspect model.

Table 3. Effectiveness of the proposed TraceGuard.

SSL	Model	ImageNet				STL-10			
		FP	NFP-MIN	FP-ACC (%)	Base-ACC (%)	FP	NFP-MIN	FP-ACC (%)	Base-ACC (%)
SimCLR	RN-18	0.1293	0.2031	85.36	86.81	0.0334	0.1666	67.81	67.76
	RN-34	0.1378	0.2639	86.33	90.61	0.0686	0.1351	70.40	71.66
	DN-121	0.2518	0.3187	91.22	91.86	0.2704	0.3367	72.86	72.36
MoCo v2	RN-18	0.0315	0.1542	84.52	81.69	0.0337	0.1000	61.85	59.93
	RN-34	0.0143	0.1377	83.43	80.86	0.1119	0.2447	60.24	58.89
	DN-121	0.0786	0.1612	84.72	86.58	0.0695	0.2009	62.75	63.05
MAE	ViT-B	0.0012	0.1333	88.64	86.78	0.2000	0.3042	72.38	70.63
	ViT-L	0.1001	0.1667	79.05	84.56	0.1332	0.2618	60.08	67.46

4.3.2. Utility

The impact of fingerprinting on the utility of the pre-trained model is also probed. The fingerprinted pre-trained model and the base pre-trained model are adapted separately on the downstream classification task. Then, two tested pre-trained models on the downstream classification task are fine-tuned. By comparing the downstream performances of the two models, the impact of fingerprinting on the pre-trained model is assessed. FP-ACC and base-ACC denote the performance of the fingerprinted model and base model, respectively. As shown in Table 3, the performance of the fingerprinted model and the base model are comparable. Some FP-ACC values are even larger than base-ACC values. Thus, we believe that the performance of the fingerprinted model has not decreased due to the fingerprinting.

4.3.3. Independent Model Decision

If the suspect model is an independent model, TraceGuard can still find a key with minimal FP. However, the independent model should not be attributed to any authorized user, and we need to set a threshold T to solve the problem thereafter. If the smallest FP is less than T, the suspect model is traced to the user who has been allocated this key. Otherwise, it is determined to be an independent model. In fact, this is a binary classification to determine whether the suspect model is independent or fingerprinted. Therefore, the ROC curve is used to search an appropriate T. The fingerprinted models are used in our experiment as positive examples, and the same number of independent models are learned as negative examples. Positive examples are tested with their corresponding secret key to obtain FP, while negative examples are tested with all keys, and we choose the lowest error rate as FP. Then, the T value is changed for calculating TPR and FPR to generate the ROC curve plotted in Figure 6a. As the AUC value is quite close to one, it is presumed that T can effectively distinguish fingerprinted and independent models.

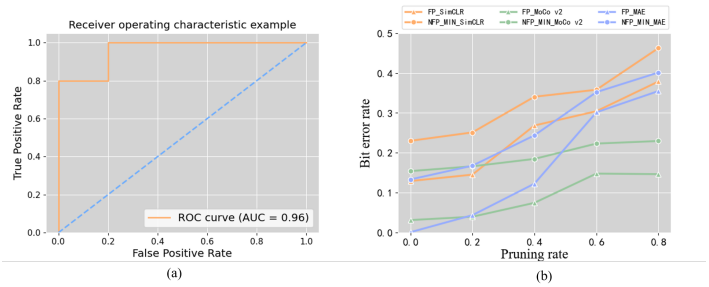

Figure 6. (**a**) ROC curve for evaluating binary classification of the independent model and fingerprinted model. (**b**) Robustness against pruning.

4.4. Robustness of TraceGuard Against Fingerprint Removal Attack

To prevent the inventor of a pre-trained model from tracing a pirate model, the adversary may fine-tune or prune the fingerprinted pre-trained model to conceal the evidence of piracy. Additionally, the adversary can also follow a fingerprinted pre-trained model with a similar

function via a model stealing attack. Thus, evaluating robustness against fine-tuning, pruning, and model stealing is essential. In this experiment, the fingerprinted model is fine-tuned, pruned, and stolen to assess its robustness against a fingerprint removal attack.

4.4.1. Pruning Attack

Model pruning is used to remove redundant neurons from a network to obtain a reduced model while maintaining its performance. In this experiment, the neurons of the fingerprinted model are removed. As shown in Figure 6b, as the pruning rate increases, the pre-trained model is more damaged, leading to a situation in which it is more challenging to extract the key from the embedding of the fingerprinted model. However, overall, FP is always lower than NFP-MIN, so TraceGuard can still successfully trace the suspect model under a pruning attack.

4.4.2. Fine-Tuning Attack

Fine-tune all layers(FTAL) and re-train all layers (RTAL) are the two primary approaches for fine-tuning the pre-trained model. These refer to fine-tuning the pre-trained model with data that are the same as or distinct from the data used in the pre-trained model's learning, respectively. The pre-trained models that learned with three self-supervised algorithms are tested, the epoch number of fine-tuning is set from among the options 0, 20, 40, 60, 80, 100, and FP and NFP-MIN are obtained, respectively. As shown in Figure 7, as the epoch of fine-tuning increases, the error rate of key extraction also increases. In addition, after more than 20 epochs, the key extraction error rate has shown a smooth fluctuation. Importantly, FP is consistently lower than NFP-MIN, demonstrating that TraceGuard can still trace the source of the suspect model under a fine-tuning attack.

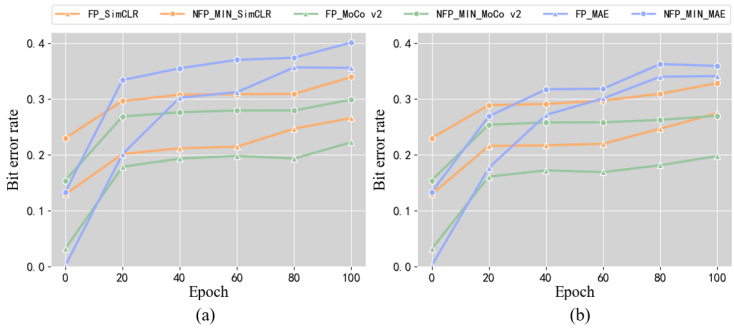

Figure 7. The robustness against fine-tuning: (**a**) as a result of FTAL, (**b**) as a result of RTAL.

4.4.3. Model Stealing Attack

The model stealing attack employs a query dataset to acquire embeddings from the victim pre-trained model. These embeddings are then used as labels to train a functionally similar pirate model. This evaluation focuses specifically on the case of stealing the original fingerprinted model, as a pirate model is expected to perform worse than the original. In the experiments, the pirate model was trained using the same dataset that was used to train the original fingerprinted model. This alignment in datasets enhances the effectiveness of the attack by allowing the pirate model to leverage the same data distributions and features learned by the original model. The pirate model is evaluated using both the fingerprint extraction error rate (FP) and the non-fingerprint minimum error rate (NFP-MIN). As shown in Table 4, the extraction error rate for the pirated model is higher than that of the original fingerprinted model, with FP generally around 0.4. This indicates that the characteristics preserved in the pirate model are significantly attenuated due to the stealing process. Furthermore, it is observed from Table 4 that FP and NFP-MIN are relatively close, although FP remains lower than NFP-MIN. This proximity suggests the possibility of inadvertently misidentifying the pirate model during the tracing process. Overall, while the fingerprinted model demonstrates a certain degree of robustness against model stealing attacks, there remains room for improvement in ensuring more reliable tracing capabilities.

Table 4. Robustness against model stealing attack.

SSL	Model	Original Model		Pirate Model	
		FP	NFP-MIN	FP	NFP-MIN
SimCLR	RN-18	0.1293	0.2301	0.3967	0.4183
	RN-34	0.1378	0.2639	0.4554	0.4591
MoCo v2	RN-18	0.0315	0.1542	0.4312	0.4725
	RN-34	0.0143	0.1377	0.3914	0.4112
MAE	ViT-B	0.0012	0.1333	0.3722	0.4013

4.4.4. The Overall Evaluation for Robustness

To comprehensively evaluate the robustness of TraceGuard against the above fingerprint removal attacks, M keys are utilized to fingerprint an identical pre-trained model, and M fingerprinted models are generated accordingly. In order to better mimic real situations, several independent models are also incorporated. Experiments are conducted to record the tracing accuracy. As shown in Table 5, TraceGuard is reliable on the original, the fine-tuned, and the pruned model. However, the robustness against model stealing attacks is unsatisfactory. In future works, there is still room for improving the robustness against model stealing attacks.

Table 5. Performances of TraceGuard under fingerprint removal attacks.

SSL	Original		Fine-Tuning		Pruning		Model Stealing	
	M = 5	M = 10	M = 5	M = 10	M = 5	M = 10	M = 5	M = 10
SimCLR	1	1	1	1	1	1	1	1
MoCo v2	1	1	1	1	1	1	0.95	0.90
MAE	1	1	1	1	1	1	1	0.93

4.5. Ablation Study

4.5.1. Impact of the Number of Examples Used in Fingerprinting Phase

In this experiment, the number of examples used in the fingerprinting step are varied to probe its influence on the extraction error rate. The number of examples are varied from 4.8 k to 0.3 k and tested on three pre-trained models that have learned with the simclr, MoCo v2, and MAE algorithms, respectively. The trend in Figure 8a shows that the extraction error rate increases with fewer used examples. However, this does not imply that more examples are preferable, as more training time is also consumed. Consequently, fewer examples can be selected without compromising the extraction error rate. Eventually, 4.8 k example are used in the fingerprinting step.

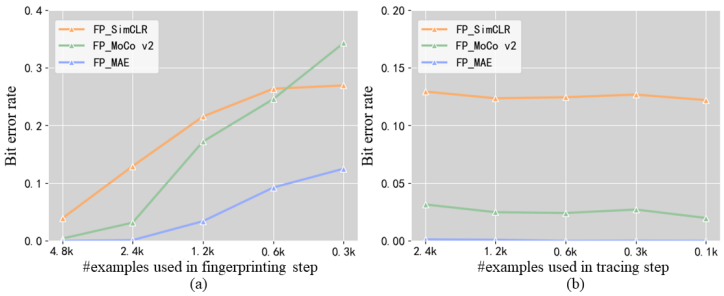

Figure 8. (**a**) Impact of the number of examples used in fingerprinting step. (**b**) Impact of the number of examples used in tracing step.

4.5.2. Impact of the Number of Examples Used in Tracing Phase

In this experiment, the number of examples used in the tracing step is varied to probe their impact on the extraction error rate. The number of tracing examples is varied from 2.4 k to 0.1 k and tested on the three previously mentioned pre-trained models. As shown

in Figure 8b, the number of examples has a negligible effect on the bit error rate. Therefore, to improve the efficiency of tracing, the number of examples used in the tracing step can be small.

4.5.3. Impact of Self-Supervised Fine-Tuning

In this experiment, the influence of fine-tuning with self-supervised learning on performance is probed. Self-supervised fine-tuning is used to maintain the utility of the fingerprinted model. Keeping all other conditions unchanged, two fingerprinted models are obtained with and without self-supervised fine-tuning; the resultant models are named A and B, respectively. The performances evaluated by the bit error rate and ACC(%) of the two models are shown in Figure 9a and Figure 9b, respectively. First, the extraction error rate is used to compare model A and model B. As shown in Figure 9a, the FP of A is consistently lower than that of B, indicating that self-supervised fine-tuning can also increase the uniqueness. Second, accuracy on the downstream task is used to compare model A and model B. The two fingerprinted models are evaluated on downstream classification tasks separately, and the results are shown in Figure 9b. The accuracy of A is higher B, indicating that self-supervised fine-tuning can increase the utility.

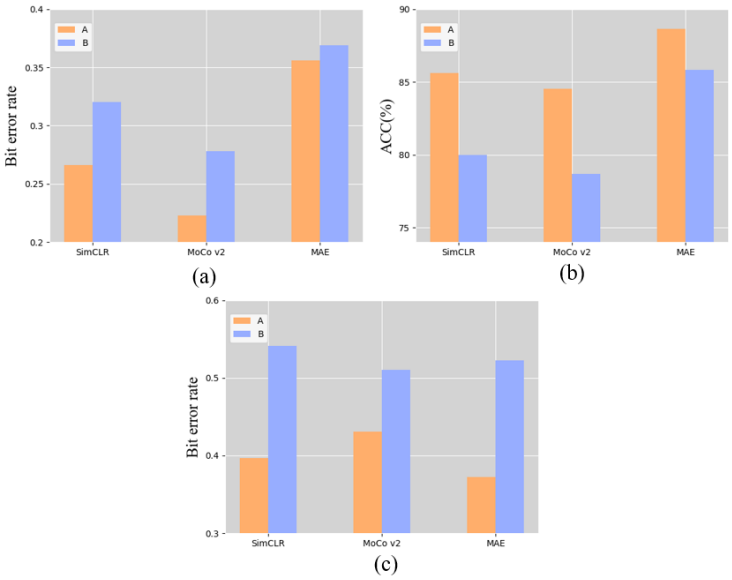

Figure 9. (**a**) The impact of self-supervised fine-tuning on uniqueness. (**b**) The impact of self-supervised fine-tuning on utility. (**c**) The impact of model stealing simulation on robustness.

4.5.4. Impact of the Model Stealing Simulation

In this experiment, the influence of a model stealing simulation on the robustness of a fingerprinted model against a model stealing attack is probed. Keeping all other conditions unchanged, two fingerprinted models are obtained with and without a model stealing simulation in fingerprinting the pre-traind model; these are named A and B, respectively. Then, the two fingerprinted models are stolen, and two pirate models are obtained accordingly. FP results for the two pirate models are shown in Figure 9c, and the results show that the FP of A is consistently lower than that of B, indicating that the model stealing simulation can improve the robustness of the fingerprinted model against a model stealing attack.

5. Conclusions

In this paper, TraceGuard, a framework for tracing infringement of a pre-trained model, is proposed. Our investigation contributes to counteracting the piracy of pre-trained models conducted by authorized users. Each authorized user is allocated a unique secret key. First, the steganography network composed of encoder and decoder learns to embed the secret

key into a cover image and extract the secret key from the output embedding of the pre-trained model via the steganography network. Then, the pre-trained model is fine-tuned using stego images embedded with a unique secret key so as to force the decoder to extract the unique key with the minimum error, and such fine-tuning is performed according to each key designated to each authorized user. Finally, during tracing, if the minimum key extraction error is larger than T, the suspect model is identified as an independent model; otherwise, it has been pirated by the authorized user holding the key with the minimum extraction error. For TraceGuard, empirical results demonstrate both its reliability and robustness against multiple fingerprint removal attacks, including fine-tuning, pruning, and model stealing attack. In the future, its robustness against model stealing attacks will be improved.

Author Contributions: Conceptualization, L.Z. and X.R.; methodology, L.Z., X.R., C.Q. and G.S.; software, X.R. and C.Q.; validation, L.Z., X.R. and C.Q.; supervision, G.S.; project administration, C.Q. and G.S.; funding acquisition, L.Z. and G.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported by the National Natural Science Foundation of China (Grant No. 61901096), the CCF-Ant Privacy Computing Special Research Fund (Grant No. CCF-AFSG RF20220019.a), and the Guangdong Basic and Applied Basic Research Foundation (Grant No. 2023A1515010815).

Data Availability Statement: The datasets employed in the experiments are open source. The ImageNet dataset is accessible at https://image-net.org/index.php (accessed on 15 October 2024), and the STL-10 dataset is available at https://cs.stanford.edu/~acoates/stl10/ (accessed on 15 October 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Al Bdairi, A.J.A.; Xiao, Z.; Alkhayyat, A. Face recognition based on Deep Learning and FPGA for ethnicity identification. *Appl. Sci.* **2022**, *12*, 2605. [CrossRef]
2. Zou, Z.; Chen, K.; Shi, Z.; Guo, Y. Object detection in 20 years: A survey. *Proc. IEEE* **2023**, *111*, 257–276. [CrossRef]
3. Sun, J.; Wang, Z.; Zhang, S. Onepose: One-shot object pose estimation without cad models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 6825–6834.
4. Chen, T.; Kornblith, S.; Norouzi, M. A simple framework for contrastive learning of visual representations. In Proceedings of the International Conference on Machine Learning, Baltimore, MD, USA, 17–23 July 2022; pp. 1597–1607.
5. Chen, X.; Fan, H.; Girshick, R. Improved baselines with momentum contrastive learning. *arXiv* **2020**, arXiv:2003.04297.
6. He, K.; Chen, X.; Xie, S. Masked autoencoders are scalable vision learners. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 16000–16009.
7. Ribeiro, M.; Grolinger, K.; Capretz, M.A. Mlaas: Machine learning as a service. In Proceedings of the 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 9–11 December 2015; pp. 896–902.
8. Zhao, X.; Yao, Y.; Wu, H. Structural watermarking to deep neural networks via network channel pruning. In Proceedings of the 2021 IEEE International Workshop on Information Forensics and Security (WIFS), Montpellier, France, 7–10 December 2021; pp. 1–6.
9. Yadollahi, M.M.; Shoeleh, F.; Dadkhah, S. Robust black-box watermarking for deep neural network using inverse document frequency. In Proceedings of the 2021 IEEE International Conference on Dependable, Autonomic and Secure Computing, International Conference on Pervasive Intelligence and Computing, International Conference on Cloud and Big Data Computing, International Conference on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Online, 25–28 October 2021; pp. 574–581.
10. Zhang, T.; Wu, H.; Lu, X.; Han, G.; Sun, G. AWEncoder: Adversarial Watermarking Pre-Trained Encoders in Contrastive Learning. *Appl. Sci.* **2023**, *13*, 3531. [CrossRef]
11. Zhang, J.; Chen, D.; Liao, J.; Ma, Z.; Fang, H.; Zhang, W.; Feng, H.; Hua, G.; Yu, N. Robust Model Watermarking for Image Processing Networks via Structure Consistency. *IEEE Trans. Pattern Anal. Mach. Intell.* **2024**, *46*, 6985–6992 [CrossRef] [PubMed]
12. Zhang, J.; Chen, D.; Liao, J. Passport-aware normalization for deep model protection. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 22619–22628.
13. Zhu, H.; Liang, S.; Hu, W.; Li, F.; Jia, J.; Wang, S. Reliable Model Watermarking: Defending Against Theft without Compromising on Evasion. *arXiv* **2024**, arXiv:2404.13518.
14. Kuribayashi, M.; Tanaka, T.; Suzuki, S. White-box watermarking scheme for fully-connected layers in fine-tuning model. In Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security, Online, 22–25 June 2021; pp. 165–170.

15. Wang, Z.; Wu, Y.; Huang, H. Defense against Model Extraction Attack by Bayesian Active Watermarking. In Proceedings of the Forty-First International Conference on Machine Learning, Vienna, Austria, 21–27 June 2024.
16. Maung, A.P.; Kiya, H. Piracy-resistant DNN watermarking by block-wise image transformation with secret key. In Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security, Online, 22–25 July 2021; pp. 159–164.
17. Szyller, S.; Atli, B.G.; Marchal, S. Dawn: Dynamic adversarial watermarking of neural networks. In Proceedings of the 29th ACM International Conference on Multimedia, Online, 20–24 October 2021; pp. 4417–4425.
18. Rouhani, B.D.; Chen, H.; Koushanfar, F. Deepsigns: An end-to-end watermarking framework for protecting the ownership of deep neural networks. In Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Providence, RI, USA, 13–17 April 2019; pp. 1–12.
19. Wu, H.; Liu, G.; Yao, Y.; Zhang, X. Watermarking neural networks with watermarked images. *IEEE Trans. Circuits Syst. Video Technol.* **2021**, *31*, 2591–2601. [CrossRef]
20. Chen, H.; Rouhani, B.D.; Koushanfar, F. DeepMarks: A Digital Fingerprinting Framework for Deep Neural Networks. *arXiv* **2018**, arXiv:1804.03648.
21. Sun, S.; Xue, M.; Wang, J. Protecting the intellectual properties of deep neural networks with an additional class and steganography images. *arXiv* **2021**, arXiv:2104.09203.
22. Yu, N.; Skripniuk, V.; Chen, D.; Davis, L.; Fritz, M. Responsible, Disclosure, of Generative, Models Using Scalable Fingerprinting. In Proceedings of the International Conference on Learning Representations (ICLR), Online, 25 April 2022.
23. Li, M.; Wu, H.; Zhang, X. Generating traceable adversarial text examples by watermarking in the semantic space. *J. Electron. Imaging* **2022**, *31*, 063034. [CrossRef]
24. Liu, H.; Zhang, W.; Li, B.; Ghanem, B.; Schmidhuber, J. Lazy Layers to Make Fine-Tuned Diffusion Models More Traceable. *arXiv* **2024**, arXiv:2405.00466.
25. Li, J.; Wang, H.; Li, S.; Qian, Z.; Zhang, X.; Vasilakos, A.V. Are handcrafted filters helpful for attributing AI-generated images? In Proceedings of the 32nd ACM International Conference on Multimedia, Melbourne, VIC, Australia, 28 October–1 November 2024. [CrossRef]
26. Yao, Y.; Wang, J.; Chang, Q.; Ren, Y.; Meng, W. High invisibility image steganography with wavelet transform and generative adversarial network. *Expert Syst. Appl.* **2024**, *249*, 123540. [CrossRef]
27. Yu, J.; Zhang, X.; Xu, Y.; Zhang, J. Cross: Diffusion model makes controllable, robust and secure image steganography. In Proceedings of the 37th International Conference on Neural Information Processing Systems (NIPS '23), New Orleans, LA, USA, 10–16 December 2023; pp. 80730–80743.
28. Bui, T.; Agarwal, S.; Yu, N.; Collomosse, J. Rosteals: Robust steganography using autoencoder latent space. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, 17–24 June 2023; pp. 933–942.
29. Yansong, G.; Qiu, H.; Zhang, Z.; Wang, B.; Ma, H.; Abuadbba, A.; Xue, M.; Fu, A.; Nepal, S. Deeptheft: Stealing dnn model architectures through power side channel. In Proceedings of the 2024 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2024; pp. 3311–3326.
30. Chuan, Z.; Liang, H.; Li, Z.; Wu, T.; Wang, L.; Zhu, L. PtbStolen: Pre-trained Encoder Stealing Through Perturbed Samples. In Proceedings of the International Symposium on Emerging Information Security and Applications, Hangzhou, China, 29–30 October 2023; Springer Nature: Singapore, 2023; pp. 1–19.
31. Pratik, K.; Basu, D. Marich: A Query-efficient Distributionally Equivalent Model Extraction Attack Using Public Data. In Proceedings of the 37th International Conference on Neural Information Processing Systems (NIPS '23), New Orleans, LA, USA, 10–16 December 2023; pp. 72412–72445.
32. Kariyappa, S.; Prakash, A.; Qureshi, M.K. Maze: Data-free model stealing attack using zeroth-order gradient estimation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 13814–13823.
33. Truong, J.B.; Maini, P.; Walls, R.J. Data-free model extraction. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 4771–4780.
34. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Fei-Fei, L. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.* **2015**, *115*, 211–252. [CrossRef]
35. Wang, D.; Tan, X. Unsupervised feature learning with C-SVDDNet. *Pattern Recognit.* **2016**, *60*, 473–485. [CrossRef]