DisguisedNets: Secure Image Outsourcing for Confidential Model Training in Clouds

KEKE CHEN and YUECHUN GU, Marquette University, USA
SAGAR SHARMA, TikTok, Inc, USA

Large training data and expensive model tweaking are standard features of deep learning with images. As a result, data owners often utilize cloud resources to develop large-scale complex models, which also raises privacy concerns. Existing cryptographic solutions for training deep neural networks (DNNs) are too expensive, cannot effectively utilize cloud GPU resources, and also put a significant burden on client-side pre-processing. This article presents an image disguising approach: DisguisedNets, which allows users to securely outsource images to the cloud and enables confidential, efficient GPU-based model training. DisguisedNets uses a novel combination of image blocktization, block-level random permutation, and block-level secure transformations: random multidimensional projection (RMT) or AES pixel-level encryption (AES) to transform training data. Users can use existing DNN training methods and GPU resources without any modification to training models with disguised images. We have analyzed and evaluated the methods under a multi-level threat model and compared them with another similar method—InstaHide. We also show that the image disguising approach, including both DisguisedNets and InstaHide, can effectively protect models from model-targeted attacks.

CCS Concepts: • **Security and privacy → Domain-specific security and privacy architectures**; • **Computing methodologies → Supervised learning by classification;**

Additional Key Words and Phrases: Outsourced deep learning, confidential computing, image disguising

ACM Reference format:
Keke Chen, Yuechun Gu, and Sagar Sharma. 2023. DisguisedNets: Secure Image Outsourcing for Confidential Model Training in Clouds. *ACM Trans. Internet Technol.* 23, 3, Article 47 (August 2023), 26 pages.
https://doi.org/10.1145/3609506

1 INTRODUCTION

Deep neural network (DNN) training is resource-intensive and time-consuming, requiring large training data, careful model architecture selection, and exhaustive model parameter tweaking, where GPU accelerated training plays a central role. For most model developers, a popular option is to use cloud GPUs or online model training services to access the scale of resources they cannot afford on-premise.

Despite its popularity, outsourcing DNN training to the cloud raises privacy and security concerns about training data and trained models [8, 40] for sensitive applications, such as medical

This research was partially supported by the National Science Fundation (Award# 2232824).
Authors' addresses: K. Chen and Y. Gu, Marquette University, Milwaukee, WI, 53201; emails: {keke.chen, ethan.gu}@marquette.edu; S. Sharma, TikTok, Inc, Bellevue, WA, 98004; email: sagar.sharma@tiktok.com.

1533-5399/2023/08-ART47 $15.00
https://doi.org/10.1145/3609506

image processing [32] and AI-enhanced security [2]. On the one hand, cloud users cannot verifiably prevent the cloud provider from getting access to their data. Using public clouds often means fully trusting your cloud provider. On the other hand, public cloud providers are not immune to security attacks, which may lead to data breaches through various insider [6, 8] and external attacks [29, 47]. Furthermore, leaking trained models is not safe. Various model-based attacks have been reported, such as membership inference attacks [42], model inversion attacks [13], and adversarial example exploration [5, 33]. Thus, it is necessary to protect both data and model for training in clouds.

While challenges remain, there have been a few efforts trying to address the critical issues with confidential training:

- *Encrypted data and models in training.* The classical approach is to use advanced encryption methods, e.g., homomorphic encryption [3], or sophisticated cryptographic protocols, e.g., secure multiparty computation methods [30, 39], to train encrypted DNN models over encrypted data. However, since the scale of training data is often very large and the deep training process takes many iterations, cryptographic model training approaches are too costly to be practical. A recent study on training small-scale neural networks [30, 39] (e.g., just two layers with a maximum of 128 neurons per layer) has shown astonishingly high communication, computation, and storage costs with secure two-party computational protocols. Furthermore, cryptographic approaches cannot utilize GPU resources effectively. A few cryptographic approaches have been proposed to achieve confidential GPU-based training [31, 44, 45], which, however, still cannot fully utilize the power of GPUs, offering much lower GPU speedups than we have witnessed in plaintext-based training.
- *Client-cloud federated learning.* Another solution is to apply cloud-client federated learning [24]. The data owner can partition the training data into sensitive and non-sensitive parts. The non-sensitive portion, assuming it is much larger than the sensitive one to justify the cloud-based training, is exported to and processed by the cloud. Correspondingly, the federated learning process runs the "server" part in data owner's on-premise and the "client" in the cloud. Federated learning framework enhanced with differential privacy [1, 41] may be tweaked into such a partition-based setting. However, some attacks [19] can be applied to the compromised cloud-side program to reconstruct images in the sensitive dataset held by the data owner's on-premise. It is also difficult to identify which part of the training data is indeed non-sensitive and safe for outsourcing.
- *Trusted execution environments.* Hardware-assisted **trusted execution environments (TEEs)**, such as Intel SGX, can also be applied to deep learning in the cloud. The TEE idea is to create a secure enclave in the specific memory area (**enclave page cache (EPC)**) that is enabled by the new CPU TEE features. TEE protects data confidentiality and integrity from attackers who compromise the whole operating system or hypervisor. However, recent studies on side-channel attacks [11] make a side-channel-safe deep learning approach challenging to develop and deploy. Without careful protection, attackers can peek or infer the plaintext content inside the secure enclave via side channels, such as page fault interrupts and cache loading. More importantly, current GPUs do not provide equivalent TEEs as CPUs do. While GPU manufacturers will be developing TEEs for GPU,[1] it is to be tested to determine whether they are resilient to side-channel attacks.

Another seemingly relevant setting is **differential privacy (DP)** [9] applied in distributed (federated) learning scenarios [41] or a trusted central training server [1]. However, DP-related

[1]NVidia has announced a GPU TEE in their Hopper architecture, which is available in the H100 GPU in 2023.

methods aim to share data and models without breaching individual training examples' privacy, using a different threat model that does not demand data and model confidentiality, and thus cannot be used in the outsourcing setting. Therefore, we consider these DP-based approaches are orthogonal to our research.

Scope and contributions. This article focuses on the *image disguising* approach that transforms training-data images on the client side before submitting them to the cloud for training, where the transformed images can still take full advantage of GPUs in the cloud. Specifically, image disguising aims to achieve two goals: (1) the disguised images are difficult to reverse or recognize by adversaries, and (2) they are still learnable by using existing deep learning algorithms without any modification.

In the preliminary work in 2018 [38] and 2021 [37], we reported the initial idea of DisguisedNets. In 2020, Huang et al. [21] proposed a different method, InstaHide, to achieve the same two goals. However, InstaHide has two weaknesses: (1) Its security is not well guaranteed due to the mix-up method [51] it depends on and the information leakage in the mixed-up labels (details in Section 2). Carlini et al. [4] show a clustering-based attack can reconstruct original images from the observed disguised images. (2) InstaHide uses public datasets for the mix-up operation to disguise the original image. The larger the public datasets, the better the privacy is preserved. It is thus not feasible for resource-strapped clients to conduct image-disguising operations. (3) It is originally designed for collaborative multiparty learning, allowing each party to submit training data protected by InstaHide and sharing the trained model among parties. Some InstaHide settings may generate models still working for original images, which does not directly protect the trained model.

In contrast, DisguisedNets utilizes a combination of image blocktization, random block permutation, and **block-level AES-based (AES)** or **random-projection-based (RMT)** disguising methods, which can be efficiently applied to each training image and provide better security guarantee. Our preliminary study has presented the basic idea of RMT and AES methods. However, there are a few critical questions to be answered:

- While the security was preliminarily discussed [4, 37], no comprehensive analysis for DisguisedNets is available. It is unclear under which threat models these methods are still safe to use, which will be critical for practitioners and researchers.
- More interestingly, we can also use image-disguising methods to protect models from model-based attacks, which were not sufficiently studied. Most model-based attacks depend on domain-specific or similar-domain training data to explore exposed models. Image disguising breaks the direct link between original training data and models, which might make most model-based attacks ineffective.
- DisguisedNets methods are much more efficient than existing cryptographic approaches. Furthermore, they do not require extra storage as InstaHide does. However, our previous study has not answered whether resource-strapped platforms, such as mobile phones, can conduct image disguising and how they will impact the expense of server-side model learning.

We have conducted an in-depth analysis to answer the above questions. In summary, this article's unique contributions include:

- We study the DisguisedNets methods under a refined threat model with two levels of adversarial knowledge: known disguised images only vs. known additional pairs of original images and their disguised version (please check Section 3.2 for the detailed definitions). The result shows that both RMT and AES are resilient to Level-1 attacks. RMT becomes vulnerable under Level-2 attacks. While the AES method with a small amount of noise addition

is resilient to Level-2 attacks, its data utility is seriously affected. With this analysis, potential users can confidently adopt these methods under their application scenarios and adversarial assumptions.
- We have analyzed how the disguising methods help protect from the three types of model-based attacks: model inversion, model extraction, and membership inference, under different settings (adversarial knowledge). We also show in experiments that even with adversaries knowing domains and auxiliary data, both DisguisedNets and InstaHide can effectively protect from model-inversion attacks [52].
- We have conducted a thorough complexity analysis and an experimental evaluation to show the expenses of the disguising mechanisms on the client side and the potential impact on the server side.

In the coming sections, Section 2 presents the related work and distinguishes relevant but different studies. Section 3 gives the general image disguising framework under the outsourcing context and the detailed threat model for confidential outsourced training with GPU support. Section 4 describes the details of the DisguisedNets methods and analyzes the complexity, security, and use of DisguisedNets in defending model-based attacks. Section 5 presents the evaluation of DisguisedNets methods to support the formal analysis result.

2 RELATED WORK

Attacks on model training or model inference. Sensitive deep learning assets may include training data, models, and online testing data. Protection methods may target the model training phase or the application (i.e., inference) phase when both phases can be exported to the public cloud. However, the computational complexity and the resource demand of model training is far more than that of model application. Interested readers may refer to recent work on model application, such as CryptoNets [15], CrypTFlow2 [34], and Cheetah [22], which are implemented with cryptographic approaches, e.g., homomorphic encryption and secure multiparty computation. To be clear, the proposed image disguising methods focus on the training phase, for which the pure-software cryptographic approaches are too expensive to be practical.

Crypto approaches for model training. As we have mentioned, due to the high computational complexity of training algorithms and the large size of training data, there is no practical cryptographic approach, e.g., homomorphic encryption or secure multi-party computing-based approaches for protecting model training. A few recent studies in this direction have shown prohibitively high costs even for a small neural network model [30, 39]. TEEs [7] are a more promising approach that depend on CPU hardware supports to speed up confidential operations. While GPU manufacturers have started considering the TEE concept for GPUs, we foresee side-channel attacks that have bothered CPU TEEs will also be major challenges for GPU TEEs. A few CPU-TEE-based solutions utilize masked GPU operations to speed up training [11, 31, 44, 45], which, however, cannot achieve the speedups we observed on GPU-based deep learning solutions for unprotected data.

Image disguising methods. Due to the challenges handling GPU confidentiality, researchers also explore more practical *image disguising* methods with a weaker security notion. We briefly discuss several most-related methods. Fan [10] proposed the differentially private image pixelization method to disguise sensitive objects in images. However, it is difficult to apply, as users have to manually identify the sensitive patches of images, and it does not protect the confidentiality of the whole image. PrivyNet [28] uses a shallow network to transform the original images to feature vectors, which are then exported to the cloud for training. It requires the client to have sufficient computing resources to train the shallow network, the quality of which might affect the cloud-side training. Furthermore, the feature vectors still keep important visual features and

the trained models are exposed to model-based attacks. NeuraCrypt [50] used a similar method but generalized with random encoder networks.

More recently, two methods, DisguisedNets [37] and InstaHide [21], target the visual re-identification attack and aim to provide high data utility. InstaHide [21] mixes up each image [51] in the training set with other images in the same dataset or another public image dataset along with randomizing pixel signs for further protection. The mix-up procedure is defined as a randomly weighted linear combination of the original image and randomly selected images. Assume the mix-up process consists of k images, which includes the target image $X$and $k-1$ randomly selected images $\{U_j\, j = 1..k-1\}$ from the training data and the public dataset. Let $\{\lambda_i\}$ be the randomly generated k weights, $i = 0..k-1$ and $\sum \lambda_i^2 = 1$. The disguised image is $X' = \lambda_0 X + \sum_{j=1}^{k-1} \lambda_j U_j$, and the label $y' = \lambda_0 y + \sum_{j=1}^{k-1} \lambda_j y_j$, where y is the image's original label in one-hot representation and y_j is a non-zero one-hot vector if the corresponding image is from the training data. We will discuss the detail of DisguisedNets and the problems with InstaHide in later sections.

Image transformation methods against adversarial samples. The term "image transformation" is also frequently used in methods against adversarial samples, which, however, is significantly different from image disguising methods. Adversarial samples are often small perturbations of original images in the training dataset, leading to different predictions from the original images' [16, 36]. To reduce adversarial samples or increase the difficulty of finding such samples, some defense methods use augmented training data [17] to improve model robustness and make the model less sensitive to training data perturbation. Data augmentation techniques [43], such as image rotation, cropping, and noise injection, are commonly used to achieve this purpose, which do not aim to hide the image content and thus cannot be used for confidential model training.

3 A GENERAL IMAGE DISGUISING FRAMEWORK FOR CONFIDENTIAL OUTSOURCED DEEP LEARNING

In the following, we will introduce the general image disguising framework first and then discuss the threat model in detail. This section sets up a clear context of confidential learning for analyzing the image disguising methods for outsourced training in general.

3.1 Image Disguising Framework

Figure 1 depicts the general image disguising framework. A data owner disguises her private images before outsourcing them to the cloud for DNN learning. She can either fully outsource the entire image datasets and the learning procedure to the cloud or selectively retain sensitive images in the cloud-client partitioning setting. She transforms all of her images using one secure transformation key secret to her. Note that this transformation should be at a reasonable cost, practical for a client's infrastructure to process.

Specifically, assume the data owner owns a set of images for training, notated as pairs $D = \{(X_i, y_i)\}$, where X_i is the image pixel matrix ($l \times m$ and $3 \times l \times m$ for grayscale and RGB images, respectively) and y_i the corresponding label. We formally define a disguising mechanism as follows: Let the disguising mechanism be a transformation T_K, where K is the secret key that depends on the selected perturbation techniques. By applying image disguising, the training data is transformed to $\{(T(X_i), c_i)\}$ with c_i mapped to $0, 1, \ldots$ randomly representing the classes y_i. The original model is a function $\mathbf{M}(D)$, which is learned with a DNN learning method $\mathbf{G}$: $\mathbf{G}_\theta(D) \rightarrow \mathbf{M}(D)$, with a certain parameter setting θ. The image disguising mechanism enables the same learning method $\mathbf{G}$ to be applied with possible parameter tuning θ' to the transformed data directly without any modification: $\mathbf{G}'_\theta(T(D) \rightarrow \mathbf{M}_T(T(D))$. For any new data X_{new}, the model application (or inference) is defined as $\mathbf{M}_T(T(X_{new}))$, i.e., the new data transformed with the same key. To make such a

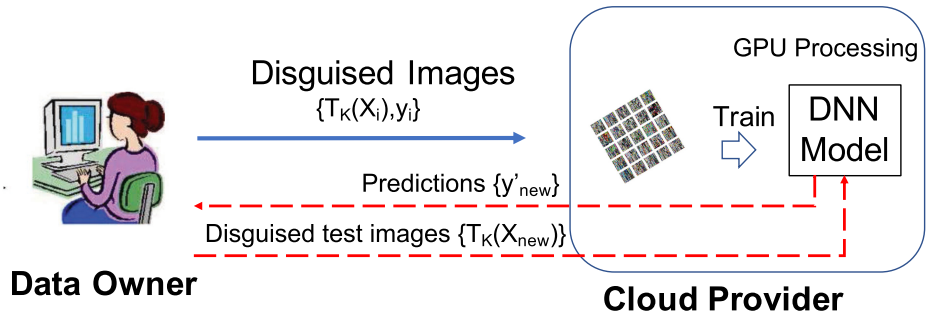


Fig. 1. The general image disguising framework for outsourced deep learning.

transformation method practical for modeling, i.e., a model trained with transformed data still working satisfactorily, a user may expect the error of modeling is not far away from the original model's. Thus, a utility-preserving mechanism should have

$$|Err(\mathbf{M}_T) - Err(\mathbf{M})| < \delta,$$

where δ is the level of model quality degradation acceptable to the user. While for a specific DNN modeling method and a specific dataset, it is difficult to theoretically justify what this gap will be, one can always directly evaluate the model quality to check whether it is acceptable for the application. We have empirically evaluated the δ levels for different mechanisms, datasets, and a few popular DNN modeling architectures in experiments.

Note that the InstaHide method also fits this framework, although originally it was designed for protecting participants' private data in federated learning. However, some InstaHide methods allow the learned model $\mathbf{M}_T$ to be applicable to the original test data, i.e., $\mathbf{M}_T \approx \mathbf{M}$. This property is undesirable for outsourced training, as it leaves the learned models unprotected, vulnerable to model-based attacks.

3.2 Threat Modeling

We are concerned with the confidentiality of the sensitive training image data and the DNN models in the *outsourced training phase*. Here, we make some relevant security assumptions about confidential training.

- Attacking in polynomial time. While training is typically done with a reasonably small amount of time and resources, we may assume adversaries can utilize a much larger amount to break, e.g., a highly sensitive piece of information. However, it is also reasonable to assume the amount of time and resources do not exceed a *polynomial* complexity, as normally assumed in cryptography.
- We consider the cloud provider to be an honest-but-curious adversary, which implies that a curious provider will still honestly deliver desired results to the data owner. However, it may keep a copy of the data and programs it can observe.
- The adversary can observe the training data, the training process, and the trained models, including the structure of the DNN architecture and parameter settings for training. Thus, they can probe the observed items with methods such as image reconstruction, re-identification, and membership attacks.
- We do not address evasive attacks and poisoning attacks [5, 33], where adversaries will tamper with the training data, which can be guarded with training data integrity checking.
- The client infrastructure and communication channels are secure.

Assets to Be Protected. With a disguising mechanism, we generalize that the training data D is transformed to $T_{key}(D)$, and the model $M(D)$ is changed to $M_T(T_{key}(D))$, where the modeling method keeps unchanged but with necessary parameter tuning. Training data can be sensitive for many reasons. The attacker might want to know whether an image is likely used to train a model, e.g., the membership inference attack [42], examine training images that may contain sensitive objects, or steal a proprietary training dataset to build a replica model. The exposed model is also targeted by all kinds of model-based attacks, as we have discussed.

Adversarial Prior Knowledge. The adversary may have two levels of prior knowledge. For each level, we may design a disguising technique.

- *Level-1:* They may know what the model is used for, e.g., the background application, the distribution of the data, e.g., face images, and the type of disguising technique used, but do not know the disguising parameter setting for a specific dataset, which serves as the secret key to the protocol.
- *Level-2:* In addition to Level-1 knowledge, they may try to obtain pairs of images and their disguised versions via other attacking channels (not including the ones they are targeting). They hope to use these known pairs to explore various image reconstruction attacks. A stronger assumption is that the adversary can access an "oracle" that will return a disguised image for any input image.

Potential Attacks. Based on the threat model, we consider two types of attacks.

(1) *Visual Re-identification Attack.* With a protection mechanism on data and models, we consider a basic attack: *training image re-identification*, which aims at matching the disguised images to original images with the Level-1 knowledge only. Note that most older disguising methods [10, 28] are not even resilient to human visual re-identification. Apparently, using human editors to validate this attack is time-consuming and inefficient. Thus, we introduce a model-based re-identification test—DNN examiner, which uses a model trained on the original data to tell whether a protected image is re-identifiable. As most models on the original data are quite capable, comparable to human editors, this method can significantly simplify the evaluation process.

(2) *Reconstruction Attack.* More sophisticated are *reconstruction attacks*, which try to reverse the disguising mechanism to approximately reconstruct the original images and then conduct the visual re-identification attack. The specific attack will depend on the disguising method used. We can also use the DNN examiner approach to evaluate how successful a reconstruction attack works in our experiments.

4 DISGUISEDNETS METHODS

The DisguisedNets methods work under the general image disguising framework. They incorporate pixel-block partitioning, random block permutation, and block-wise transformations of images along with noise additions. The premise is that after the dramatic transformation, it is difficult to link the disguised images to the original images, while, unlike pure encryption schemes, the essential patterns for distinguishing between classes of images still exist, learnable for DNN methods. This amalgam of multiple transformations provides a sufficiently large parameter space so the attacks are computationally intractable (Section 4.4) under the Level-1 knowledge.

4.1 Pixel-block Partitioning and Block-based Random Permutation

In this section, we present one way of image transformation: image block permutation, which will be combined with other mechanisms later.

An image $X_{l\times m}$ is first partitioned into t blocks of uniform size $r\times s$. If we label the blocks sequentially as $v =< 1, 2, 3, 4, \ldots, t >$, then a pseudorandom permutation of the image, $T_\pi(X)$, shuffles

the blocks and reassembles the corresponding image accordingly. Block-based permutation preserves the in-block information and the relative positions of related blocks. Thus, we understand it preserves a great amount of information for effective modeling. However, while the permutation may break the global patterns of the images and achieve good visual privacy already, the between-block characteristics such as boundaries, color, content shape, and texture of the original neighboring blocks may provide clues for adversaries to recover the original image—imagine the jigsaw puzzle! For large t, such attacks can be time-consuming due to the vague similarity between block boundaries. However, with the prior knowledge—knowing an original image and its block-permuted version—it is not difficult to solve such a jigsaw puzzle. Thus, we use this as a preliminary step that will be further enhanced with other steps in the disguising framework.

4.2 Pixel-block Transformations

Next, we establish pixel-block-level protection mechanisms that aim to preserve the data utility for DNN modeling and further increase the resilience to attacks. We consider two candidate mechanisms—random projection and encryption schemes—and discuss their characteristics.

Specifically, when an image is partitioned into t pixel blocks for random permutation, we get a list of t parameters $\{K_i, i = 1 \dots t\}$, one for the pixel-block at the same position across the whole dataset. For example, K_i contains the block-wise random projection matrix for the RMT method and the block-wise AES encryption key for the AES method. We name the specific position of the pixel block in the image *the pixel-block position*. The list of parameters acts as a secret key and will be shared, together with the permutation key, by each image in the dataset. The purpose of this setting is to maximize the preservation of distinguishable patterns between image classes—i.e., a pair of similar image patterns (blocks) can still be transformed to another pair of (likely) similar ones after applying the disguising mechanism.

Randomized Multidimensional Transformation (RMT). For an image represented as a pixel matrix X, a general linear transformation can be defined as $G(X) = R(X + \Delta)$, where $R_{m\times m}$ is a random orthogonal matrix generated following the Haar distribution [14], or a random invertible matrix, e.g., a random projection matrix [49], and Δ is an optional noise matrix. We call this method the randomized multidimensional transformation. When an image is partitioned into t blocks for random permutation, we prepare a list of random matrices $\{R_i, i = 1..t\}$, one for each image-block position, and share this list for each image. Such transformation is known to preserve (or approximately preserve by random projection) the Euclidean distance between columns of the matrix X. For real application, we may arrange the pixel blocks accordingly to form the column of X. For example, a 4×4 pixel matrix can be partitioned into 4 2×2 block to preserve the smaller block-level similarity with RMT. Figure 2 shows the effects of RMT on MNIST and CIFAR-10 datasets.

AES Block Transformation (AES). The existing AES encryption schemes typically use 128-bit encryption keys, which encode every 16 bytes of data sequentially. If we use AES for pixel-block encryption, assuming each pixel is stored in one byte, then 16 original pixels are mapped to 16 encrypted bytes (pixels), and a whole pixel block is encoded to 16-byte units. Putting all encrypted pixel blocks together, we get a disguised image. For clear presentation, we use "AES encryption unit," i.e., by default 16 bytes, to distinguish from the "pixel block" used in partitioning and permutation in our image disguising framework. Each pixel block at the specific position of the image will use a distinct AES key to ensure better security, while the AES key at the same position will be used for different images to preserve the locality information across images. Figure 3 shows some example AES transformations on images.

As AES has multiple modes, we have carefully checked different modes and decided one in our design based on the trade-off between data utility and security. (1) With the AES **Cipher Block Chaining (CBC)** mode, any pixel-level change in the pixel block between two images will result

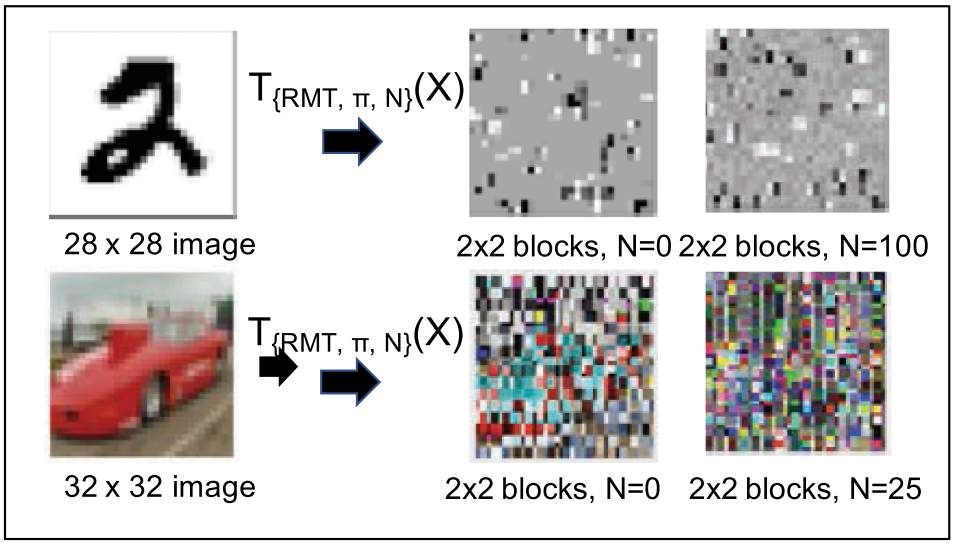


Fig. 2. Block-wise RMT+Noise on MNIST and CIFAR-10 images.

ALGORITHM 1: DN_RMT (X, t, Key)

Require: X: image of size $l \times m$; t: number of blocks; Key = {permutation_key, transformation_matrices, noise_level $\in$ [0, N]}

r, s $\leftarrow$ compute image block size with $l \times m$ and t;
Partition image $X_{l \times m}$ into blocks $X_1, X_2, \ldots, X_t$;
Shuffle the image blocks pseudorandomly with permutation_key
for each block $i, i = 1 \ldots t$ **do**
    $\Delta_i \leftarrow$ Generate random matrix with elements from the uniform distribution in [0,N];
    use the transformation matrix at the position i: R_i;
    $Y_i \leftarrow R_i(X_i + \Delta_i)$;
end for
Re-assemble $\{Y_i\}$ to make the transformed image Y and return Y;

in different encoding results. This feature makes two similar blocks very different after encoding, seriously damaging data utility, and thus it is not ideal for our purpose. (2) Then, we turn to the AES **Electronic Code Book (ECB)** mode to preserve more utility. ECB is considered as a fixed mapping function between 16-byte original data to 16-byte encrypted data. Different from CBC, the neighboring 16-byte blocks do not affect the encoding of the current block. This matches our requirement of data utility preservation, e.g., to preserve the block-level distinguishable patterns after the transformation.

In the initial design, when the pixel block size is larger than 16, we can use 16 encryption units. One may wonder how to handle the block size < 16, which might preserve more information. We design a scaling-up-down procedure for handling this setting. Specifically, we can scale up the block to 16-byte size first, i.e., duplicating pixels in a pixel-by-pixel way. For example, for a 2×2 block, each pixel is duplicated along the x axis and y axis, respectively, to get a 4×4 block, which is then encrypted. Finally, we can choose to scale down the whole image to the original size by merging the neighboring pixels. This procedure can also be applied to pixel blocks larger than

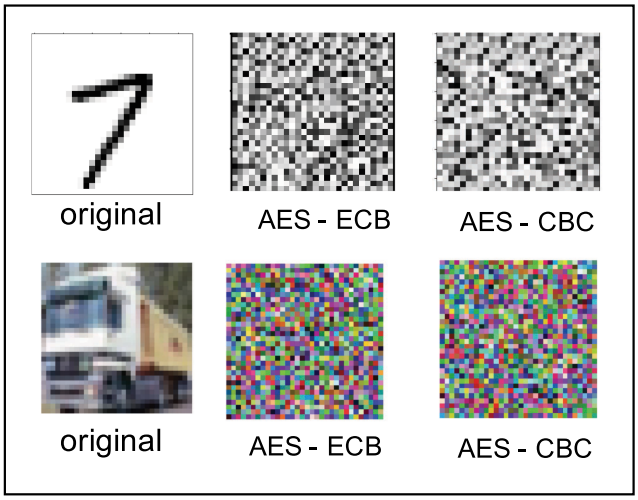


Fig. 3. Pixel-block-based AES encryption of MNIST and CIFAR-10 images.

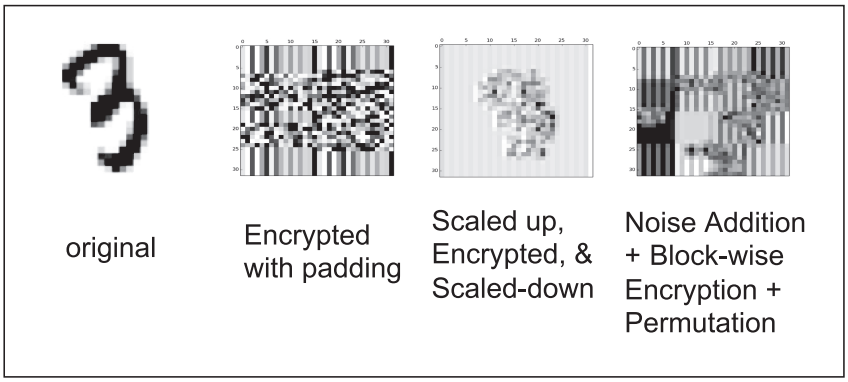


Fig. 4. AES-ECB encryption of MNIST image with different strategy.

16 pixels—by changing the encryption unit to be less than 16. In experiments, we found smaller encryption units actually improve model quality for the AES method. Algorithm 2 shows the AES method without the scaling-up-down procedure, and Figure 4 demonstrates how the result looks like for different settings of the AES method.

4.3 Complexity Analysis

The additional costs of the disguising methods are twofold: the client-side encoding cost and the server-side additional learning cost. The client-side encoding cost can be critical for both large data holder and small-batch data contributor with limited computing and storage space, e.g., submitting data from mobile phones. On the other side, the model trainer may also wonder whether the server-side learning can converge, and if it can converge, whether the disguised images will affect the convergence speed. We leave the second part to the experimental evaluation and analyze the client-side encoding cost here.

For clear presentation, we assume the image size $m \times m$. For schemes involving partitioning, we assume the image is partitioned into t^2 blocks with block size $m/t \times m/t$. The cost of RMT

Table 1. Per-image Client-side Encoding Costs

	InstaHide	RMT	AES
time	$O(km^2)$	$O(m^3/t)$	$O((ms)^2/16)$
basic ops.	multiplication	multiplication	encryption
storage	auxiliary data	O(1)	O(1)

Parameters: image size $m \times m$, k-image mix-up for InstaHide, $t \times t$ block partitioning in RMT and AES, and scaling factor s for AES.

ALGORITHM 2: DN_AES (X, k, t, Key)

Require: X: image of size $l \times m$; k: scale-up factor; t: number of blocks; Key = {permutation_key, AES_keys, p = probability of salt-pepper noise}

Ensure: the selection of k and t results in image blocks that can be further partitioned to 4×4 pixel patches;

1: $X_{lk \times mk} \leftarrow$ scaled up the image;
2: r, s $\leftarrow$ compute image block size with $lk \times mk$ and t;
3: Partition image $X_{l \times m}$ into blocks $X_1, X_2, \ldots, X_t$;
4: Shuffle the image blocks pseudorandomly with permutation_key;
5: **for** each block $i, i = 1 \ldots t$ **do**
6: $Y_i \leftarrow$ for each pixel in block X_i, with probability $1-p$ it is unchanged; otherwise randomly turned to white or block pixel (salt-pepper noise);
7: $E(Y_i) \leftarrow$ every 16 bytes is encrypted to 16 bytes of AES digest with AES_key_i;
8: **end for**
9: re-assemble image blocks $E(Y_i)$ and return $E(Y)$

encoding per image is dominated by t^2 matrix multiplications with each costing $(m/t)^3$. Thus, the total cost is $O(m^3/t)$—the more blocks, the lower costs. With the same setting, if the block size $(m/t)^2 \geq 16$ (assuming times of 16 for simplicity), then the AES-128 encryption cost is $(m/t)^2/16$ AES encryption operations per block and thus the overall cost is $O(m^2/16)$ encryption operations. For $(m/t)^2 < 16$, the scale-up-down process uses a scaling factor s and the final cost is $O((ms)^2/16)$ encryption operations. Our experimental evaluation shows that per image cost relatively low and can be comfortably done by any PC or mobile phone—lower than 5 milliseconds per image for the sample datasets.

In contrast, while the computational cost of InstaHide is also relatively low, it requires accesses to a large auxiliary dataset, such as ImageNet, to achieve good confidentiality. For k images involved in the mix-up process, the overall cost is $O(km^2)$. The accesses to the auxiliary dataset have to be carefully handled to ensure security and reasonable storage or communication costs. Table 1 summarizes the per-image time and space complexity for applying these methods. The cost scales linearly with the number of images in the training dataset.

4.4 Attack Analysis

This section aims to analyze the possible threats to the proposed disguising mechanisms and clarify the applicable settings. With Level-1 adversarial knowledge, DisguisedNets mechanisms provide strong confidentiality protection, as shown in the discussion of "brute-force attacks" and "clustering attacks." Other simpler methods are struggling with visual re-identification by human eyes [10, 28], while InstaHide is broken with only the Level-1 knowledge by a clustering-based method [4]. However, DisguisedNets methods are still vulnerable to more sophisticated reconstruction attacks that depend on Level-2 adversarial knowledge. This study clearly states the

settings that DisguisedNets can be applied and distinguishes the security features of the DisguisedNets methods from InstaHide's.

4.4.1 Level-1 Adversarial Knowledge and Attacks. Recall that Level-1 knowledge includes knowing only the disguised images and possibly the model domain, i.e., the background application. This assumption matches practical settings that users want to protect training from curious adversaries in the cloud environment. It is clear that with only Level-1 knowledge, the brute-force attack on the AES-encrypted cipher is not possible, and thus we focus on the RMT scheme.

Visual Re-identification. The first simple attack is to visually identify images by human attackers. We have proposed the *DNN examiner* approach to efficiently conduct this evaluation, instead of slow human-editor-based checking. The result will be reported in the experiments.

Brute-force Parameter Estimating. Assume the attacker knows that RMT is used. The brute-force attack method for image reconstruction is to enumerate each possible parameter setting of the disguising mechanism and then check the recovered result with visual re-identification. As AES encryption is already resilient to the brute-force attack, we examine the RMT method only. Let us start with a block-level transformation for any image block i with RMT. Let us look at a simpler setting: $X'_i = R_i X_i$ without the noise component. The adversary knows only X'_i. In the brute force attack, the number of possible X_i is determined by the number of possible R_i matrices. We show that the number of possible R_i can be exponentially large for given parameters.

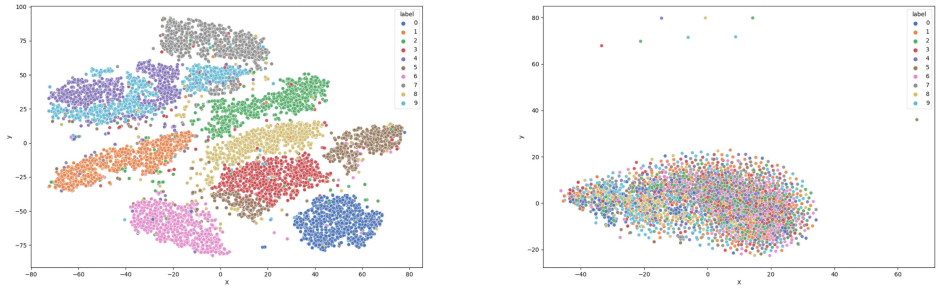
Proposition 1. *For values encoded in h-bit finite field, there are* $O(2^{hm^2})$ *invertible matrices* $R_{m\times m}$.

The proof is based on the theory of matrices in a finite field of order q [18], where $q = 2^h$ for h bits. There are $(q^m - 1)(q^m - q)..(q^m - q^{m-1})$ invertible $m \times m$ matrices in such a field. With a small setting $h = 8$ and $m = 4$, the overall complexity has reached $O(2^{128})$, equivalent to a 128-bit security. Combined with the random permutation of blocks, the attack complexity is even higher. Thus, a brute-force attack is generally impractical.

Clustering Attack. Certainly, a successful attack on images does not need to exactly recover the whole image. As long as the sketch of image is reconstructed, human vision can effectively detect the image content. Due to the utility preserving nature, there might be some features in the disguised images that can help the attacker. For example, Carlini et al. [4] proposed a clustering method to attack InstaHide [21]-disguised images. The attack assumes, with a sufficiently large number of training epochs, most images in the training data will be used for multiple times in generating disguising images, which likely forms a cluster of disguised images that can be used to de-mask and de-noise. As InstaHide disguised images are essentially linear combinations of plaintext images, the attack result can be visually re-identified.

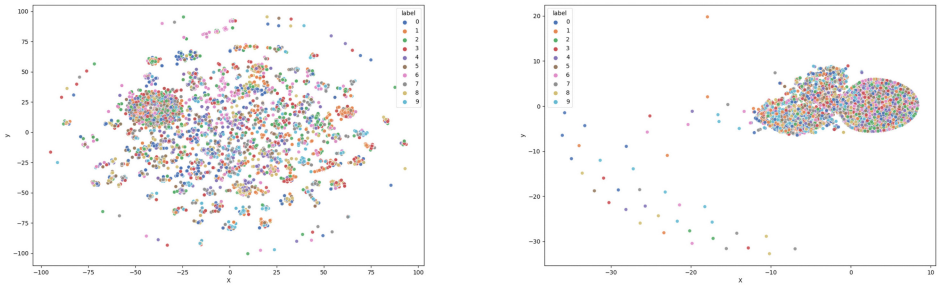
Important questions are whether RMT and AES methods can generate images with clustering structures and whether such clusters can be used to break the disguising methods. To answer these questions, we visualize the disguised images, each of which is converted to a vector, with t-SNE [48] to understand the existence of clustering structure in the Euclidean-distance space. Figure 5 shows that RMT might preserve the clustering structures for some datasets like MNIST and FASHION, if the original datasets contain such clustering structures. In contrast, AES does not preserve any clustering structure, as shown in Figure 6.

Can such RMT-preserved clustering structures be used for attacks? So far, the only risk seems the matching between the clustering structure with the original dataset, which can potentially help attackers verify whether a target dataset is used for training. For example, cluster-cluster distance and cluster size distribution are all useful information for matching. If the matching works, then the attacker does not need to breach the confidentiality of *individual* image, but uses the identified dataset directly for other attacks, such as membership inference and model extraction attack.



(a) RMT on MNIST (b) RMT on CIFAR10

Fig. 5. t-SNE visualization of RMT disguised datasets (4×4 blocks). Colors represent different labels. A dense area covered with one color means that the clustering structure matches the label distribution well for the specific subset.



(a) AES on MNIST (b) AES on CIFAR10

Fig. 6. t-SNE visualization of AES disguised datasets (4×4 blocks).

4.4.2 Level-2 Adversarial Knowledge and Attacks. We move one more step further to study the more challenging issue: What if a powerful adversary can obtain additional knowledge: pairs of original images and their transformed ones? This assumption corresponds to the chosen-plaintext attack in cryptographic analysis [25]. Below, we focus on the *codebook attack* on the AES-ECB-based disguising mechanism and the regression-based attack on the RMT mechanism. This study helps us understand when we should not use the proposed DisguisedNets method.

Codebook Attack. The assumption is that the adversary is knowledgeable of the encryption procedure described previously but does not have the encryption/decryption key. Since the AES ECB method is deterministic, the basic attack utilizing the level-2 knowledge is to build the mapping (i.e., the codebook) between the plaintext unit (e.g., the 16-byte pixel block) and its encrypted counterpart (e.g., the 16-byte AES cipher block). By processing the known image pairs, the adversary constructs a codebook as a dictionary mapping 16-byte pixel blocks to encrypted 16-byte blocks. Since different images, especially those in the same class, might share some 16-byte pixel blocks, some 16-byte encrypted blocks in the targeted images are likely already in the codebook, which will be used to recover the original blocks. For encrypted pixel blocks not present in the codebook, the adversary may use a fixed pattern, e.g., all zero values or most likely values to pad. By repeating this procedure for each 16-byte block, the adversary can recover some parts of the image, which can be simply re-identified via human eyes or trained image-recognition models at the adversary's hands.

Projection Matrix Estimation Attack. Note that noise addition can easily defend the RMT method from the codebook attack, which is already a part of the RMT method. However, if the adversary has obtained enough original and transformed image pairs, then there is a possibility that the transformation matrix might be estimated with linear regression. Specifically, a noise-added block-wise transformation, e.g., $Y_i = R_i(X_i + \Delta_i)$, where Δ_i is a random noise matrix, re-generated for each image block X_i, and drawn uniformly at random from $[0, N]$, where N is the tunable noise level. With enough known pairs of (X_i, Y_i), the regression method can be applied to estimate R_i. Generally, the more known pairs, the more precise the estimation can be. However, it is unclear how the noise level affects the effectiveness of estimation and how we can achieve a good balance between data utility and attack resilience. We will examine the regression-based attacks in the experiments.

Possible mitigation methods to Level-2 attacks. Because the above attacks depend on the knowledge of pairwise examples (the original and the disguised image pairs), one may wonder whether breaking the certainty of the mapping between the pairs will help. For example, random noise injection can be applied to both RMT and AES. Intuitively, a simple salt-pepper noise addition can affect the codebook attack on AES, e.g., making the pixel-block mapping non-unique: The same 16-byte pixel block with different noises can be mapped to different ciphertexts. However, how effective this defense can be should be examined in experiments.

4.5 Model Protection via Image Disguising

Note that the models trained with disguised images work only on disguised images.[2] This property also protects models from existing model-based attacks, including model-inversion attacks [12, 52], membership-inference attacks [20, 42], and model-extraction attacks [23, 46]. In the following, we analyze whether and how the proposed image disguising methods can also protect from these model-based attacks.

These model-based attacks are possible only when the model can be accessed by the attacker. Thus, we consider two model exposure scenarios.

(1) Model exposure in training, i.e., models are exposed at the end of training in the cloud. Since a model trained on disguised images cannot be directly used, such a model exposure is almost immune to the model-based attacks.
(2) Model access API exposure during model application. After the model is trained, the prediction API might be an open service to the public; or the attacker breaches the access control system to access the private model API. With image disguising, the model owner will operate a secret image transformation component, as shown in Figure 7, which employs the secret key for disguising to transform an input image. When an attacker gains the access to such an API service, all model-based attacks can still be applied—the attacker can treat the model as a black box.

Since our focus is on confidential model training, we will skip the discussion about model access API exposure. In the following, we will discuss how the disguising methods can protect from model exposure in training.

A *model-inversion attack* uses a learning procedure, e.g., a GAN method [52], to progressively adjust randomly generated or seed images from similar domains towards most likely training examples. This type of attack needs to access the model freely. With an exposed model, the model-inversion attack recovers only the disguised training data, not the original data. Thus, if the

[2]Some InstaHide models also work on non-disguised original data [21]. Thus, they do not have the benefits of model protection.

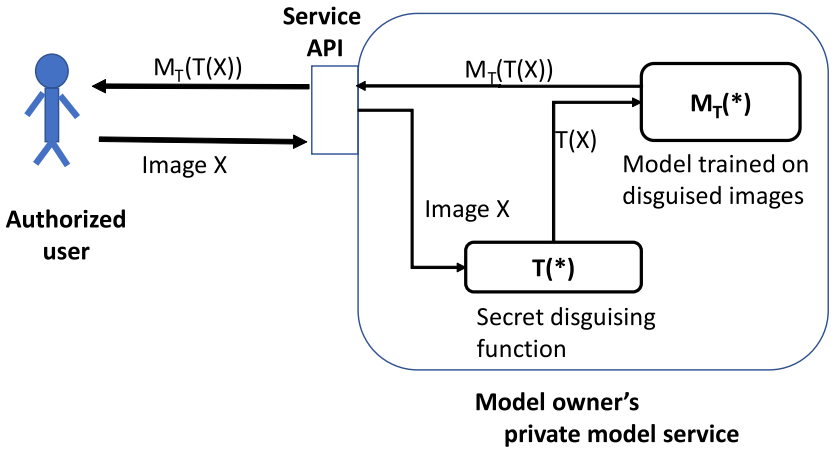


Fig. 7. Model owner hosts the API service secretly in house or using a trusted execution environment in the cloud.

disguised training data is secure, then model-inversion attacks are not meaningful. We will show how disguising methods protect from this kind of attack in experiments. The attacker can use the service API like accessing a normal model; how internal data transformation happens does not interfere with the attack. Thus, the disguising methods can also be an effective defensive mechanism against model-inversion attacks.

A ***membership-inference attack*** **(MIA)** estimates the possibility of a target sample belonging to the training data of a model. This attack assumes the adversary knows the type and distribution of training data [42] and tries to estimate the probability of using a target sample in model training. Most such attacks look at the model's output probability vector for the target sample to distinguish in- or out-training samples. Since the disguised model is trained on the distribution of transformed data, without transforming the target sample with the secret disguising parameter, the output probability vector more likely indicates original images to be out-training samples.

A ***model-extraction attack*** **(MEA)** assumes the attacker can freely access the model to collect the (input, predicted-label) pairs, which are then used to reconstruct the model. MEA is often applied to paid model API to reconstruct a free model offline. The model trained on disguised images effectively deter this attack, as the attacker cannot forge effective input data, which are generated with the secret disguising function.

Therefore, we conclude that image disguising can also effectively protect exposed models during training.

5 EXPERIMENTS

The experiments have three goals: (1) Understand the costs of the image disguising methods on the client side and server side, respectively; (2) Explore the effects of parameter settings on utility and confidentiality; and (3) show that image disguising can also effectively defend against the model-inversion attack—a representative model-based attack.

Setup. All experiments are done on an Intel Xeon server with Nvidia GPU V100 available for training models. The image disguising methods are implemented with Python, and we use PyTorch for model training implementation.

Compared Methods. In addition to the proposed RMT and AES methods, we also evaluate InstaHide in some experiments with InstaHide's published code [21]. Cryptographic approaches

Table 2. Datasets and Baseline Accuracy

Datasets	Records	ImageSize	Network	Baseline Accuracy
MNIST	(60k Tr, 10k Te)	{28 × 28}	AlexNet	96.7 ± 0.2%
FASHION	(60k Tr, 10k Te)	{28 × 28}	AlexNet	88.7 ± 0.3%
CIFAR-10	(50k Tr, 10k Te)	{32 × 32}	ResNet-18	93.4 ± 0.2%
LFW	(1,164 Tr, 292 Te)	{60 × 48}	ResNet-18	94.3 ± 2.0%

Tr: Training, Te: Testing.

for training [35] also achieve confidential training. However, they are still too expensive to be practical. Our previous study [39] shows that training a shallow small network with SecureML [30] already takes hours to days. Thus, we do not include them in comparison.

Measures. We used different measures for tasks in the evaluation. *Model Quality* means the classification accuracy. *Attack Success Rate* is the result of a DNN examiner's classification accuracy on disguised (or reconstructed) images, which is considered as a (loose) upper bound of human visual attack, as DNN examiners can often pick up features that human eyes cannot. We also normalize this accuracy with the original classifier accuracy to be more comparable for different disguising methods and datasets.

$$AttackSuccessRate = \frac{\text{accuracy of DNN examiner on disguised/reconstructed images}}{\text{accuracy of DNN examiner on original images}}$$

Hit Rate is specifically used in evaluating the codebook attack on the AES method, which is the proportion of the 16-byte blocks in the disguised images that can be found in the codebook. A higher hit rate may imply a higher attack success rate. Both rates will be presented with percentages.

Datasets. We use four prevalent DNN benchmarking datasets: MNIST, FASHION, CIFAR10, and LFW [27] for experiments. MNIST (handwritten digits) and FASHION (fashion items) are gray-scale 28 × 28-pixel images with 10 classes. CIFAR10 has 60k 32 × 32 color images distributed into 10 classes. LFW is a relatively small labeled face database with only a few thousand samples. For experiments involving randomness, we repeat five times to estimate the standard deviation. Table 2 summarizes the datasets, the types of deep neural networks used to train the base models, and their baseline model accuracy on the original image data.

5.1 Expense of Disguising Mechanisms

As we discussed, the costs of disguising mechanisms are twofold: the client-side encoding and the server side additional training time. We have briefly analyzed the per-image disguising cost in Section 4.3. Figure 8 shows the average costs of the RMT, AES, and InstaHide methods for random batches of 100 images from the four datasets, respectively. All methods finish within 100–500 milliseconds. While a typical mobile processor might be much slower than the Intel Xeon CPU that we use for experiments, we expect the actual encoding costs for mobile processors will not be too high.

The server side additional training cost is another major concern. Since the standard deep learning software is used in training disguised images, the per-epoch cost is only affected by the image size and the architecture complexity, regardless of how the image is encoded. The major cost difference comes from the number of additional epoches required to converge. Figure 9 shows the evaluation of convergence speed on MNIST and CIFAR10 for the methods: baseline, RMT, AES, and InstaHide. The baseline refers to the models reported in Table 2. Both RMT and AES run with the basic setting of 4 × 4 pixel blocks without noise addition. The scale-up-down method is not applied to AES. The result shows that training with the RMT-disguised or InstaHide disguised images does not affect the convergence speed, and we have seen the same convergence pattern as the regular

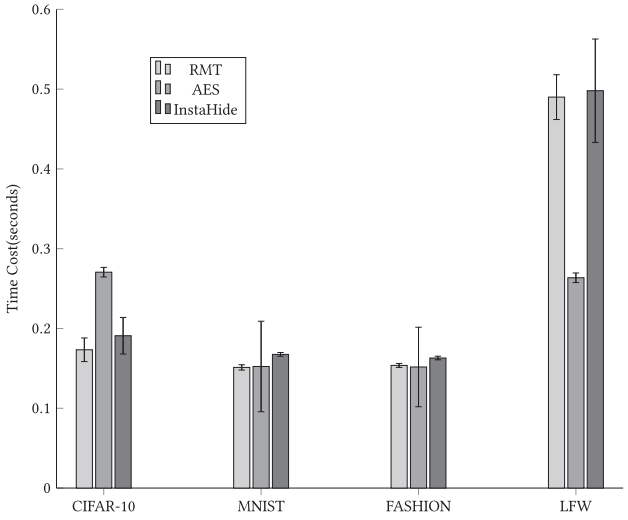


Fig. 8. Average encoding costs for 100-image batches.

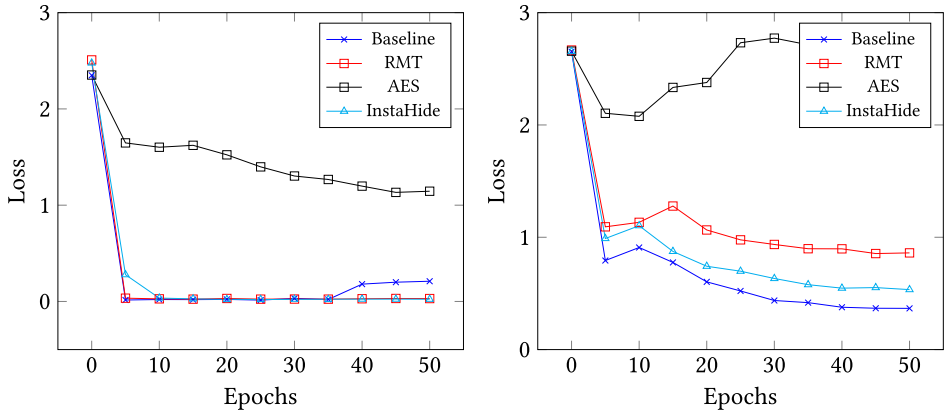


Fig. 9. Convergence speed on disguised images. Baseline: models trained on original datasets.

training for both datasets. However, the AES-disguised images make the convergence difficult. It converges around 50 epochs for MNIST, compared to 10 epochs for other methods; its convergence pattern is unstable for CIFAR10. In addition, the convergence quality for AES-disguised images is significantly affected. Both figures show much higher losses than other methods.

5.2 Under Level-1 Attacks

As we have discussed, RMT and AES are resilient to Level-1 attacks that aim to recover parameter settings. However, it is unclear how they are resilient to the basic visual re-identification test. In the following, we will evaluate the resilience of RMT, AES, and InstaHide to visual re-identification and then focus on the effect of parameter settings for RMT and AES.

Visual Re-identification. The test on visual re-identification is done with the DNN examiners trained on the original datasets, respectively. The intuition is, if the image was not sufficiently

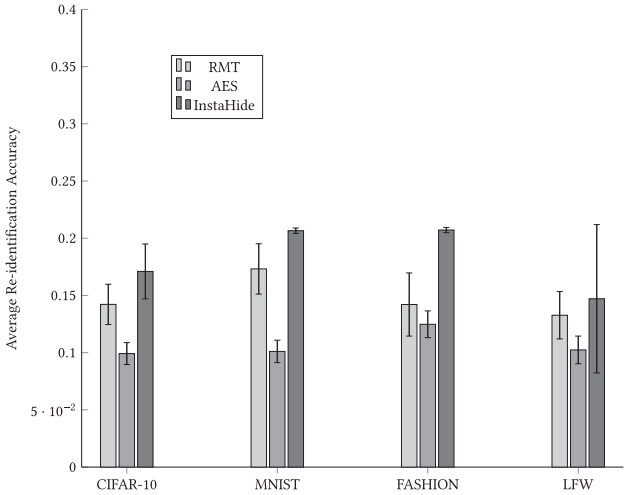


Fig. 10. Visual re-identification test for different methods, performed on 100-image random batches extracted from the testing data.

disguised, then its content would still be visually recognized by a human attacker that is implemented with the DNN examiner. We can use the DNN examiner's accuracy on disguised (or reconstructed) images to represent the level of re-identifiability, i.e., the *Attack Success Rate*, as we defined earlier. Since DNNs are better than human especially for detecting images from heavily noisy and distorted images [26], we believe these attack success rate numbers should be generally higher than what a human attacker can do. Figure 10 summarizes how resilient these methods are to visual re-identification attacks, where both RMT and AES use the block size 4×4 without noise addition. The result shows all methods are resilient to visual re-identification.

Effect of RMT Parameter Settings. Block size and noise level settings of RMT may affect the quality of models trained on RMT-disguised images. For easier presentation, we convert block size into the number of blocks: 1 block on the x-axis means the image is not split into blocks; while 196 blocks means 196 2×2 blocks for 32×32 images (CIFAR10) or padded 28×28 images (MNIST and FASHION), and 196 4×3 blocks for padded 60×48 images (LFW). Thus, a smaller block size results in a larger number of blocks after partitioning, as the image size is fixed. If more than one block is generated in partitioning, then we also apply a secret block-wise permutation. Figure 11(a) shows that the model quality is slightly decreased with smaller block sizes (more blocks per image). Overall, the model quality is well preserved, only 2%–3% worse than the baseline. Figure 11(b) uses normal noise distribution $(noise\_level) * N(0, 1)$ to generate the per-pixel noise. It shows that more sophisticated images, such as CIFAR and LFW are more sensitive to noise addition.

Effect of AES Parameter Settings. Block size is the only parameter for the AES method. Figure 12(a) shows different block size settings from 1 block (e.g., 32×32 per block for 32×32 images) to 64 blocks (e.g., 4×4 pixels per block for 32×32 images). We tested two schemes: no scaling vs. scaling. The no-scaling scheme uses the block size ≥ 16 bytes, while scaling can use even smaller block sizes. Specifically, when we use a block size < 16, e.g., 2×2 blocks, the scaling-up factors are determined for the x and y axes, corresponding, e.g., the scaling factor for x-axis is 2 and also 2 for y-axis for 2×2 blocks, so we can partition the scaled image with 4×4 blocks. Figure 12(a) shows that the model quality is much lower by the no-scaling scheme. For some datasets, e.g., CIFAR10 and LFW, the model quality is too low to be used. Figure 12(b) with a small block size 2×2 and the scaling-up-down method shows that the model quality is boosted to the level comparable to

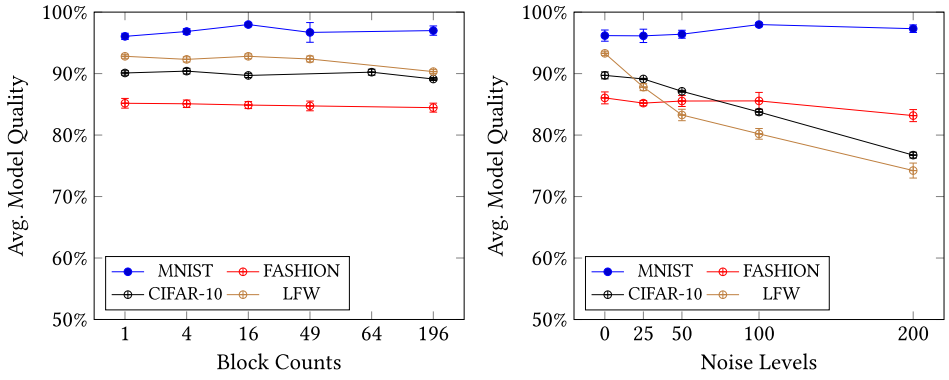


Fig. 11. Effects of block size (left) and noise level (right) on model quality for RMT-disguised images.

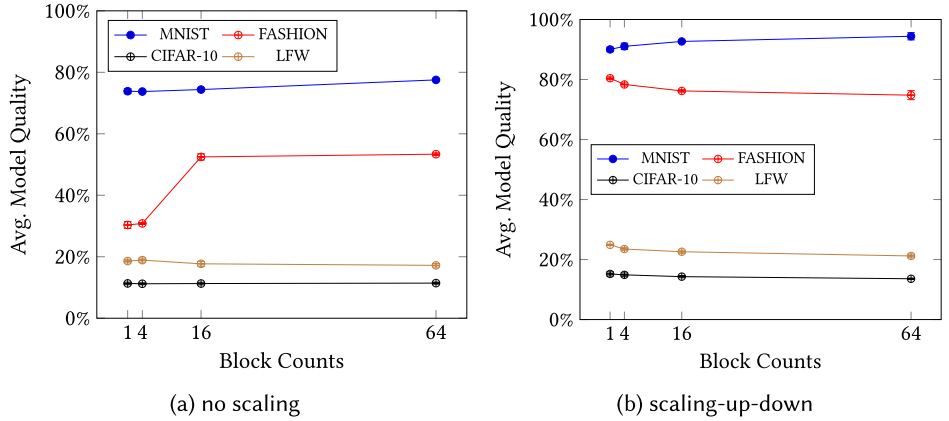


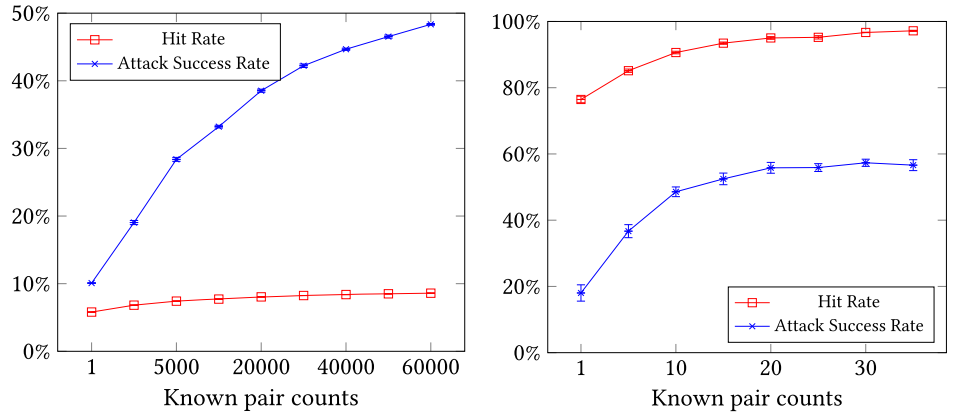
Fig. 12. Effect of block size on model quality for AES-ECB-disguised images

the RMT's results for MNIST and FASHION, while the other two still stay at unusable levels. The possible reason is that the colored (multi-channel) images contain more noisy image blocks, which change significantly after the AES transformation. Thus, different from the RMT scheme, the AES scheme may only preserve utility for some datasets.

5.3 Under Level-2 Attacks

With the known additional knowledge, i.e., pairs of original and disguised images, the disguising mechanisms might be under the reconstruction attack, and attackers can visually check the reconstructed images to re-identify the features of original images. The re-identification step after reconstruction still uses the DNN examiner to generate the attack success rate.

AES under Codebook Attack. Assume the attacker knows k pairs of original images and their ECB encrypted ones with other necessary knowledge such as block sizes. The codebook attack uses the known pairs to construct a mapping between the known plaintext 16-byte pixels (or a reduced number of pixels if the scaling up/down method is used to preserve more utility) and the corresponding encrypted 16-byte pixels. The attacker might be able to use the codebook to partially recover the original pixel blocks of a disguised image (with random pixel patches for unrecognized blocks).



(a) 16-pixel encoding unit (b) 2-pixel encoding unit (2x1)

Fig. 13. Codebook attack on MNIST dataset with varying number of known pairs.

As MNIST and Fashion perform reasonably well with the AES scheme (Figure 12), compared to the other two, we pick only the MNIST data for clear presentation—the Fashion data has a similar pattern. Figure 13 compares the attack results on 16-pixel encryption units (subfigure (a)) and 2-pixel encryption units (for 2×1 blocks) with scaling (subfigure (b)). The attacker's known pairs are selected randomly from the training data, while the targeted images are selected from the testing data. 16-pixel encryption unit gives a one-to-one mapping between the original pixel units and the encrypted ones. We observed hit rates are quite low (lower than 10%), but success rates are increasing steadily due to the increased codebook size. Overall, attackers will need a large number of pairs to achieve a good success rate. 2-pixel encryption unit may create a one-to-many mapping between original pixel units and the encrypted ones, due to the scale up/down processes. We used the Python library function for image scaling. With the scaling process, we observed that hit rates initially increase to around 10% and then drop to 2%–3%. However, the success rate quickly reaches the plateau—around 50% with only 20 image pairs. Therefore, the no-scaling method is more resilient to attacks, as both the hit rates and success rates grow slowly, and knowing the whole training data does not help improve the success rates much. In contrast, the scaling method can help gain better model quality, while it also makes the method more vulnerable to Level-2 attacks.

Aiming at achieving a better balance of utility and attack resilience for the setting of the 2-pixel encryption unit, we test the use of "salt-and-pepper" noises to disturb the codebook attack. The AES encrypted pixel block changes dramatically when any of the original pixels change, which helps reduce the attack success rate. Figure 14 shows that by adding a small amount of noise, e.g., 2%–3%, the attack success rate drops by 10%, while the model quality is not significantly damaged. Certainly, the level of noise should be carefully chosen to avoid destroying the data utility: An increase of noise intensity to 4% will dramatically degrade the model quality, as Figure 14(b) shows.

RMT under Regression Attack. As we have shown, the regression attack can be a practical threat to RMT if the attacker is equipped with Level-2 knowledge. We also assume a stronger attack scenario: The attacker already knows the pixel-block size and the permutation pattern. We know that RMT without noise addition is very vulnerable to known pairs. With as little as one pair of known images, the block-wise transformation parameters $\{R_i, i = 1..m\}$ can be recovered using simple matrix inversion. Thus, we consider how the noise addition can help improve RMT's resilience to attacks. Different from the "salt-and-pepper" noise for selected pixels in the enhanced

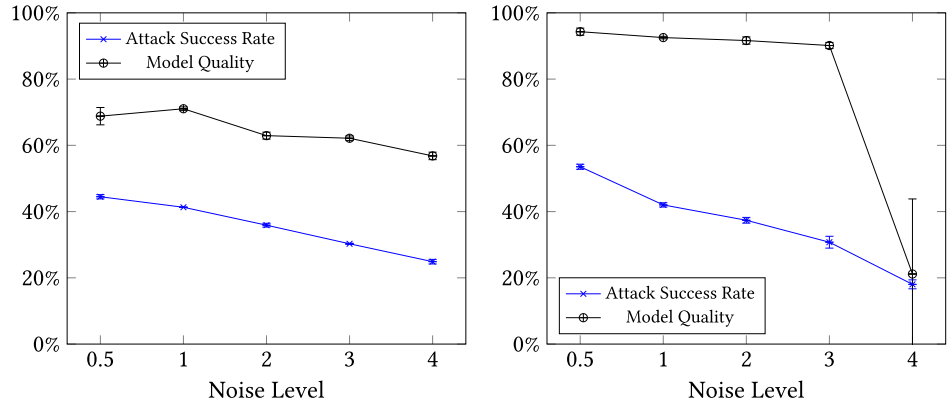


Fig. 14. Protecting AES-based disguising with noise addition (MNIST data)

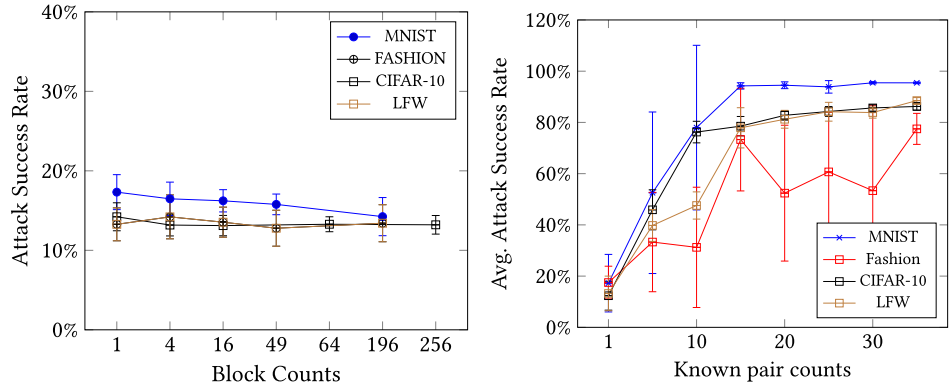


Fig. 15. Attacks on RMT-disguised images: Direct re-identification does not work on RMT, but regression attacks on the noise-added RMT disguising method still work with enough known pairs. Images with block-size 7×7 and noise level $u = 100$

AES scheme, we generate a noise value for each pixel and add it to the original pixel value before applying the projection, i.e., $Y_i = R_i(X_i + \Delta_i)$, where the noise Δ_i is drawn from the uniform distribution $U(0, u)$. With noise addition, the regression attack is used to estimate the parameters $\{R_i\}$, the accuracy of which might be affected by the noise intensity (i.e., the u setting) and the number of available pairs.

Figure 15(a) shows that direct re-identification (with Level-1 attack) is generally not effective at all. However, Figure 15(b) shows that the regression attack is highly effective on all datasets. With a small number of known image pairs, the attack can achieve surprisingly high success rates. Thus, it is not safe to use the RMT scheme when Level-2 attack knowledge is available.

5.4 Use Image Disguising to Protect Models

Exposing models may have high risks, as shown in model-inversion attacks, membership-inference attacks, and model-extraction attacks. This experiment shows that image-disguising methods can

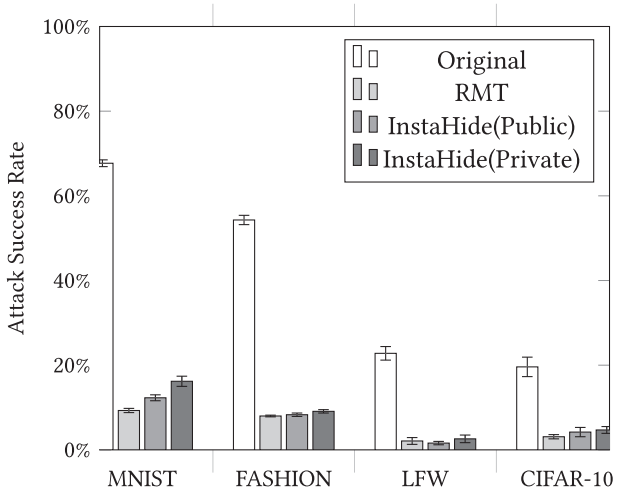


Fig. 16. RMT and InstaHide protect models from model-inversion attacks. Original: the MI attack applied to the original non-protected model to recover images. RMT: the MI attack applied to the model trained on RMT-disguised training data.

work effectively against such model-targeted attacks. We take the **model-inversion (MI)** attack, for example, which tries to recover training data from the exposed model.

This experiment checks whether models trained on disguised images by RMT and InstaHide are resilient to the MI attack. We adopt the recently develop GAN-based attack [52] that has shown better performance than the original method in recovering training data. Specifically, we used a 4×4 block without noise addition for RMT to generate disguised data and models. InstaHide (Private) uses only the private training data for mix-up. For InstaHide(Public), we use EMNIST (Capital letters A–J) as the public dataset for mix-up in disguising MNIST and FASHION and CIFAR100(class 0-9) as the public dataset for mix-up in disguising CIFAR-10 and LFW.

We then apply the MI attack to generate 2,000 images for each dataset evenly distributed in each class. Finally, we use the DNN examiners to recognize the recovered images. Figure 16 shows that the MI attack with good auxiliary data can be very effective, while models trained on both RMT and InstaHide disguised data are very resilient to the MI attack. Indeed, the MI attack recovers the RMT-disguised training data, which are very different from the original images and thus still unrecognizable.

5.5 Discussion

Based on the experimental results, we have the following observations:

- With Level-1 adversarial knowledge, the RMT mechanism preserves good data utility for most datasets. In contrast, the AES scheme only works for some datasets. Table 3 summarizes the best result under the Level-1 adversarial knowledge assumption. For references, InstaHide works comparably or slightly better than RMT in terms of utility preservation [21].
- With Level-2 adversaries, the RMT mechanism should not be used, as the regression attack can likely break RMT with a small number of known pairs. The AES scheme with a small encryption unit and small (e.g., 2%) noise addition is resilient to the codebook attack and still preserves model quality for MNIST and FASHION. Table 4 summarizes the best results for AES. Note that the AES scheme does not preserve model quality for CIFAR10 and LFW and

Table 3. Best Result under Level-1 Assumption

Datasets	No Disguise	RMT Disguise	AES Disguise
MNIST	96.7 ± 0.2%	96.6 ± 0.4%	91.6 ± 1.1%
FASHION	88.7 ± 0.3%	85.1 ± 0.6%	68.9 ± 1.4%
CIFAR-10	93.4 ± 0.2%	89.3% ± 0.1%	11.3 ± 1.7%
LFW	94.3 ± 2.0%	92.6 ± 2.3%	17.2 ± 0.4%

Table 4. AES Best Result under Level-2 Attacks: encryption unit 2×1 (with scaling), noise level 2%

Datasets	No Disguise	Model Accuracy	Attack Success Rate
MNIST	96.7 ± 0.2%	90.14 ± 1.1%	30.76 ± 0.87%
FASHION	88.7 ± 0.3%	73.08 ± 0.86%	23.51 ± 0.27%
CIFAR-10	93.4 ± 0.2%	–	–
LFW	94.3 ± 2.0%	–	–

The model accuracy for CIFAR-10 and LFW are too low and thus skipped.

thus no need to evaluate these datasets. So far, we have not discovered satisfactory utility-preserving disguising methods against Level-2 adversaries.

- Finally, if only Level-1 adversaries are expected, then RMT can also be used to effectively protect from model-targeted attacks. InstaHide works, too, but it is vulnerable to Level-1 attacks.

6 CONCLUSION

Outsourcing large image datasets to use cloud GPUs for training deep learning models has been an economical and popular option. However, it also raises concerns about data and model confidentiality. The existing cryptographic solutions are too expensive to be practical. We propose the DisguisedNets image disguising mechanisms that efficiently thwart the attacks and preserve model quality. The combination of random image-block permutation and block-wise AES encryption or multidimensional transformation (RMT) does not require any changes to the existing DNN modeling architectures. We have discussed the training image reconstruction and re-identification attacks under a refined threat model with two levels of adversarial knowledge. We also comparatively analyze a similar method, the recently proposed InstaHide, under the threat model.

Experimental results show that the RMT method can preserve the model quality and provide sufficient attack resilience under Level-1 adversarial knowledge—adversaries knowing only the disguised images and the domain information. The AES method improves the attack resilience against Level-2 adversaries who manage to obtain pairs of original images and disguised ones. However, the AES method may seriously damage some datasets' utility. We also show that all the disguising methods, including InstaHide, can protect the trained models from model-targeted attacks.

A few future directions are promising to explore. (1) Based on the current results, we have seen methods such as RMT and InstaHide can well preserve data utility. However, their confidentiality guarantees are still limited. An important direction is to design methods with stronger confidentiality guarantees while the utility of disguised image data is still well preserved. (2) The image disguising methods have only been applied to classification tasks. We can explore other more challenging tasks, such as representation learning and generative modeling. (3) The current disguising methods are specifically designed for images. We think it is also possible to extend them to non-image data.

ACKNOWLEDGMENT

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funders.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. ACM, New York, NY, 308–318. DOI: https://doi.org/10.1145/2976749.2978318

[2] Samet Akcay, Mikolaj E. Kundegorski, Chris G. Willcocks, and Toby P. Breckon. 2018. Using deep convolutional neural network architectures for object classification and detection within x-ray baggage security imagery. *IEEE Trans. Inf. Forens. Secur.* 13, 9 (2018), 2203–2215. DOI: https://doi.org/10.1109/TIFS.2018.2812196

[3] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Proceedings of the 31st Annual Conference on Advances in Cryptology (CRYPTO'11)*. Springer-Verlag, Berlin, 505–524.

[4] Nicholas Carlini, Samuel Deng, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Shuang Song, Abhradeep Thakurta, and Florian Tramèr. 2021. Is private learning possible with instance encoding? In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'21)*.

[5] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. 2018. Adversarial attacks and defences: A survey. *CoRR* abs/1810.00069 (2018).

[6] Andrian Chen. 2010. GCreep: Google engineer stalked teens, spied on chats. *Gawker*, Retrieved from http://gawker.com/5637234/

[7] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.

[8] A. J. Duncan, S. Creese, and M. Goldsmith. 2012. Insider attacks in cloud computing. In *Proceedings of the IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*.

[9] Cynthia Dwork. 2006. Differential privacy. In *Proceedings of the International Colloquium on Automata, Languages andProgramming*. Springer, 1–12.

[10] Liyue Fan. 2018. Image pixelization with differential privacy. In *Proceedings of the 32nd Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy*. 148–162. DOI: https://doi.org/10.1007/978-3-319-95729-6_10

[11] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. 2021. Security vulnerabilities of SGX and countermeasures: A survey. *ACM Comput. Surv.* 54, 6, Article 126 (2021), 36 pages.

[12] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the ACM Conference on Computer and Communications Security*.

[13] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. 2014. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security'14)*. USENIX Association, 17–32.

[14] Jean Gallier. 2000. *Geometric Methods and Applications for Computer Science and Engineering*. Springer-Verlag, New York.

[15] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. 201–210.

[16] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations*. Retrieved from http://arxiv.org/abs/1412.6572

[17] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. 2018. Countering adversarial images using input transformations. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=SyJ7ClWCb

[18] Richard M. Heiberger. 1978. Generation of random orthogonal matrix. *J. R. Stat. Soc.* 27, 2 (1978).

[19] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: Information leakage from collaborative deep learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*. ACM, New York, NY, 603–618. DOI: https://doi.org/10.1145/3133956.3134012

[20] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S. Yu, and Xuyun Zhang. 2022. Membership inference attacks on machine learning: A survey. *ACM Comput. Surv.* 54, 11s (2022), 1–37.

[21] Yangsibo Huang, Zhao Song, Kai Li, and Sanjeev Arora. 2020. InstaHide: Instance-hiding schemes for private distributed learning. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Hal Daumé III and Aarti Singh (Eds.), Vol. 119. PMLR, 4507–4518.

[22] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and fast secure two-party deep neural network inference. In *Proceedings of the 31st USENIX Security Symposium (USENIX Security'22)*. USENIX Association, 809–826. Retrieved from https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong

[23] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High accuracy and high fidelity extraction of neural networks. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security'20)*. 1345–1362.

[24] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2021. Advances and open problems in federated learning. *Found. Trends Mach. Learn.* 14, 1–2 (2021), 1–210. DOI: https://doi.org/10.1561/2200000083

[25] Jonathan Katz and Yehuda Lindell. 2007. *Introduction to Modern Cryptography*. Chapman and Hall/CRC.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (May 2017), 84–90. DOI: https://doi.org/10.1145/3065386

[27] Gary B. Huang and Erik Learned-Miller. 2014. *Labeled Faces in the Wild: Updates and New Reporting Procedures*. Technical Report UM-CS-2014-003. University of Massachusetts, Amherst.

[28] Meng Li, Liangzhen Lai, Naveen Suda, Vikas Chandra, and David Z. Pan. 2017. PrivyNet: A Flexible framework for privacy-preserving deep neural network training with a fine-grained privacy control. *CoRR* abs/1709.06161 (2017).

[29] Steve Mansfield-Devine. 2015. The Ashley Madison affair. *Netw. Secur.* 2015, 9 (2015), 8–16.

[30] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'17)*. 19–38.

[31] Lucien K. L. Ng, Sherman S. M. Chow, Anna P. Y. Woo, Donald P. H. Wong, and Yongjun Zhao. 2021. Goten: GPU-outsourcing trusted execution of neural network training. *Proc. AAAI Conf. Artif. Intell.* 35, 17 (May 2021), 14876–14883. DOI: https://doi.org/10.1609/aaai.v35i17.17746

[32] Andreas S. Panayides, Amir Amini, Nenad D. Filipovic, Ashish Sharma, Sotirios A. Tsaftaris, Alistair Young, David Foran, Nhan Do, Spyretta Golemati, Tahsin Kurc, Kun Huang, Konstantina S. Nikita, Ben P. Veasey, Michalis Zervakis, Joel H. Saltz, and Constantinos S. Pattichis. 2020. AI in medical imaging informatics: Current challenges and future directions. *IEEE J. Biomed. Health Inform.* 24, 7 (2020), 1837–1857. DOI: https://doi.org/10.1109/JBHI.2020.2991043

[33] E. Raff, J. Sylvester, S. Forsyth, and M. McLean. 2019. Barrage of random transforms for adversarially robust defense. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'19)*. 6521–6530.

[34] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-party secure inference. In *Proceedings of the 27th Annual Conference on Computer and Communications Security (ACM CCS'20)*. ACM.

[35] Sharma Sagar and Chen Keke. 2021. Confidential machine learning on untrusted platforms: A survey. *Cybersecurity* 4, 1 (2021), 30. DOI: https://doi.org/10.1186/s42400-021-00092-8

[36] Ali Shafahi, W. Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. 2019. Are adversarial examples inevitable? In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=r1lWUoA9FQ

[37] Sagar Sharma, A. K. M. Mubashwir Alam, and Keke Chen. 2021. Image disguising for protecting data and model confidentiality in outsourced deep learning. In *Proceedings of the IEEE Conference on Cloud Computing*.

[38] Sagar Sharma and Keke Chen. 2018. Image disguising for privacy-preserving outsourced deep learning. In *Proceedings of the Poster Session of ACM CCS*.

[39] Sagar Sharma and Keke Chen. 2019. Confidential boosting with random linear classifiers for outsourced user-generated data. In *Proceedings of the 24th European Symposium on Research in Computer Security (ESORICS'19)*. 41–65.

[40] Sagar Sharma, Keke Chen, and Amit Sheth. 2018. Toward practical privacy-preserving analytics for IoT and cloud-based healthcare systems. *IEEE Internet Comput.* 22, 2 (Mar./Apr. 2018), 42–51. DOI: https://doi.org/10.1109/MIC.2018.112102519

[41] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*.

[42] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *Proceedings of the IEEE Symposium on Security and Privacy*. 3–18.

[43] Connor Shorten and Taghi M. Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *J. Big Data* 6, 1 (2019), 60. DOI: https://doi.org/10.1186/s40537-019-0197-0
[44] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. 2021. CryptGPU: Fast privacy-preserving machine learning on the GPU. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'21)*. IEEE, 1021–1038.
[45] Florian Tramer and Dan Boneh. 2019. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=rJVorjCcKQ
[46] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction APIs. In *Proceedings of the 25th USENIX Conference on Security Symposium (SEC'16)*. USENIX Association, 601–618.
[47] Lucy Unger. 2015. Breaches to customer account data. *Comput. Internet Law.* 32, 2 (2015), 14–20.
[48] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *J. Mach. Learn. Res.* 9 (2008), 2579–2605. Retrieved from http://www.jmlr.org/papers/v9/vandermaaten08a.html
[49] Santosh S. Vempala. 2005. *The Random Projection Method*. American Mathematical Society.
[50] Adam Yala, Homa Esfahanizadeh, Rafael G. L. D'Oliveira, Ken R. Duffy, Manya Ghobadi, Tommi S. Jaakkola, Vinod Vaikuntanathan, Regina Barzilay, and Muriel Médard. 2021. NeuraCrypt: Hiding private health data via random neural networks for public training. *CoRR* abs/2106.02484 (2021).
[51] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. 2018. mixup: Beyond empirical risk minimization. In *Proceedings of the International Conference on Learning Representations*.
[52] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. 2020. The secret revealer: Generative model-inversion attacks against deep neural networks. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'20)*.

Received 2 April 2023; revised 19 June 2023; accepted 28 June 2023