Toward Universal Detection of Adversarial Examples via Pseudorandom Classifiers

Boyu Zhu, *Student Member, IEEE*, Changyu Dong, *Member, IEEE*, Yuan Zhang, *Member, IEEE*, Yunlong Mao, *Member, IEEE*, and Sheng Zhong, *Fellow, IEEE*

***Abstract*— Adversarial examples that can fool neural network classifiers have attracted much attention. Existing approaches to detect adversarial examples leverage a supervised scheme in generating attacks (either targeted or non-targeted) for training the detectors, which means the detectors are geared to the attacks chosen at the training time and could be circumvented if the adversary does not act as expected. In this paper, we borrow ideas from cryptography and present a novel approach called *pseudorandom classifier*. In a nutshell, a pseudorandom classifier is a classifier equipped with a mapping to encode the category labels into random multi-bit labels, and a keyed pseudorandom injective function to transform the input to the classifier. The multi-bit labels enable attack-independent and probabilistic detection if the input sample is adversarial. The pseudorandom injection makes the existing white-box adversarial example generation methods, largely based on back-propagation, no longer applicable. We empirically evaluate our method on MNIST, CIFAR10, Imagenette, CIFAR100, and GTSRB. The results suggest that its performance against adversarial examples is comparable to the state-of-the-art.**

***Index Terms*— Adversarial examples, pseudorandom classifier, attack-independent detection.**

I. Introduction

DEEP neural networks (DNNs) achieve state-of-the-art performance on image classification, speech recognition, and game-playing, among many other applications. However, concerns over the security of DNNs greatly hinder DNNs' wide adoption. Among all security problems, adversarial example (AE) is perhaps the most significant one. AEs are carefully crafted inputs with small and imperceptible perturbations, which cause a DNN to make a false prediction. Since the emergence of AEs [1] for computer vision tasks, there have been extensive studies on the AE attacks and defenses [2], [3], [4], [5], [6]. AEs have been shown to be ubiquitous and are likely unavoidable as long as we want classifiers to achieve expected usability [7], [8], [9], [10], [11]. Therefore, developing an effective method to defend against AEs is an important topic. Adversarial defenses to date can be broadly categorized into two branches: one aims to improve the model robustness by ensuring the output is correct given an AE as the input [12], [13], [14], [15]; the other one aims to detect AEs [3], [16], [17], [18], [19], [20], [21], [22].

In this paper, our focus is on detection rather than robustness. Existing detection mechanisms try to distinguish AEs from natural samples based on *what the adversaries can do*. The common methodology is to train a detector using a supervised scheme with natural samples and AEs generated using assumed target attacks (e.g. PGD). The detector learns the features that tell AEs and natural samples apart. However, the effectiveness of detection depends on multiple assumptions, such as the adversary is oblivious to the detection mechanism, and/or the adversary's strategy is predictable. They can be easily bypassed if the adversary adapts its attack strategy based on what the detector does, or use other attacks not anticipated when training the detector.

The idea presented in this paper is fundamentally different. Rather than relying on what the adversaries can do, our detection mechanism relies on what the adversaries *CANNOT* do. The inspiration comes from cryptography. Designing protocols that work in an adversarial environment has been a problem investigated in cryptography for many years. Cryptographic solutions for detecting adversarial behaviors often involve designing a probabilistic test and utilizing some sort of information asymmetry or computational intractability. Roughly speaking, if the adversary wants to cheat, it must pass a test that requires it to either know some secret (e.g., a cryptographic key) or solve a hard problem (e.g., to find the factors of a large integer); otherwise, its cheating behavior can be detected with some probability. The detection probability can be amplified to almost 1 by repeating the test with independent randomness. Hence, by keeping the secret secure and ensuring the hard problem is intractable, we can detect the cheating adversary, no matter what else it can do.

Manuscript received 10 March 2023; revised 28 October 2023; accepted 29 November 2023. Date of publication 7 December 2023; date of current version 26 December 2023. This work was supported in part by the Natural Science Foundation on Frontier Leading Technology Basic Research Project of Jiangsu under Grant BK20222001; in part by the Leading-Edge Technology Program of Jiangsu National Science Foundation under Grant BK20202001; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20230080; in part by the National Key Research and Development Program of China under Grant 2020YFB1005900; and in part by the National Natural Science Foundation of China under Grant 61872176, Grant 62272215, Grant 61872179, Grant 62272222, Grant 62072132, and Grant 62261160651. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ali Dehghantanha. *(Corresponding authors: Changyu Dong; Sheng Zhong.)*

Boyu Zhu, Yuan Zhang, Yunlong Mao, and Sheng Zhong are with the State Key Laboratory for Novel Software Technology and the Computer Science and Technology Department, Nanjing University, Nanjing 210023, China (e-mail: zhuzby@outlook.com; zhangyuan05@gmail.com; maoyl@nju.edu.cn; zhongsheng@nju.edu.cn).

Changyu Dong is with the Institute of Artificial Intelligence, Guangzhou University, Guangzhou 510000, China (e-mail: changyu.dong@gmail.com).

Digital Object Identifier 10.1109/TIFS.2023.3340889

A. Contributions

Our main contribution in this paper is a new cryptography-inspired AE detection mechanism called pseudorandom classifier. Our first observation is that since the data space can contain not only valid natural samples but also invalid samples, forcing a classifier to only output a legitimate label is problematic. Hence we enlarge the label space, using a random multi-bit encoding, to include new categories for invalid samples. The random encoding also induces an attack-independent probability of detecting invalid samples. We also use a pseudorandom injective function (with a secret key) to transform the input, so that the detection cannot be easily bypassed by the adversary. Then, we can amplify the probability by an ensemble of independent pseudorandom classifiers. Pseudorandom classifiers are not just a theoretical concept. We also show how to instantiate it with readily available cryptographic building blocks. The performance of pseudorandom classifiers is evaluated empirically using MNIST, CIFAR10, Imagenette, CIFAR100 and GTSRB on L_2 and L_∞ based adversarial perturbations. The results show our method achieves or surpasses state-of-the-art results.

II. Related Works

Existing defenses against AEs can be categorized into two types: counter-AE defenses and detection-based defenses. Counter-AE defenses [23], [24] aim to correctly classify AEs. They work by making models more robust so that AEs are no longer misclassified. Detection-based defenses [25] take a different approach. Rather than trying to classify AEs correctly, they aim to detect AEs and reject classifying them. In this section, we introduce common defensive techniques that are widely used in both types of defenses. We also discuss some specialized techniques that are unique to detection-based defenses. This overview contextualizes our method within the broader field and highlights the differences between our method and existing approaches.

A. Defensive Techniques

Mainstream defensive techniques include adversarial training, gradient masking, and input transformations [26].

1) Adversarial Training: One approach for defending against adversarial examples is to build a more robust classifier through adversarial training [27], [28]. The idea is to expand the training dataset to include adversarial examples along with their true labels. This teaches the model to correctly classify samples with adversarial perturbations. The adversarial training examples can be generated by attack methods [29] or separate generative models like GANs [30]. However, models trained on specific adversarial techniques may still be vulnerable to new, more advanced attacks [31]. For robustness, adversarial training would need to cover almost all attack techniques, which is computationally prohibitive [32].

2) Gradient Masking: Many white-box attacks rely on calculating gradients of the model to craft adversarial examples. Gradient masking defenses aim to thwart these attacks by preventing attackers from obtaining useful gradient information. For instance, Papernot et al. [33] introduced defensive distillation, which smooths the prediction outputs from an existing DNN model. It then trains a new model on these smoothed outputs and replaces the last layer with a revised softmax function to conceal the model's gradients [34]. However, defensive distillation has been shown to be fragile against advanced attacks [35]. Athalye et al. [36] also demonstrated that gradient masking can be bypassed through proper gradient approximation. In Section VII, we provide a more detailed discussion and analysis of our approach from the perspective of gradient masking.

3) Input Transformation: Several works [37], [38], [39], [40] use input transformations to mitigate adversarial perturbations before feeding inputs into the DNN model. FS [41] is a typical input transformation defense. By coalescing samples that correspond to many different feature vectors in the original space into a single sample, it reduces the search space available to an adversary. MagNet [42] trains a reformer (autoencoder) on clean data to purify AEs near the data manifold, along with detectors to identify AEs far away using reconstruction error and divergence metrics. During inference, MagNet screens out detected inputs, reforms the remaining inputs, and classifies the reformed inputs. Since this defense can be broken if the attacker knows the parameters of the reformer, the authors introduce randomness to strengthen MagNet by training multiple autoencoders and randomly picking one for each input. However, Carlini et al. were still able to break this using adaptive transferable adversarial examples [43]. After that, more random input transformation defenses [44], [45], [46] were proposed to improve robustness, but Tramer et al. [2] showed these are still ineffective towards adaptive attacks. Later, defenses like BaRT [47] resort to larger randomization space, but Sitawarin et al. [48] broke these too. One explanation for why random input transformations do not work well against AEs is that they do not inherently make the model more robust. Instead, they just introduce errors and variance during inference. The observed "robustness" comes from these errors, not the model learning to be more resistant to adversarial examples [49]. Other studies have explored injecting noise during training to improve model robustness against adversarial attacks [50]. In particular, random smoothing [51], [52], [53], [54] showed adding Gaussian noise to samples can provide provable defenses in limited cases [55]. Building on this, Salman et al. [56] applied a custom-trained denoiser to a pre-trained classifier, converting it into a new smoothed classifier provably robust to adversarial attacks under certain conditions. Most recently, Carlini et al. [57] utilized a denoising diffusion probabilistic model as the denoiser, achieving state-of-the-art certified robustness against L_2 norm perturbations.

Our method also utilizes the input transformation as one building block, but it differs from existing random input transformation defenses in a few key ways: i) For the same input, our method always outputs the same prediction. This is because we apply pseudorandom injection whose output is deterministic once the key is fixed. Also, we connect predictions from all models in the ensemble as the final prediction instead of randomly selecting one prediction. ii) Through pseudorandom injection used in our method, the

TABLE I

COMPARISON OF DIFFERENT DETECTION-BASED DEFENSE METHODS ON DIFFERENT DIMENSIONS: (I) CLASSIFICATION TYPE SIGNING WHETHER THE METHOD USES GENERATED AES IN TRAINING: SUPERVISED LEARNING (S) MEANS USING AES AND UNSUPERVISED LEARNING (U) MEANS WITHOUT AES. (II) THREAT MODEL SHOWING THE METHOD'S ROBUSTNESS AGAINST ADAPTIVE ATTACKS (A), WHERE THE ADVERSARY KNOWS THE METHOD, VERSUS NON-ADAPTIVE (N), WHERE THE ADVERSARY IS OBLIVIOUS. (III) DNN OPTIMIZATION LOSS FUNCTION USED TO TRAIN THE BASE MODEL: CROSS-ENTROPY (C), REVERSE CROSS-ENTROPY (R), AND BINARY CROSS-ENTROPY (B). (IV-IX) REFER TO THE DEFENSIVE TECHNIQUES DISCUSSED IN SECTION II, WHERE (IX) LABEL ENCODING INCLUDES ONE-HOT ENCODING (O) AND MULTI-BIT ENCODING (M)

Detection Method	Classification Type	Threat Model	DNN Optimization	Adversarial Training	Input Transformation	Auxilary Models	Statistical Approach	Ensemble	Label Encoding
MagNet [42]	U	N	C		•	•		•	O
FS [41]	U	N	C		•				O
NIC [34]	U	N	C			•		•	O
Pang et al. [59]	U	A	R				•		O
LID [60]	S	N	C				•		O
KD+DU [61]	S	N	C				•		O
LNG [62]	S	A	C			•			O
GAT [6]	S	A	C	•		•		•	O
Ours	U	A	B		•			•	M

transformed input retains all of the original input information. Each encoded image can be perfectly decoded using the true key. This differs from other techniques like adding noise or squeezing, which permanently distort or discard information from the original input.

4) Label Encoding: In our method, we use multi-bit label encoding instead of the conventional one-hot encoding. This looks similar to, but is actually quite different from the error-correcting output code (ECOC) proposed by Verma and Swami [58]. ECOC is designed to improve a classifier's robustness, not to detect AEs. It maps K classes into M-dimensional binary code words in an error-correcting code. If the output from the classifier for an example differs from the legitimate codes, it will be corrected so that the example will always be assigned to a class. The fundamental difference is that they believe every example presented to the classifier belongs to a legitimate class, while we do not (see Section III).

B. Detection-Based Defenses

There are two typical techniques used in detection-based defenses: training auxiliary models and utilizing statistical properties.

1) Auxilary Models: Several works have developed auxiliary models in addition to the base DNN model to detect differences between clean samples and AEs. GAT [6] trains a separate binary classifier for each class to detect AEs. During training, each classifier is trained on positives from in-class clean samples and negatives from AEs crafted to fool the classifier. However, training on AEs can cause overfitting to the particular attacks used for training, failing to generalize to new attacks [63]. This is because the AEs generated during training may not cover the full space of possible AEs an attacker could create. In contrast, some auxiliary models only observe clean sample behaviors during training. NIC [34] works by modeling the normal distribution of activation values and provenance channels for each layer using clean training data. It then uses one-class classification to detect when an input induces channel values that fall outside the learned clean distribution, marking it as an adversarial example. It achieves the state-of-the-art performance [64], [65].

2) Statistical Approach: Prior work has developed various statistical properties to distinguish between clean samples from AEs. These properties assess whether an input is sampled from the training data distribution/manifold. Local Intrinsic Dimensionality (LID) [60] calculated the distance distribution of the input sample to its neighbors to assess the space-filling capability of the region surrounding that input sample. Empirically, AEs tend to have high LID values. Feinman et al. [66] proposed Kernel Density (KD) and Bayesian Uncertainty (BU). They observed that AEs subspaces usually have lower density than clean samples. Both LID and KD+BU train classifiers using the proposed statistics to identify clean, noisy, and AEs. Pang et al. [59] used kernel density as a threshold metric to detect AEs. They also modify DNN optimization to learn representations that better separate normal and adversarial inputs using reverse cross-entropy. A key challenge of the statistical approach is how to define a high-quality statistical metric that can clearly tell the difference between clean samples and adversarial samples.

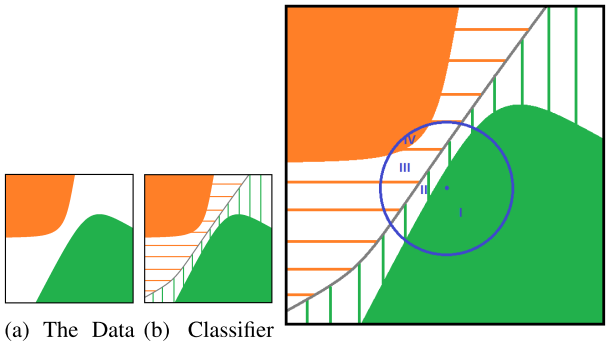
In addition, the success of many proposed detection-based defenses [16], [34], [41], [60], [67], [68], [69] depends on that the adversary is oblivious to the detection mechanisms. Actually, these detection mechanisms are less effective than their claims under adaptive attacks [2], [36], [43].

Table I summarizes key differences in adversarial detection methods based on the above defensive techniques. We compare the performance of GAT, NIC, FS with ours in Section VI.

III. AE DETECTION: THE PROBLEM STATEMENT

Without loss of generality, let us explain with the case of binary classification. Consider an n-dimensional data space $U = [0, 1]^n$ and two categories $\{0, 1\}$, such that each category is associated with a subset $D_i \subset U$ and $D_0 \cap D_1 = \emptyset$. We also assume, as in practice, that D_0 and D_1 do not take up the whole data space, i.e. there are invalid samples. For example, the categories may be dog and cat images, but the n-dimensional data space also contains many images that are neither a dog or a cat (e.g. a bird, white noise etc.). Hence we also define the valid space as $D_{val} = D_0 \cup D_1$ and the invalid space as it complement $\overline{D_{val}} = U \setminus D_{val}$ (as depicted in Fig. 1(a)).

When we train a binary classifier $C : [0, 1]^n \rightarrow \{0, 1\}$, the training dataset D_{tr} is sampled from D_{val}. If well-trained, the classifier learns how to distinguish samples from D_0 and



(a) The Data Space (b) Classifier C (c) The ϵ-noisy space $\mathcal{B}_\epsilon(x)$

Fig. 1. The case of binary classification. The solid green, orange, and white areas denote D_0, D_1, and invalid sample space respectively. The black line is the classifier's decision boundary. Shadowed areas denote invalid samples that are given a legitimate label by the classifier.

D_1. This is fine if we only use the classifier in a benign environment where the samples to be predicted are all from D_{val}. However, in an adversarial environment, the samples to be predicted may be maliciously crafted and come from the invalid space. The learned decision boundary divides U into two parts, and the classifier has no choice but to label invalid samples with a label 0 or 1 (as depicted in Fig. 1(b)).

Given a valid sample x and a classifier C, an AE x' is defined as $x' \in \mathcal{B}_\epsilon(x)$ and $C(x') \neq C(x)$, where $\mathcal{B}_\epsilon(x)$ is the ϵ-noisy space $\{x'|x' = x + \delta, \|\delta\|_p \leq \epsilon\}$. As we can see in Figure 1(c), for any $x \in D_i$, $\mathcal{B}_\epsilon(x)$ can be divided into four different parts:

- **Part I**: $\mathcal{B}_\epsilon(x) \cap D_i$. For any x' in part I, the noise term δ does not change the semantics of the sample, and C can correctly classify x', i.e. the noise is ineffective.
- **Part II**: $\overline{D_{val}} \cap \{x'|C(x') = i, x' \in \mathcal{B}_\epsilon(x)\}$. Samples in this area are invalid but classified into the original category as x, i.e. the classifier is robust against the samples in Part II.
- **Part III**: $\overline{D_{val}} \cap \{x'|C(x') \neq i, x' \in \mathcal{B}_\epsilon(x)\}$. Samples in this part are AEs, i.e. the noise term now causes misclassification on an unnatural sample.
- **Part IV**: $\mathcal{B}_\epsilon(x) \cap D_{i'}, i' \neq i$. Samples in this part are "overly-perturbed" AEs: x is modified into a natural sample of another category and its semantics is completely changed.

AEs are samples in Part III and IV. However, Part IV AEs are not detectable or preventable, because they are indistinguishable from natural samples (in the other category). Defenses against AEs hence focus on detecting or tolerating samples in Part III. For example, robust classifiers (e.g. by adversarial training) try to move the decision boundary and minimize Part III (and maximize Part II).

In our paper, our focus is to detect samples in Part III, i.e. when given an AE in Part III, we want our classifier to be able to label it as "invalid" with a large probability.

IV. The Proposed Method

A. Intuition

First of all, as we have seen in the previous section, when the data space can include invalid samples, forcing a classifier to classify each sample into a legitimate category is problematic. We also need to consider a category for the invalid samples, in addition to the legitimate categories. For example, for the binary classification problem, that means we could have three categories: $\{-1, 0, 1\}$, where -1 means "invalid". Then the classifier is capable of detecting AEs by labeling them as invalid.

$\pi(l_1) \rightarrow 10010$
$\pi(l_2) \rightarrow 11011$
$\pi(l_3) \rightarrow 01100$
$\pi(l_4) \rightarrow 00110$

(a)

	Category 1	Category 0
1st bit	$\{l_1, l_2\}$	$\{l_3, l_4\}$
2nd bit	$\{l_2, l_3\}$	$\{l_1, l_4\}$
3rd bit	$\{l_4, l_3\}$	$\{l_1, l_2\}$
4th bit	$\{l_1, l_2, l_4\}$	$\{l_3\}$
5th bit	$\{l_2\}$	$\{l_1, l_3, l_4\}$

(b)

Fig. 2. Multi-bits labels: an example.

Having an invalid category is not enough. The first challenge we face is that we may not always know what the attack is at the training time. Using randomly generated data samples or AEs generated by certain assumed attacks does not work well, as they may not reflect the true distribution of the invalid sample in the real attacks. Hence, if the adversary uses a different strategy than assumed at the training time, the detection can be ineffective. However, if we do not assume any attacks and uses only natural samples in training a classifier with the additional invalid category, then the trained classifier is also unlikely to perform well because it has no idea what is an invalid sample. To address this problem, we use a random multi-bit label for each legitimate category and cast the AE detection problem into detecting whether an input has incompatible features.

The second challenge we face is the white-box attacks, such as PGD and CW, that use gradients to direct the search of AEs. They can easily avoid detection by optimizing towards a target category and thus avoid being classified as invalid. To address this challenge, we use pseudorandom injection functions to transform the input before it enters the classifier.

The last challenge is how to ensure the detection probability is high enough. For this, we use an ensemble of multiple independent classifiers to amplify the detection probability.

B. Technical Details

Our design roughly resembles the common blueprint in cryptography: the multi-bit labels give rise to an attack-independent probabilistic test, the pseudorandom injection functions ensure the test cannot be bypassed easily, and the detection probability is amplified by an ensemble of pseudorandom classifiers.

1) Multi-Bit Labels: An AE is often generated by modifying a natural sample. It still looks like a sample in the original category, but the victim classifier misclassifies it into another category. Consequently, it has features of both the source and target categories. If a sample is classified into category i but also has features that do not belong to i, then it is likely to be an AE. This is the idea behind the multi-bit labels.

Given an m-classification problem with labels $\{l_1, \cdots, l_m\}$, we define a deterministic mapping $\pi : \{l_1, \cdots, l_m\} \rightarrow \{0, 1\}^d$ that maps each label to a distinct random d-bit string, where

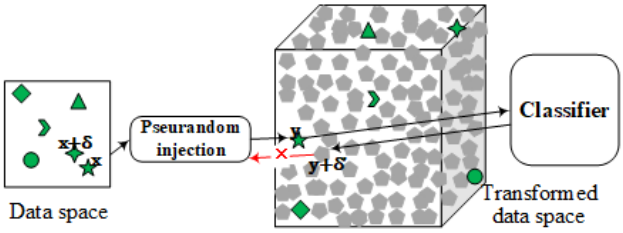


Fig. 3. White-box attack towards pseudorandom injection.

$2^d > m$. Then $\pi(l_i)$ is the multi-bit label of category l_i $(1 \le i \le m)$, and each d-bit string not having a corresponding category is a label for the invalid category. An example is shown in Figure 2(a), which maps $\{l_1, \cdots, l_4\}$ to 5-bit labels. Other bit patterns not shown, e.g. 00000, are labels for "invalid".

Each bit in the label corresponds to a binary classification problem. An example is shown in Figure 2(b). Let us start from the first bit in the table. The first bit of the new labels of the categories l_1, l_2 is 1, hence l_1, l_2 are combined into a new category 1, and the categories l_3, l_4 are combined into a new category 0. We can train a binary classifier that classifies samples into the new 0/1 categories based on some features $\mathcal{F}$. For the simplicity of explanation, let us say l_i has $\mathcal{F}$ if l_i is in category 1, and l_j do not have $\mathcal{F}$ if l_j is in category 0. Likewise, each of the other bits gives rise to a different random binary partition of the original set of labels, thus a different binary classification problem. The features in each case $\mathcal{F}_1, \cdots, \mathcal{F}_d$ are also different. So if category l_1 is mapped to a 5-bit label "10010", it essentially says samples in l_1 should have features $\mathcal{F}_1$ and $\mathcal{F}_4$, but not $\mathcal{F}_2$, $\mathcal{F}_3$ and $\mathcal{F}_5$. When we classify a sample, if the classifier outputs a label "10110", then the sample has features $\mathcal{F}_1$ and $\mathcal{F}_4$. This combination does not match any of the l_i's, so the sample is likely an AE.

In the above, we do not need to assume how exactly the adversary would modify the natural samples into AEs. Therefore, the detection is independent of the attacks. The detection however is probabilistic, which means some AEs could go through without being detected (maybe with a significant probability). We will later show how the detection probability can be amplified.

2) Pseudorandom Injections: Using only the multi-bit labels, the detection mechanism can be circumvented by using white-box attacks. To counter white-box attacks, we use pseudorandom injective functions. In cryptography, a pseudorandom function is a deterministic function of a key and an input such that its output is indistinguishable from uniformly random bit-strings. We make use of pseudorandom functions that are injective. More precisely, let λ be a security parameter (the length of the key), then a pseudorandom injection $\phi : \{0,1\}^\lambda \times [0,1] \to [0,1]^t$ takes a λ-bit key and expand a real number in $[0,1]$ into a randomly looking t-vector of real numbers in $[0,1]$. For $x \in [0,1]^n$, $\phi(k,x) \in [0,1]^{n\times t}$ means that $\phi(k,\cdot)$ is applied element-wisely to the vector x.

The intuition is depicted in Fig 3. With a pseudorandom injection, the input to a classifier is not the natural input x, but its transformed value $y = \phi(k,x)$. The adversary running a white-box attack can utilize gradients to find $y+\delta'$ in the transformed data space. To map $y+\delta'$ back to the data space, the adversary has a problem that with an overwhelming probability, $y+\delta'$ does not have a pre-image and is independent of samples in the sample space. Therefore, with an overwhelming probability, $y+\delta'$ is useless, in the sense the probability the adversary can find the true adversarial example $x+\delta$ given $y+\delta'$ is almost the same if the adversary guesses without $y+\delta'$. Formally, We have the following:

Theorem 1: Let $\phi : \{0,1\}^\lambda \times U \to T$ be a pseudorandom injection, $x \in U$ be a natural sample, and $y = \phi(k,x) \in T$ be the image of x in the transformed data space. An adversary who runs a white-box adversarial example attack $A_\theta : T \times T$ has only negligible advantage in finding an adversarial example $x+\delta \in U$. That is,

$$\Pr[x = x+\delta | Y = y+\delta'] = \Pr[x = x+\delta] \pm \epsilon,$$
$$\text{where } y+\delta' \leftarrow A_\theta(\phi(k,x)), \epsilon \text{ is negligible}$$

Proof: We start from x as a single pixel which is encoded as a byte, and is transformed by the pseudorandom injection into t bytes. There are two mutually exclusive cases: (1) Case 1: $\phi^{-1}(k, y+\delta') \neq \perp$, i.e. $y+\delta'$ found by the white-box attack algorithm has a pre-image; (2) Case 2: the negation of case 1, $\phi^{-1}(k, y+\delta') = \perp$. In case 1, $\Pr[X = x+\delta | Y = y+\delta'] = p$ for some p at most 1. In case 2, since the inverse mapping is not defined, $X = x+\delta$ and $Y = y+\delta'$ are independent, hence $\Pr[X = x+\delta | Y = y+\delta'] = \Pr[X = x+\delta]$. Overall, we have:

$$\Pr[X = x+\delta | Y = y+\delta'] = \tag{1}$$
$$\Pr[(X = x+\delta | Y = y+\delta') | \text{case1}] \cdot \Pr[\text{case1}]$$
$$+ \Pr[(X = x+\delta | Y = y+\delta') | \text{case2}] \cdot \Pr[\text{case2}]$$
$$= p \cdot \Pr[\text{case1}] + \Pr[X = x+\delta] \cdot \Pr[\text{case2}] \tag{2}$$

For each element in T, because the mapping is random, the probability that it has a pre-image (case 1) is $|U|/|T|$ (here $|\cdot|$ means the cardinality), and the probability not having a pre-image (case 2) is $1 - |U|/|T|$. In the case of 1 byte x to t bytes y, $\Pr[\text{case1}] = 2^{-8(t-1)}$ and $\Pr[\text{case2}] = 1 - 2^{-8(t-1)}$. When t is sufficiently large, $2^{-8(t-1)}$ is negligibly small and $1 - 2^{-8(t-1)}$ is almost 1. Taking that into Eq.(2), we can conclude $\Pr[X = x+\delta | Y = y+\delta'] = \Pr[X = x+\delta] \pm \epsilon$. This conclusion can be extended to multi-byte x easily. □

Remark: Some readers may say the adversary can bypass the pseudorandom injection by learning an approximate inverse function that maps $y+\delta'$ back to a point close to $x+\delta$. This however is impossible because such an approximate inverse function contradicts to the fact we are using an injection and the injection is pseudorandom. An analogy is that we can create a mapping between the round fruit of a tree of the rose family and the word "apple", but there is no way to determine what is an "appla" if there is no pre-defined mapping for this word, even though the two words look similar.

3) Pseudorandom Classifier: A normal classifier is a function $[0,1]^n \to \{l_1, \cdots, l_m\}$. To train such a classifier, a labeled training set is used, such that each training sample in the training set D_{tr} is a pair (x,y) where $x \in [0,1]^n$ is a data sample and y is x's label. We augment a classifier with a

label mapping π and a key k of the pseudorandom injection ϕ. A pseudorandom classifier is then a function $C_\theta : [0, 1]^{n\times t} \rightarrow \{0, 1\}^d$.

To train C_θ, each training sample (x, y) is transformed into $(\phi(k, x), \pi(y))$, and then the transformed pair is used as the actual input and label in training the classifier. Recall that each bit in the d-bit label $\pi(y)$ corresponds to a binary classification problem. To realize this, the classifier has d output nodes, and we train by optimizing a multi-task objective:

$$\min_\theta \mathbb{E}_{(x,y)\in D_{tr}} \left[\sum_{i=1}^{d} L(y_i'', y_i') \right],$$
$$\text{where } y'' = C_\theta(\phi(k, x)), y' = \pi(y),$$

in which L is the binary cross entropy loss function.

The inference process starts by transforming a sample x into $\phi(k, x)$, which is passed to the classifier, then the classifier outputs $y'' \in [0, 1]^d$. The distance of y'' to each valid label $1 \leq i \leq m$ is calculated $\mathbf{d}_i = \|y'' - \pi(i)\|$, and the class with the shortest distance is determined $l = arg \min_{1\leq i\leq m} \mathbf{d}_i$. A predefined threshold τ is used, if $\mathbf{d}_l \leq \tau$, x is labeled as l; otherwise, x is labelled as invalid.

4) Ensemble: We use an ensemble of pseudorandom classifiers to increase detection probability. The ensemble has K independent pseudorandom classifiers. Each classifier has a different random mapping π_i and a different random key k_i for ϕ. Given an invalid sample, if the probability of it being labeled as invalid by the i-th pseudorandom classifiers is p_i, then the probability of it being detected by the ensemble is $1 - \prod_{i=1}^{K}(1 - p_i)$.

C. Concrete Instantiation

In this subsection, we explain how to instantiate a pseudorandom classifier concretely. In the next section, the experiments were performed using a pseudorandom classifier instantiated in this way.

To instantiate the multi-bit labels, we create a lookup table π. For each class, t bits in its label are set to 1. Then there are $\binom{d}{t}$ different possible labels, and we select m labels randomly to build a lookup table. Of course, the labels can also be instantiated in other ways as long as the label space is large enough to accommodate all classes.

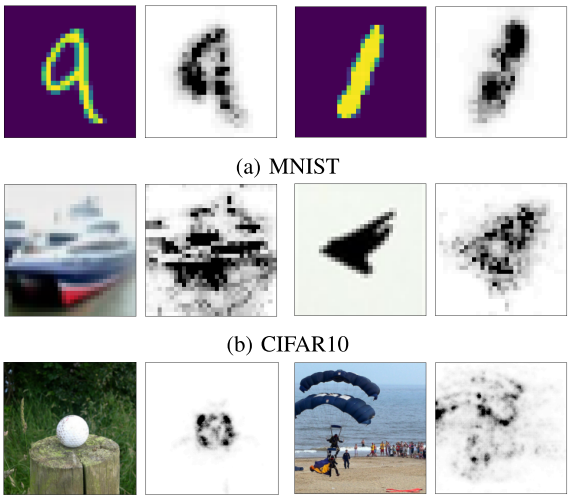
The pseudorandom injection ϕ is instantiated with SHA-256, a cryptographic hash function that has a fixed output length of 256-bit. We chose the key length to be 128-bit, conforming to the recommended security level. Given a sample x, we view it as a vector of bytes. Then the output $h = \phi(k, x)$ is constructed as the following: for each byte x_i in x, compute $h_i = \mathsf{SHA\text{-}256}(k||x_i)$ where $||$ means concatenation, then append h_i to h. If x is n bytes, the output h is $32n$ bytes.

A pseudorandom classifier consists of a multi-bit labels lookup table π, a random key for the pseudorandom injection ϕ, and a neural network (NN). The input to the NN is $\phi(k, x)$ (each byte is scaled to a real number in $[0, 1]$), and the output layer of the NN corresponds to the multi-bit label. The loss function, as mentioned earlier, is the binary cross entropy. When using a pseudorandom classifier for inference, we use L_1 distance for decision-making. Given the output confidence vector from the pseudorandom classifier, we round the confidence vector to a binary vector c' first.

Then we compute the L_1 distance between c' and each legitimate multi-bit label. Given the distances between the output vector and all labels, we choose the one with the smallest distance as our decision. If the smallest distance is lower than a given threshold τ, we accept the decision, otherwise reject (i.e., label the input as invalid and regard it as an AE).

1) Discussions: Initially when training pseudorandom classifiers, we noticed that the model accuracy is significantly lower than that of conventional classifiers. The degradation in performance likely comes from the fact that the pseudorandom injection breaks the continuity of pixel values: pixels of close values are randomly mapped to values that can be far apart. This could cause poor generalization. In order to address the problem, we tweak the training process by using data augmentation [70] and adding small Gaussian noise to the transformed data $\phi(k, x)$. These two are well-known techniques for improving generalization. Although there is still a gap, in Table V we can see that after the tweaking the performance is tolerable.

Another related question is that since the transformed data output by a pseudorandom injection is drastically different from the original, can pseudorandom classifiers learn meaningful features? To answer this question, we borrow methodologies from explainable machine learning [71], [72] and generate saliency maps to show the importance of each pixel in determining a model's classification results.[1] Some results obtained from pseudorandom classifiers (without ensemble) are displayed in Figure 4. We can see from the saliency maps that the important features come mostly from the area bearing the target objects. The results suggest



(a) MNIST

(b) CIFAR10

(c) Imagenette

Fig. 4. Saliency Maps. For each pair of images, the left is the original image and the right is its saliency map. Darker regions correspond to parts of the image that most strongly influenced the model's prediction, while lighter regions had little influence.

[1]To generate saliency maps, we use Integrated Gradients [71] with Noise Tunnel [72] provided by Captumn (https://captum.ai/).

TABLE II
DATASETS AND VICTIM MODELS

Dataset	# samples	# classes	image size	model Arch.
MNIST	60000	10	28,28,1	LeNet
CIFAR10	50000	10	32,32,3	ResNet50
Imagenette	9469	10	224,224,3	ResNet50
CIFAR100	50000	100	32,32,3	ResNet50
GTSRB	39209	43	32,32,3	VGG19-bn

that pseudorandom classifiers can learn (or at least partially) the correct semantics from transformed examples. This is understandable as a pseudorandom injection is deterministic, hence (1) the same pixel value is always mapped to the same pseudorandom value, and (2) the mapping does not alter the structural relations among pixels.

V. EXPERIMENT SETUP

A. Datasets & Models

We evaluated our method on several image classification datasets: MNIST [73], CIFAR10 [74], Imagenette [75], CIFAR100 [74], and GTSRB [76]. We trained victim models on these datasets using standard architectures: LeNet [77] for MNIST, VGG [78] for GTSRB, and ResNet [79] for CIFAR10, CIFAR100, and Imagenette. The details of datasets and victim models are listed in Table II.

B. Implementation of Psedurandom Classifiers

1) Multi-Bit Labels: For MNIST, CIFAR10, and Imagenette, we used 5-bit labels (d=5) with 2 bits set to 1 (t=2) in each label. This resulted in lookup tables with 10 total 5-bit labels. For GTSRB, we used 8-bit labels (d=8) with 4 bits set to 1 (t=4), creating lookup tables with 70 total 8-bit labels. For CIFAR100, we used 50-bit labels (d=50) with 2 bits set to 1 (t=2), resulting in lookup tables with 1225 total 50-bit labels. The label encodings were designed based on the number of classes and complexity of each dataset. To ensure diversity across models in the ensemble, we generated distinct lookup tables by scrambling the label mappings for each model.

2) Pseudorandom Injections: For MNIST, CIFAR10, Imagenette, and GTSRB, we used SHA-256 with a 128-bit key to encode each color value. For example, each 1-byte color value in a (28,28,1) shaped sample was transformed into 256 bits (32 bytes), resulting in encoded samples of size (28,28,32). For CIFAR100, we used SHA-512 with a 256-bit key due to the larger number of classes and greater complexity of this dataset compared to the others. We varied the hash function based on the dataset complexity to balance security and utility. Each pseudorandom classifier was associated with a different key to ensure distinct encoded representations across models.

3) Training Details: For training, we used batches of 50 image-label sample pairs. The labels were encoded into multi-bit ones using a lookup table unique to each model. The images were augmented using TrivialAugment Wide [70], a data augmentation technique, and then transformed via our proposed pseudorandom injection method. Gaussian noise ($N(0, 0.1)$) was also added to the encoded images before feeding them as inputs to the model. We trained each model using the Adam optimization algorithm with an initial learning rate of 5e-4, decaying it by 0.5 every 100 epochs. The models were trained for 150 epochs (MNIST), 400 epochs (CIFAR10, GTSRB), 600 epochs (Imagenette), and 800 epochs (CIFAR100). This allowed the models to learn mappings between the encoded images and labels for each dataset.

4) Inference Details: To obtain a classification from the model output, we first normalized each output value to between 0 and 1 using the sigmoid function. Values greater than 0.9 were rounded to 1, values less than 0.1 were rounded to 0, and other values were rounded to 0.5. We then calculated the L_1 distance between the rounded output and each of the legitimate multi-bit label encodings. The class corresponding to the label encoding with the smallest L_1 distance to the model output was selected as the classification result. If this smallest distance was less than the threshold τ, we returned the predicted class. Otherwise, we rejected classification by returning -1. The threshold τ was set to 3 for CIFAR100, which has a larger label space, and 0 for the other datasets.

C. State-of-the-Art Detections

We compared our method against three state-of-the-art detection-based defenses: GAT [6] designed to withstand adaptive attacks; NIC [34] achieving state-of-the-art detection performance; FS [41] applying input transformations. While most baselines have been bypassed by more advanced attacks [2], [80], they still perform well against limited attackers unaware of the defense. We reproduced these baseline methods using their publicly available open-source code for a fair comparison. We elaborate on those detection methods as follows:

1) Generative Adversarial Training (GAT): GAT trains an independent binary classifier for each class. Each binary classifier is trained on natural examples from that class as positives. The negatives are adversarial examples generated from natural examples of other classes. GAT has two detection frameworks: integrated classifier (GAT-I) and generative classifier (GAT-G). GAT-I framework trains the binary classifiers with a base classifier that classifies natural examples correctly. During inference, the input is first classified by the base classifier, and then fed to the corresponding binary classifier. If the binary classifier's confidence score is below a threshold, the input is rejected as an adversarial example. The GAT-G framework does not have a base classifier. During inference, the input is fed to all binary classifiers. If the highest confidence score is below the threshold, the input is rejected. Otherwise, the predicted class is the one with maximum confidence. Since GAT loads all classifiers during inference based on its implementation,[2] it cannot scale to datasets with many classes. Thus, we only evaluate GAT on MNIST, CIFAR10 and ImageNette.

2) Neural Network Invariant Checking (NIC): NIC trains a set of models to capture the invariant distributions of each layer in a base model when it processes normal images. Specifically, it trains two types of models per layer: the value invariant model that describes the normal range of activation

[2]GAT open-source code: https://github.com/xuwangyin/GAT-MNIST.

values for that layer's neurons, and the provenance invariant model that describes which neurons in the next layer are typically activated by that layer. Invariant models are trained as a One-Class Classification (OCC) problem where all training samples are positive. At test time, NIC passes input through the base model layer by layer. At each layer, the invariant models check if the activation values violate the invariant distributions. Finally, the outputs from all invariant models are aggregated through a joint OCC classifier. If the final decision declares the input is out-of-distribution, NIC labels it as adversarial.

3) Feature Squeezing (FS): FS coalesces inputs corresponding to many different feature vectors in the original space into a single sample during inference. Specifically, FS squeezes each input sample by color bit-depth reduction and spatial smoothing. The original and squeezed images are then passed through the target model to obtain predictions. FS calculates the L_1 distance between these predictions. If the distance exceeds a predefined threshold, the input is declared adversarial.

D. Threat Model

Threat models can be divided into two scenarios based on the adversary's knowledge of the detection mechanism:

The adaptive scenario assumes the adversary is aware of the detection mechanism in place and knows the detection scheme. This means the adversary can jointly attack both the base model and detection mechanism. We evaluate GAT and our method under this scenario. Here, black-box adversaries have no access to the detector or model parameters. White-box adversaries know the parameters of both the base model and detectors, including the lookup tables used in our multi-bit label method. Depending on whether the adversary steals the keys for the pseudorandom injection, we further consider two cases. In one the adversary knows the keys, and in the other, the adversary does not.

The non-adaptive scenario assumes the adversary is unaware of the detection mechanism and generates adversarial examples solely based on fooling the base model. Under this scenario, both white-box and black-box adversaries create adversarial examples targeting only the base model. Then the detection performance is evaluated on those adversarial examples. This scenario is less realistic, especially for black-box adversaries that would craft adversarial examples using the base model's responses while ignoring the detector. Therefore, we focus the evaluation of our method on the more practical adaptive threat model. To allow comparison to prior work, we evaluate FS and NIC under non-adaptive scenarios to match their original threat model.

E. Attack Models

We use the following adversarial example generation methods to evaluate against white-box and black-box adversaries:

1) Projected Gradient Descent Attack (PGD) [28]: PGD aims to find an AE x' for an input x that satisfies the norm-bound $\|x' - x\|_\infty \leq \epsilon$. Let S denote the L_∞ ball of radius ϵ centered at x. The attacks start at a uniformly random point $x'_0 \in S$ and generates x' with many iterations, and in each iteration, it updates the generated sample as follows:

$$x'_{i+1} = \prod_S [x'_i + \alpha \cdot \text{sign}(\nabla_x L(\text{NN}(x'_i), y))] \tag{3}$$

where $\prod$ is a clipping operator that projects an input onto S. α is step size, ∇ represents the gradient, L is cross-entropy loss, y is the correct label for x in untargeted attacks, and the target label in targeted attacks.

2) Carlini-Wagner Attack (CW) [35]: Instead of maximizing the loss function L to calculate the gradients, CW aims to find the smallest successful adversarial perturbation. That is, it constructs an effective objective function f as follows:

$$f(x') = \max(\max\{\text{NN}(x')_i : i \neq t\} - \text{NN}(x')_t, -\kappa) \tag{4}$$

where t is targeted class, $\text{NN}(x')_t$ denotes the probability of class t, and $\max\{\text{NN}(x')_i : i \neq t\}$ represents the maximal probability except class t. The parameter κ encourages to find a x' that will be classified as class t with high confidence.

3) HopSkipJump Attack (HSJA) [81]: HSJA is a query-based attack, in which the adversary crafts adversarial examples by sending queries and observing corresponding returned labels. It works by iteratively approaching the decision boundary between a natural sample (x, y) to be modified and another sample (x_t, y_t) in another category. Each round the algorithm outputs x' that is classified as y_t and the aim of the algorithm is to find an x' with the least $\|x' - x\|_p$. The performance evaluation metric we use here is the final output's L_2 distortion $\|x' - x\|_2$, which reflects the difficulty of the attack in the presence of detection mechanisms.

4) Transfer-based Attack [82]: It has been shown that AEs generated for attacking one model could attack other models that do the same task. Here, we trained shadow models using the same training dataset and model architecture listed in Table II in a normal way. Then we applied CW and PGD attacks on the shadow models to generate AEs.[3]

The perturbation bounds are set according to the literatures [28], [83], [84], and [85]. Namely, the L_2 threshold is 3 for MNIST and Imaganette, 128/255 for CIFAR10, CIFAR100 and GTSRB. The L_∞ threshold is 0.3 for MNIST, 4/255 for Imagenette, and 8/255 for CIFAR10, CIFAR100 and GTSRB.

All experiments were conducted on two servers. One is equipped with 4 Xeon Gold 5122 3.6GHz 4-core processors, 128 GB of RAM, and 4 TITAN Xp GPU cards. The other one has 2 Xeon Gold 6326 2.9GHz 16-core processors, 256 GB of RAM, and 4 NVIDIA RTX 3090 GPU cards.

F. Performance Metrics

Since the outcome of classification is 3-valued (correct, wrong, invalid), we cannot use metrics for binary outcomes such as precision and recall. Given a mechanism f, when using a set of natural samples $\mathcal{D}$ as the input, the metrics are (1) Accuracy (ACC): the portion of the sample being correctly classified, i.e. $\frac{|\{(x,y)|(x,y)\in\mathcal{D}, f(x)=y\}|}{|\mathcal{D}|}$; (2) False Invalid Rate

[3]We used Adversarial Robustness Toolbox (ART) v1.6 for both attacks (https://github.com/Trusted-AI/adversarial-robustness-toolbox).

TABLE III
MULTI-BIT LABELS' PERFORMANCE: RANDOMLY GENERATED AES

method		MNIST		CIFAR10	
		ER	DR	ER	DR
11-Cat. classifier		0.00034	0	0.89988	0
multi-bit labels	4	0.00004	0.93846	0.00220	0.91157
	5	0.00005	0.97747	0.00236	0.89473
	6	0.00002	0.98019	0.00193	0.89595
	7	0.00004	0.97101	0.00380	0.77554
	8	0.00007	0.93457	0.00340	0.88859

TABLE IV
PERFORMANCE OF ENSEMBLES

#model	MNIST		CIFAR10	
	ER	DR	ER	DR
1	0.01926	0.96949	0.00356	0.72338
2	0.00090	0.99894	0.00009	0.99612
3	0.00018	0.99980	0.00001	0.99971
4	0.00002	0.99997	0	1
5	0.00001	0.99999	0	1

(FIR): the portion of the sample being incorrectly classified as invalid, i.e. $\frac{|\{(x,y)|(x,y)\in\mathcal{D}, f(x)=\text{“Invalid”}\}|}{|\mathcal{D}|}$.

When using a set of perturbed samples $\mathcal{D}' = \{(x', y)|(x, y) \in \mathcal{D}, x' = x + \delta\}$ as the input, the metrics we use (except for the HopSkipJump black-box attack, see later section) are: (1) Error Rate (ER): the portion of samples in $\mathcal{D}'$ that are successful AEs causing misclassification, i.e. $\frac{|\{(x',y)|(x',y)\in\mathcal{D}', \|x'-x\|\leq\epsilon, f(x')\neq y, f(x')\neq\text{“Invalid”}\}|}{|\mathcal{D}'|}$; (2) Detection Rate (DR): the portion of AEs (i.e. Part III + Part IV) being detected,i.e. $\frac{|\{(x',y)|(x',y)\in\mathcal{D}', \|x'-x\|\leq\epsilon, f(x')\neq y, f(x')=\text{“Invalid”}\}|}{|\{(x',y)|(x',y)\in\mathcal{D}', \|x'-x\|\leq\epsilon, f(x')\neq y|}$.

VI. EVALUATION RESULTS

A. Ablation Study

1) Multi-Bit Labels: Multi-bit labels were proposed to allow us to train a detector without assuming a particular attack. We created ablated models that used the multi-bit labels but without the pseudorandom injection function. Each model was trained using training samples whose labels were transformed (i.e. $(x, \pi(y))$). We trained multiple models and varied the label bit-length from 4 - 8. As a comparison, for each dataset we also trained a model with 11 categories (10 normal categories plus "invalid"). The model was trained with the training dataset plus images labeled as invalid whose pixel values were uniformly sampled at random.

To evaluate the models, for each dataset (MNIST and CIFAR10), we randomly sampled 1000 samples from the test set, then randomly perturbed each sample with noise whose L_∞ norm was bounded by a threshold ϵ (0.3 for MNIST, 8/255 for CIFAR10). From each sample, we obtained 100 noisy samples, so in total 100,000 attack-independent noisy samples for each dataset. Then the noisy samples were classified by the classifiers. The results are shown in Table III. As we can see, although the 11 categories classifier has an invalid category, it did not detect any noisy samples, and some noisy samples eventually caused misclassification. When using multi-bit labels, the classifiers could detect a large portion of AEs (DR is high), and the portion of noisy samples that caused misclassification is small (ER is low). Increasing the bit-length of the label has some impact on ER and DR, but not significant, especially when ≥ 6.

2) Ensemble: To study how the number of classifiers affects the performance in more details, we compared the result when using ensembles (of pseudorandom classifiers) whose sizes varies from 1 - 5. Each model was trained with 5-bit labels. The test were done on noisy samples generated in the same way as above. The results are shown in Table IV. As we can see, more models means higher detection probability and less misclassification.

B. Performance on Natural Samples

This section shows the performance of our method on natural sample inputs using ensembles of different sizes. We tested on the test sets of each dataset and compared them to GAT, FS, and NIC. For GAT and FS, we tuned the threshold, and for NIC, we tuned the parameters of its joint OCC classifier, so their accuracy was closest to ours. The results are shown in Table V. By adjusting parameters, we aligned the methods' accuracies. In most cases, FS and NIC mislabeled fewer natural samples as "invalid" (FIR) than our method and GAT. Our method is more or less the same FIR as GAT. The results also show that when the number of classifiers increases, the classification accuracy decreases. Hence there is a trade-off between utility and security. How to improve the utility is an interesting future work.

C. Performance Under Black-Box Attacks

This section shows the performance of our method against two black-box attacks. The first attack we evaluated is HSJA. Specifically, we randomly sampled 100 images from the test sets of MNIST, CIFAR10, CIFAR100, GTSRB, and 30 from Imagenette due to slower attack speeds for larger images. For each image, we limited the attack to 10,000 queries and calculated the median L_2 distortion. Table VI displays the results. Our method used an ensemble of 3 pseudorandom classifiers, with parameters of other detection mechanisms set to match accuracies in Table V. As we can see, the median distortion required to fool our method is higher than other detection techniques, with the exception of Imagenette (row 3 of Table VI). For Imagenette, NIC achieved NaN distortion since it detected all AEs - so the exact distortion could not be measured. Importantly, HSJA cannot generate successful attacks against NIC, since NIC is evaluated in non-adaptive scenarios, where AEs crafted by HSJA to fool the base model rather than NIC itself. However, the distortion induced by our method on Imagenette is still noticeably larger than other techniques. It demonstrates our method is more difficult to attack than others.

There are two main reasons why our approach is effective. First, unlike other detection methods that use one-hot encoding, our approach utilizes multi-bit encoding to represent each class. With one-hot encoding, each of the classes is signed as a m-length vector with a single element set to 1, while all other elements are 0. In contrast, with multi-bit encoding, each class is represented as a vector with d features,

TABLE V
ACCURACY AND FALSE INVALID RATE ON NATURAL SAMPLES (THE TOP-2 FIR ARE IN BOLD FONT)

Dataset	#model	Ours		GAT-I		GAT-G		FS		NIC	
		ACC	FIR	ACC	FIR	ACC	FIR	ACC	FIR	ACC	FIR
MNIST	1	0.9866	0.0093	0.9866	0.0110	0.9865	0.0100	0.9865	**0.0081**	0.9853	**0.0076**
	2	0.9813	0.0173	0.9816	0.0169	0.9813	0.0163	0.9811	**0.0152**	0.9813	**0.0117**
	3	0.9773	0.0217	0.9792	**0.0195**	0.9777	0.0203	0.9771	0.0207	0.9784	**0.0147**
	4	0.9748	0.0244	0.9753	0.0235	0.9743	0.0242	0.9746	**0.0233**	0.9756	**0.0175**
	5	0.9709	0.0283	0.9675	0.0320	0.9706	0.0284	0.9707	**0.0273**	0.972	**0.0211**
CIFAR10	1	0.8866	0.0832	0.8872	0.0841	0.8646	**0**	0.8864	0.0510	0.8883	**0.0315**
	2	0.8510	0.1369	0.8525	0.1252	0.8506	**0.0415**	0.8508	0.1070	0.8506	**0.0726**
	3	0.8210	0.1710	0.8117	0.1713	0.8224	**0.1000**	0.8208	0.1491	0.8171	**0.1090**
	4	0.8066	0.1870	0.8117	0.1713	0.8089	**0.1217**	0.8064	0.1678	0.8066	**0.1214**
	5	0.7947	0.2003	0.7835	0.2022	0.7951	**0.1446**	0.7945	0.1841	0.791	**0.1389**
Imagenette	1	0.8501	0.1327	0.8501	0.1243	0.8501	0.1014	0.8496	**0.0504**	0.8509	**0.0529**
	2	0.7954	0.2015	0.7954	0.1880	0.7954	0.1801	0.7949	**0.1080**	0.7959	**0.1123**
	3	0.7658	0.2331	0.7658	0.2226	0.7658	0.2173	0.7653	**0.1385**	0.7651	**0.1475**
CIFAR100	1	0.6855	**0.0023**	-	-	-	-	0.6853	**0.0900**	0.6822	0.0928
	2	0.6718	0.2027	-	-	-	-	0.6716	**0.1063**	0.6747	**0.1011**
	3	0.6018	0.3464	-	-	-	-	0.6016	**0.1907**	0.6028	**0.1822**
GTSRB	1	0.9383	0.0461	-	-	-	-	0.9381	**0.0278**	0.9353	**0.0319**
	2	0.9256	0.0637	-	-	-	-	0.9254	**0.0406**	0.9236	**0.0437**
	3	0.9168	0.0751	-	-	-	-	0.9166	**0.0494**	0.917	**0.0504**

TABLE VI
HOPSKIPJUMPATTACK PERFORMANCE (MEDIAN L_2 DISTORTION ON 3 PSEUDORANDOM CLASSIFIERS)

Dataset	Ours	GAT-I	GAT-G	FS	NIC
MNIST	**6.726**	5.480	5.811	1.891	1.262
CIFAR10	**3.393**	2.510	2.302	0.677	0.229
Imagenette	56.056	40.447	38.756	2.907	**NaN**
CIFAR100	**5.007**	-	-	0.117	0.103
GTSRB	**5.649**	-	-	0.214	0.215

TABLE VII
TRANSFER-BASED ATTACKS PERFORMANCE

Dataset	Method	CW		PGD	
		ER	DR	ER	DR
MNIST	GAT-G	0.001	0.969	0	1
	GAT-I	0.001	0.975	0	1
	FS	0.008	0.741	0.002	0.985
	NIC	0.005	0.843	0.007	0.947
	Ours	0	1	0	1
CIFAR10	GAT-G	0	NaN	0.003	0.750
	GAT-I	0.001	0.937	0.024	0.974
	FS	0.032	0.757	0.094	0.830
	NIC	0.001	0.992	0.053	0.904
	Ours	0	1	0.021	0.940
Imagenette	GAT-G	0	1	0	1
	GAT-I	0.007	0.993	0.005	0.995
	FS	0.080	0.909	0.376	0.216
	NIC	0.021	0.956	0	1
	Ours	0	1	0	1
CIFAR100	GAT-G	-	-	-	-
	GAT-I	-	-	-	-
	FS	0.027	0.035	0.211	0.268
	NIC	0	1	0	1
	Ours	0	1	0	1
GTSRB	GAT-G	-	-	-	-
	GAT-I	-	-	-	-
	FS	0.025	0	0.277	0
	NIC	0.003	0.880	0.048	0.852
	Ours	0	1	0	1

where t specific features are set to 1. That means adversarial examples crafted to fool our method must simultaneously activate multiple predefined features for a given class. While ECOC [58] also uses multi-bit encoding, our approach applies it differently. ECOC aimed to classify all samples, including AEs, into valid labels. In contrast, we leverage invalid labels to detect AEs. So for our method, successful AEs must strictly satisfy specific features in order to be classified into one of the valid classes. Second, we use a sigmoid function with a rounding scheme rather than softmax to normalize the output. This causes adversaries to not only activate certain features but also deactivate others to make L_1 distance below the threshold. In this way, our method requires adversaries to induce more intricate perturbations to beat the detection compared with others.

The second attack we evaluated is Tranfer-based attack. Specifically, for each dataset, we generated 1000 AEs on shadow models to test whether those AEs can be detected. We show ER and DR in Table VII (3 pseudorandom classifiers for our method, others with matching accuracy).

As we can see, on MNIST, ImageNette, CIFAR100, and GTSRB, our method detected all AEs under both CW and PGD attacks, outperforming others. Especially on GTSRB, our method achieved detection rates of 1 for both attacks, much higher than NIC (0.88, 0.852) and FS (0, 0). On CIFAR10, the PGD attack was more effective. GAT-G attained the best performance with the lowest ER of 0.003. Our method (0.021) performed slightly better than GAT-I (0.024) and markedly better than NIC (0.053) and FS (0.094). That means our method could mitigate the transferability phenomena.

This is because multi-bit label encoding induces invalid labels, requiring carefully crafted perturbations to activate specific valid labels. One may argue that the perturbations generated on one-hot encoding struggle to transfer to our classifier. Thereby, we consider a more adaptive transfer attack [86], training the shadow model using our method without the pseudorandom injection, i.e. ensembled classifiers sharing lookup tables. As shown in Table VIII, changing the shadow models enhances transfer-based attacks. Especially on GTSRB, increasing ER from 0 to 0.021 and 0.02 for CW and PGD attacks respectively. However, the DR of our method was

TABLE VIII
ADAPTIVE TRANSFER-BASED ATTACKS PERFORMANCE

Dataset	CW		PGD	
	ER	DR	ER	DR
MNIST	0.002	0.998	0	1
CIFAR10	0.001	0.992	0.004	0.973
Imagenette	0	1	0	1
CIFAR100	0	1	0.001	0.999
GTSRB	0.021	0.894	0.020	0.938

still higher than that of other detections (in Table VII). Thus, our method demonstrated excellent resilience against transfer-based attacks by detecting almost all AEs even with adaptive shadow models.

D. Performance Under White-Box Attacks

We consider two cases differ in whether the adversary knows the pseudorandom injection keys.

In the first case, the adversary does not know the keys, so it conducts a random-key attack. In the attack, a random key k' for each pseudorandom injection is used. Then the adversary transforms a sample x into $\phi(k', x)$. The adversary then applies CW or PGD attacks (each attack runs 100 iterations) using $\phi(k', x)$ as the input and obtains $\phi(k', x)+\delta$. Finally the adversary maps $\phi(k', x) + \delta \in [0, 1]^{n\times t}$ back to $x' \in [0, 1]^n$, and outputs x'. This inverse mapping operation is done by first dividing $\phi(k', x)+\delta$ into n blocks (each block is a vector $\in [0, 1]^t$). Then for each block B_i, we set $x'_i = a$ such that (1) $\phi(k', a)$ is the closest value to B_i, and (2) the generated adversary example x' respects the perturbation limits, i.e. $\|x' - x\|_p \leq \epsilon$. In the PGD attack, the second constraint is satisfied by requiring $|a-x_i| \leq \epsilon$, so that $\|x' - x\|_\infty$ is within the pre-defined threshold ϵ. In the CW attack, to ensure the perturbation is within the L_2 threshold, we start from x'_i that has the largest difference, i.e. with the max $\|x'_i - x_i\|_2$, replace x_i with x'_i, then subtract $\|x'_i - x_i\|_2$ from the budget ϵ, until the budget runs out.

In the second case, the adversary knows the keys, so it conducts the attack exactly as above except using the true keys (i.e. k instead of k').

For each attack instance, we used 1000 samples randomly sampled from the test set. For each sample, we generated $m-1$ targeted samples, each for another category that differs from the sample's original category and starts with random initialization. We used ensembles of 3 pseudorandom classifiers for our method in the experiments. The results can be found in Table IX.

As we can see, for MNIST, GAT-G and our method (when the adversary does not know the pseudorandom injection key) performs the best, with an ER of 0. Also, we can see the CW attack is more effective against our method when the adversary knows the pseudorandom injection keys. However, the attack success rate (0.017) remains lower than that of GAT-I (0.0497). The PGD attack did not produce any successful AEs against our method (all output images were correctly classified by our method), hence the DR is NaN (denominator is 0). We had a further look into the process. We found that in the PGD attack, given $\phi(k, x)$ as the input, the output from the attack $\phi(k, x) + \delta$ was quite different from the input (on average 90% of the byte values were different). We also observed that with a high probability (> 0.99) the output could produce a different label than the original one, i.e. $f(\phi(k, x) + \delta) \neq \pi(y)$). However the changes could not be back-propagated to the images: inverse mapping of $\phi(k, x)+\delta$ would result in an image that is almost identical to the original image x. This is why the final output x' was all classified correctly. The results confirmed our intuition that pseudorandom injection can make white-box attacks ineffective.

TABLE IX
WHITE-BOX ATTACKS PERFORMANCE. FOR OUR METHOD, r AND t MEAN RANDOM OR TRUE KEYS WERE USED IN THE ATTACK

Dataset	Method	CW		PGD	
		ER	DR	ER	DR
MNIST	GAT-G	0	1	0	1
	GAT-I	0.0497	0.9496	0.0060	0.9939
	FS	0.0015	0.9982	0.0050	0.9928
	NIC	0.0121	0.9856	0.0433	0.9377
	Ours (r)	0	1	0	NaN
	Ours (t)	0.0170	0.8974	0	NaN
CIFAR10	GAT-G	0.0151	0.6046	0.0664	0.6529
	GAT-I	0.0238	0.9741	0.1397	0.8257
	FS	0.0191	0.9776	0.5776	0.4223
	NIC	0.0662	0.9269	0	1
	Ours (r)	0	1	0	NaN
	Ours (t)	0	1	0	NaN
Imagenette	GAT-G	0.0008	0.9550	0.0010	0.9513
	GAT-I	0.0064	0.9926	0.0033	0.9952
	FS	0.0454	0.9542	0.6700	0.0242
	NIC	0.0466	0.9530	0.0046	0.9933
	Ours (r)	0	1	0	1
	Ours (t)	0	1	0	1
CIFAR100	GAT-G	-	-	-	-
	GAT-I	-	-	-	-
	FS	0.1311	0.7563	0.5680	0.1670
	NIC	0.0100	0.9814	0.0302	0.9557
	Ours (r)	0.0191	0.9808	0.0175	0.9824
	Ours (t)	0.0195	0.9803	0.0458	0.9541
GTSRB	GAT-G	-	-	-	-
	GAT-I	-	-	-	-
	FS	0.9771	0	0.6920	0
	NIC	0.5092	0.4788	0.1780	0.7427
	Ours (r)	0	1	0	1
	Ours (t)	0	1	0.0005	0.9940

For CIFAR-10 and Imagenette, our method consistently outperforms others. On CIFAR100, our method achieves very good performance against the CW attack with ERs of 0.0191 (random keys) and 0.0195 (true keys), which largely outperforms FS with a 0.0497 ER, and is comparable to NIC with a 0.01 ER. Also, PGD is more effective against our method when the adversary knows the keys. However, with random keys, our method still performs the best among all methods. On GTSRB, our method detects almost all adversarial examples, while FS and NIC detect none and around 50% respectively. This is likely because our method achieves high utility on GTSRB (in Table V). To match this utility, the threshold for FS is so low that it cannot distinguish adversarial examples from natural samples. NIC performs better than FS but still has limited detection. In summary, when attackers lack the pseudorandom injection key, our method achieves the best performance on all datasets. With the true key, the attack effectiveness improves but remains limited. This is because pseudorandom injection maps images from the original space to a higher dimensional space. During attacks, adversaries need to map all points from this high-dimensional space back to the lower-dimensional

TABLE X
WHITE-BOX ATTACKS WITH THE NEW INVERSE MAPPING OPERATION (r AND t MEANS RANDOM OR TRUE KEYS WERE USED IN THE ATTACK)

Dataset	Method	CW		PGD	
		ER	DR	ER	DR
MNIST	Ours (r)	0	1	0	1
	Ours (t)	0.0173	0.8979	0.0003	0.9996
CIFAR10	Ours (r)	0	1	0	1
	Ours (t)	0	1	0.0005	0.9890
Imagenette	Ours (r)	0	1	0	1
	Ours (t)	0	1	0	1
CIFAR100	Ours (r)	0.0191	0.9808	0.0176	0.9823
	Ours (t)	0.0195	0.9803	0.0458	0.9541
GTSRB	Ours (r)	0	1	0	1
	Ours (t)	0	1	0	1

original space. Since one-to-one mapping is infeasible, attackers cannot accurately map all high-dimensional points back to the original space. This makes successful attacks difficult, consistent with our Theorem 1.

One may argue that the good results in Table IX are largely due to the inverse mapping suppressed the changes made by the attack algorithms. We hence also tested with another inverse mapping strategy. We changed condition (1), such that now for each block B_i, it is only mapped to x_i if $B_i = \phi(k, x)$, i.e. the attack algorithm did not modify it at all; if $B_i \neq \phi(k, x)$, we require B_i to be mapped to a value $a \neq x_i$ such that $\phi(k, a)$ is the closest value to B_i. Using this inverse mapping, almost all the byte values in an image x' produced by the PGD attack are different from x. The performance of the white-box attacks with the new inverse mapping is shown in Table X. As We can see, the attacks only have limited improvement in PGD attacks. We have designed and tested a more complex adaptive attack algorithm, but it was also not effective (see discussion and results in Section VII).

VII. DISCUSSIONS

Although we evaluated adaptive attacks, a key question remains: are those adaptive attacks effective enough? Many works [12], [36], [43], [48], [86] have focused on designing more effective adaptive attacks, offering new perspectives to rethink our evaluations. In this section, we take the viewpoint of an adaptive attack designer to re-examine our evaluations. Specifically, building on two state-of-the-art works focused on adaptive attacks [12], [36], we discuss whether the attacks evaluated in Section VI were rigorous enough, and if more advanced and effective adaptive attacks could compromise our method.

A. Guidelines for Adaptive Attacks

Trammer et al. [12] proposed six themes that all defense evaluations should follow when designing adaptive attacks and assessing robustness. We discuss these themes and how our evaluations incorporate them:

- **Strive for simplicity.** This is the main theme in the paper — emphasizing the priority of simplicity when constructing adaptive attacks and introducing the following themes to simplify yet strengthen attacks specifically.
- **Attack the full defense.** Any components, especially preprocessing functions, should be targeted if possible. Our evaluation follows this by targeting all components of our defense method: the ensemble of pseudorandom classifiers, pseudorandom injection with true keys, and lookup tables for multi-bit encoding.
- **Identify and target important defense parts.** For complex defenses with many sub-components, inspecting these parts to find the ones that truly enable defense can simplify and strengthen attacks. To do it, we performed an ablation study in Section VI-A to analyze how multi-bit encoding and ensembling contribute to our method. In Section VII-B, we elaborated on the gradient masking caused by pseudorandom injection. We identify that multi-bit encoding could detect AEs against the black-box attacks, ensemble enhanced it and pseudorandom injection makes it robust against white-box attacks.
- **Adapt the objective to simplify the attack.** Adapting the attack objective (loss function) to its best could let the attack craft successful adversarial examples easily. We adapt the attack objective function of both PGD and CW to our method as follows. For PGD (Eq. 3), we use binary cross entropy as the loss f. For CW (Eq. 4), we modify the $\text{NN}(x)$ term representing the confidence score to be the distances between the model output and those valid labels. When calculating distances, we use L_2 norm instead of L_1 norm to smooth the objective function. We set $\kappa = 0.1$, matching the L_1 distance threshold $\tau = 0$. Furthermore, since untargeted attacks easily fall into invalid space, we perform multi-targeted attacks by generating $m - 1$ targeted samples per input in Section VI-D.
- **Ensure the loss function is consistent**. This theme states that the attack loss function should align with the attack target. We verify that our adapted loss functions are consistent with pseudorandom classifiers. For PGD attacks, using binary cross entropy as the L is consistent with the loss function used to train the pseudorandom classifiers. For CW attacks, modifying the confidence score term to be distance and matching κ with the distance threshold τ makes the attack objective function consistent with multi-bit encoding.
- **Optimize the loss function with different methods.** Given a useful loss function, choosing an appropriate attack algorithm is critical. Rather than selecting a single best algorithm, we evaluated a diverse range of attacks, including decision-based (HSJA), transfer-based (CW and PGD), and gradient-based (CW and PGD) attacks.
- **Use a strong adaptive attack for adversarial training.** This theme states that if the attacks used in adversarial training fail to reliably find AEs, the defense will not withstand stronger attacks. However, this consideration is beyond the scope of our work, since our method does not involve adversarial training.

Overall, our adaptive attacks used in Section VI satisfy those themes.

B. Gradient Masking

In pseudorandom classifiers, we use a pseudorandom injection function to counter white-box attacks. It works by making

gradients useless for generating adversarial examples. In the past, many defenses targeting the gradient were proposed [60], [87], [88]. However, Athalye et al. [36] developed three adaptive attack techniques to circumvent them. Here, we discuss the feasibility of each adaptive attack technique on a pseudorandom classifier:

- **Expectation over Transformation (EOT).** It is designed toward stochastic gradients caused by random noise-based defenses. In those defenses, random noise is sampled at runtime from a distribution and is added to the base model parameters or the input. In this way, the same input could result in different gradients each time querying the base model. AEs generated using those gradients would fail to attack due to different noise injection EOT attack works by estimating the expected gradients from multiple gradients computed in multiple runs using the same input. The expected gradients can make random noise-based defenses ineffective. However, our method is different. The pseudorandom injection is deterministic once the key is fixed, i.e. one input corresponds to only one transformed input. The idea of using the pseudorandom injection is to prevent gradients from back-propagating to the original input, rather than make the gradient uncertain. Hence collecting multiple gradients from multiple runs does not help.
- **Backward Pass Differentiable Approximation (BPDA).** It is designed towards shattered gradients, which is caused by non-differentiable operations [36]. BPDA finds a differentiable approximation to replace non-differential layers on the backward pass to generate AEs. Following this idea, we investigated whether the pseudorandom injection function (when knowing the key) can be replaced by a differentiable function. The pseudorandom injection $\phi(k, \cdot)$ maps an scalar in $[0, 1]$ to a vector $[0, 1]^d$ deterministically, which means we can also regard it as d mappings $\phi_1, \ldots, \phi_d$ such that each $\phi_i : [0, 1] \rightarrow [0, 1]$ maps a scalar to a scalar (in the i-th slot of the d-length vector) deterministically. For example, in our instantiation, the pseudorandom injection $\phi(k, \cdot)$ maps a byte to a 256-bit (32 bytes) string, which we can regard as 32 mappings from bytes to bytes. We can enumerate all byte values $b \in \{0, 1\}^8$, and corresponding $\phi_i(k, b)$. Then for each ϕ_i, we can interpolate a polynomial using $(b, \phi_i(k, b))$ pairs. We can use these polynomials to approximate the pseudorandom injection function. However, this does not work, because the approximation leads to exploding gradients. We show the plot of one of the polynomial functions and its gradient in Figure 5. As shown in Figure 5, the value of the gradients ranges from 10^{-200} to 10^{200}, which cannot be handled properly through backpropagation.
- **Reparameterization.** The idea is that since there is an operation that projects samples to some manifold in a specific manner, we could design a function to return points exclusively on the manifold. In our method, although the pseudorandom function maps the sample into a higher-dimension space, which leads to exploding gradients, an adversary could choose to only focus on the part holding valid samples instead of all, which is just what white-box attacks in Section VI-D did.

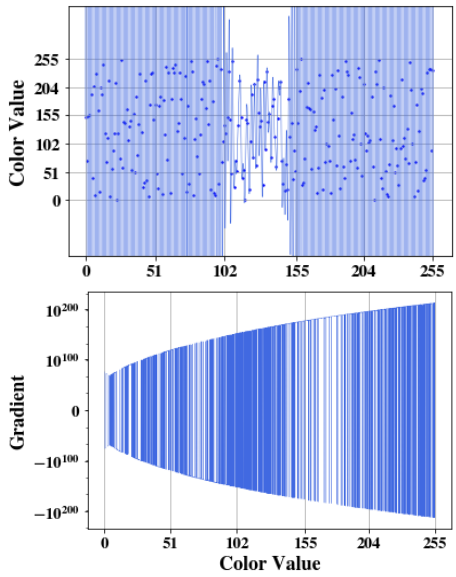


Fig. 5. Approximated polynomial function (the above) and its gradients function (the below).

TABLE XI

ADAPTIVE ATTACKS PERFORMANCES (t MEANS TRUE KEYS WERE USED IN THE ATTACK)

	Method	CW		PGD	
		ER	DR	ER	DR
MNIST	Ours (t)	0	1	0.0002	0.9997
CIFAR10	Ours (t)	0	1	0	1
Imagenette	Ours (t)	0	NaN	0	1
CIFAR100	Ours (t)	0	NaN	0	1
GTSRB	Ours (t)	0	1	0.0005	0.9942

As we can see from the above discussion, EOT and BPDA do not apply to our method.

C. Advanced Adaptive Attack Algorithm

Inspired by Reparameterization, we design an adaptive attack. The idea is to update each step with values whose transformed values is nearest to gradient-updated input, in the hope that the pseudorandom injection function could be bypassed. More specifically, given a sample x, we take its transformed form $\phi(k, x)$ as the input to the neural network, then for each attack iteration, we divided gradient-updated input into n blocks. For each block, we find a value satisfied two conditions simultaneously: (1) within norm threshold (2) has the smallest distance with corresponding gradient-updated input, to replace corresponding values in x to form $\hat{x}$. Specifically, we use the inverse mapping operations InverseMap (PGD) and InverseImpMap (CW) shown in Algorithm 1. Then we get corrected input $\hat{x}$ to start the next attack iteration. The whole algorithm is shown in Algorithm 1.

We tested the adaptive attack with 1000 random samples for MNIST and CIFAR10, and 100 random samples for Imagenette, GTSRB and CIFAR100. For each sample, adaptive attack generated $m - 1$ targeted AEs. The results are shown in Table XI. Compared to Table IX achieved by the white-box attack in Section VI-D, the attack attained a comparable

Algorithm 1 Adaptive Attack

Require: unperturbed image x, targeted label y, pseudorandom injection function $\phi(k, \cdot)$, iterations N, perturbation size ϵ

Ensure: adversarial example x_{adv}

function INVERSEMAP(x, x'_t)
  for ν, ρ in x, x'_t **do**
    $O \leftarrow \{\mu : \|\mu - \nu\|_\infty \leq \epsilon\}$ ▷ value alternative set under ϵ-L_∞ attack.
    $\hat{\nu} \leftarrow arg\min_\mu \|\phi(k, \mu) - \rho\|_2, \mu \in O$
    $x_t^{adv} \leftarrow x_t^{adv} \| \hat{\nu}$
  end for
  $x'_t \leftarrow \phi(k, x_t^{adv})$
  return x_t^{adv}, x'_t
end function

function INVERSEIMPMAP(x, x'_t, x'_{t-1})
  $I \leftarrow argsort(\{\|\rho_t - \rho_{t-1}\|_2$ **for** ρ_t, ρ_{t-1} in $x'_t, x'_{t-1}\})$ ▷ sort color value indexes by descending changed weights
  $x_t^{adv} \leftarrow x$
  $\alpha \leftarrow \epsilon$
  for i in I **do**
    $\nu, \rho \leftarrow x[i], x'_t[i]$
    $O \leftarrow \{\mu : \|\mu - \nu\|_2 \leq \alpha\}$ ▷ value alternative set under ϵ-L_2 attack.
    $\hat{\nu} \leftarrow arg\min_\mu \|\phi(k, \mu) - \rho\|_2, \mu \in O$
    $x_t^{adv}[i] \leftarrow \hat{\nu}$
    $\alpha \leftarrow \epsilon - \|x - x_t^{adv}\|_2$
    if $\alpha \leq 0$ **then**
      break
    end if
  end for
  return x_t^{adv}, x'_t
end function

function ATTACK(x)
  $x_0^{adv} \leftarrow x + \delta, \delta \sim \mathcal{N}(0, 1)$
  $x'_0 \leftarrow \phi(k, x_0^{adv})$
  for $i \leftarrow 1$ to N **do**
    if PGD attack **then**
      $L \leftarrow PGDLoss(x'_{i-1}, y)$
      $x'_i \leftarrow \nabla_{x'_{i-1}} L + \tilde{x}$
      $x_i^{adv}, x'_i \leftarrow$InverseMap($x$, x'_i)
    else if CW attack **then**
      $L \leftarrow CWLoss(x'_{i-1}, y)$
      $x'_i \leftarrow from\_tanh(\nabla_{x'_{i-1}} L + to\_tanh(x'_{i-1}))$
      $x_i^{adv}, x'_i \leftarrow$InverseImpMap($x$, x'_i, x'_{i-1})
    end if
  end for
  return x_I^{adv}
end function

successful rate in PGD attacks and a lower successful rate in CW. This is because the inverse mapping operations impede gradient convergence. This suggests that the white-box attack (in Section VI-D) is more efficient.

VIII. CONCLUSION

We propose a new detection-based defense method against DNNs adversarial examples, called pseudorandom classifier. We develop a cryptographic technique called pseudorandom injection as a novel input transformation method. Through theoretical analysis, we prove that white-box adversaries have negligible advantage in crafting adversarial examples under our defense. We evaluate the method in an adaptive attack scenario, which is a stronger threat model. The results show that our method achieves comparable or even better performance than state-of-the-art defenses against both black-box and white-box attacks. This work represents an interesting first step in exploring the intersection between cryptography and AI security. However, there are some limitations that need further research: (i) Training the pseudorandom classifier requires extra input transformations, which generally require additional time costs. (ii) For complex classification tasks, there is still room to further reduce utility loss for better accuracy. In the future, we will investigate how to improve pseudorandom classifiers, and also explore whether cryptography can help solve other security issues in DNNs.

REFERENCES

[1] C. Szegedy et al., "Intriguing properties of neural networks," in *Proc. 2nd Int. Conf. Learn. Represent. (ICLR)*, 2014, pp. 1–10.

[2] F. Tramèr, N. Carlini, W. Brendel, and A. Madry, "On adaptive attacks to adversarial example defenses," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2020, pp. 1633–1645.

[3] L. Smith and Y. Gal, "Understanding measures of uncertainty for adversarial example detection," in *Proc. 34th Conf. Uncertainty Artif. Intell. (UAI)*, 2018, pp. 1633–1645.

[4] Q. Huang, I. Katsman, Z. Gu, H. He, S. Belongie, and S.-N. Lim, "Enhancing adversarial example transferability with an intermediate level attack," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 4732–4741.

[5] A. J. Bose et al., "Adversarial example games," in *Proc. Annu. Conf. Neural Inf. Process. Syst (NIPS)*, 2020, pp. 8921–8934.

[6] X. Yin, S. Kolouri, and G. K. Rohde, "GAT: Generative adversarial training for adversarial example detection and robust classification," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–27.

[7] S. Jetley, N. A. Lord, and P. H. S. Torr, "With friends like these, who needs adversaries?" in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2018, pp. 1–11.

[8] J. Gilmer et al., "Adversarial spheres," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–15.

[9] J. Gilmer, N. Ford, N. Carlini, and E. D. Cubuk, "Adversarial examples are a natural consequence of test error in noise," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 2280–2289.

[10] A. Fawzi, H. Fawzi, and O. Fawzi, "Adversarial vulnerability for any classifier," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2018, pp. 1–10.

[11] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, "Adversarial examples are not bugs, they are features," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2019, pp. 1–12.

[12] F. Tramèr, A. Kurakin, N. Papernot, I. J. Goodfellow, D. Boneh, and P. D. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–22.

[13] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–17.

[14] Y. Wang, X. Ma, J. Bailey, J. Yi, B. Zhou, and Q. Gu, "On the convergence and robustness of adversarial training," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 1–13.

[15] A. Shafahi, M. Najibi, Z. Xu, J. P. Dickerson, L. S. Davis, and T. Goldstein, "Universal adversarial training," in *Proc. 34th AAAI Conf. Artif. Intell.*, 2020, pp. 5636–5643.

[16] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. D. McDaniel, "On the (statistical) detection of adversarial examples," 2017, *arXiv:1702.06280*.

[17] J. Lu, T. Issaranon, and D. A. Forsyth, "SafetyNet: Detecting and rejecting adversarial examples robustly," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 446–454.

[18] P. Sperl, C.-Y. Kao, P. Chen, X. Lei, and K. Böttinger, "DLA: Dense-layer-analysis for adversarial example detection," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Sep. 2020, pp. 198–215.

[19] P. Yang, J. Chen, C. Hsieh, J. Wang, and M. I. Jordan, "ML-LOO: Detecting adversarial examples with feature attribution," in *Proc. 34th AAAI Conf. Artif. Intell.*, 2020, pp. 6639–6647.

[20] G. Cohen, G. Sapiro, and R. Giryes, "Detecting adversarial samples using influence functions and nearest neighbors," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 14441–14450.

[21] F. Sheikholeslami, A. Lotfi, and J. Z. Kolter, "Provably robust classification of adversarial examples with detection," in *Proc. 9th Int. Conf. Learn. Represent. (ICLR)*, 2021, pp. 1–16.

[22] B. Huang, Y. Wang, and W. Wang, "Model-agnostic adversarial detection by random perturbations," in *Proc. 28th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2019, pp. 4689–4696.

[23] H.-Y. Chen et al., "Improving adversarial robustness via guided complement entropy," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 4880–4888.

[24] X. Zhang, J. Wang, T. Wang, R. Jiang, J. Xu, and L. Zhao, "Robust feature learning for adversarial defense via hierarchical feature alignment," *Inf. Sci.*, vol. 560, pp. 256–270, Jun. 2021.

[25] Z. Zhang, X. Gao, S. Liu, B. Peng, and Y. Wang, "Energy-based adversarial example detection for SAR images," *Remote Sens.*, vol. 14, no. 20, p. 5168, Oct. 2022.

[26] J. Wang, C. Wang, Q. Lin, C. Luo, C. Wu, and J. Li, "Adversarial attacks and defenses in deep learning for image recognition: A survey," *Neurocomputing*, vol. 514, pp. 162–181, Dec. 2022.

[27] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. 3rd Int. Conf. Learn. Represent.*, 2015, pp. 1–11.

[28] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–28.

[29] H. Zhang and J. Wang, "Defense against adversarial attacks using feature scattering-based adversarial training," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–11.

[30] G. Liu, I. Khalil, and A. Khreishah, "GanDef: A GAN based adversarial training defense for neural network classifier," in *Proc. 34th IFIP TC Int. Conf., SEC*, 2019, pp. 19–32.

[31] S. Park and J. So, "On the effectiveness of adversarial training in defending against adversarial example attacks for image classification," *Appl. Sci.*, vol. 10, no. 22, p. 8079, Nov. 2020.

[32] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," 2016, *arXiv:1611.03814*.

[33] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 582–597.

[34] S. Ma, Y. Liu, G. Tao, W. Lee, and X. Zhang, "NIC: Detecting adversarial samples with neural network invariant checking," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2019, pp. 1–15.

[35] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 39–57.

[36] A. Athalye, N. Carlini, and D. A. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 274–283.

[37] Z. Liu et al., "Feature distillation: DNN-oriented JPEG compression against adversarial examples," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 860–868.

[38] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. L. Yuille, "Mitigating adversarial effects through randomization," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–16.

[39] B. Sun, N.-H. Tsai, F. Liu, R. Yu, and H. Su, "Adversarial defense by stratified convolutional sparse coding," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11439–11448.

[40] D. Hwang, E. Lee, and W. Rhee, "AID-purifier: A light auxiliary network for boosting adversarial defense," *Neurocomputing*, vol. 541, Jul. 2023, Art. no. 126251.

[41] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2018, pp. 1–15.

[42] D. Meng and H. Chen, "MagNet: A two-pronged defense against adversarial examples," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Oct. 2017, pp. 135–147.

[43] N. Carlini and D. A. Wagner, "Magnet and 'efficient defenses against adversarial attacks' are not robust to adversarial examples," 2017, *arXiv:1711.08478*.

[44] C. Xiao, P. Zhong, and C. Zheng, "Enhancing adversarial defense by k-winners-take-all," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–30.

[45] K. Roth, Y. Kilcher, and T. Hofmann, "The odds are odd: A statistical test for detecting adversarial examples," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 5498–5507.

[46] T. Pang, K. Xu, and J. Zhu, "Mixup inference: Better exploiting mixup to defend adversarial attacks," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, Addis Ababa, Ethiopia, Apr. 2020, pp. 1–14.

[47] E. Raff, J. Sylvester, S. Forsyth, and M. McLean, "Barrage of random transforms for adversarially robust defense," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 6521–6530.

[48] C. Sitawarin, Z. J. Golan-Strieb, and D. A. Wagner, "Demystifying the adversarial robustness of random transformation defenses," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2022, pp. 20232–20252.

[49] Y. Gao, I. Shumailov, K. Fawaz, and N. Papernot, "On the limitations of stochastic pre-processing defenses," in *Proc. NIPS*, 2022, pp. 1–15.

[50] R. Pinot, R. Ettedgui, G. Rizk, Y. Chevaleyre, and J. Atif, "Randomization matters how to defend against strong adversarial attacks," in *Proc. 37th Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 7717–7727.

[51] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 656–672.

[52] J. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 1310–1320.

[53] B. Li, C. Chen, W. Wang, and L. Carin, "Certified adversarial robustness with additive noise," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2019, pp. 9459–9469.

[54] R. Pinot et al., "Theoretical evidence for adversarial robustness through randomization," in *Proc. Adv. Neural Inf. Process. Syst. (NPS)*, 2019, pp. 11838–11848.

[55] A. Blum, T. Dick, N. Manoj, and H. Zhang, "Random smoothing might be unable to certify $l(\infty)$ robustness for high-dimensional images," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 211:1–211:21, 2020.

[56] H. Salman, M. Sun, G. Yang, A. Kapoor, and J. Z. Kolter, "Denoised smoothing: A provable defense for pretrained classifiers," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 21945–21957.

[57] N. Carlini, F. Tramèr, K. D. Dvijotham, L. Rice, M. Sun, and J. Z. Kolter, "(certified!!) adversarial robustness for free!" in *Proc. 11th Int. Conf. Learn. Represent. (ICLR)*, 2023, pp. 1–14.

[58] G. Verma and A. Swami, "Error correcting output codes improve probability estimation and adversarial robustness of deep neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8643–8653.

[59] T. Pang, C. Du, Y. Dong, and J. Zhu, "Towards robust detection of adversarial examples," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2018, pp. 4584–4594.

[60] X. Ma et al., "Characterizing adversarial subspaces using local intrinsic dimensionality," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–15.

[61] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, "Detecting adversarial samples from artifacts," 2017, *arXiv:1703.00410*.

[62] A. Abusnaina et al., "Adversarial example detection using latent neighborhood graph," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 7667–7676.

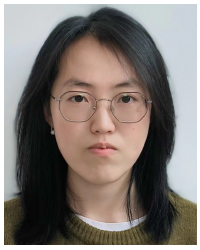
[63] S. Bulusu, B. Kailkhura, B. Li, P. K. Varshney, and D. Song, "Anomalous instance detection in deep learning: A survey," 2020, *arXiv:2003.06979*.

[64] A. Aldahdooh, W. Hamidouche, S. A. Fezza, and O. Déforges, "Adversarial example detection for DNN models: A review and experimental comparison," *Artif. Intell. Rev.*, vol. 55, no. 6, pp. 4403–4462, Aug. 2022.

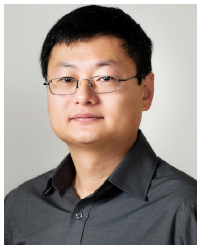
[65] F. Tramèr, "Detecting adversarial examples is (nearly) as hard as classifying them," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2022, pp. 21692–21702.

[66] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, "Detecting adversarial samples from artifacts," 2017, *arXiv:1703.00410*.

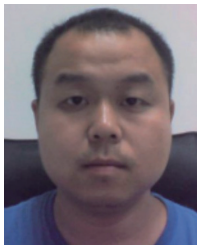
[67] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–12.
[68] D. Hendrycks and K. Gimpel, "Early methods for detecting adversarial images," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–9.
[69] X. Li and F. Li, "Adversarial examples detection in deep networks with convolutional filter statistics," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5775–5783.
[70] S. G. Müller and F. Hutter, "Trivialaugment: Tuning-free yet state-of-the-art data augmentation," 2021, *arXiv:2103.10158*.
[71] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 3319–3328.
[72] D. Smilkov, N. Thorat, B. Kim, F. B. Viégas, and M. Wattenberg, "Smoothgrad: Removing noise by adding noise," 2017, *arXiv:1706.03825*.
[73] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.
[74] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. 0, 2009.
[75] J. Howard and S. Gugger, "Fastai: A layered API for deep learning," *Information*, vol. 11, no. 2, p. 108, Feb. 2020, doi: 10.3390/info11020108.
[76] "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Netw.*, vol. 32, pp. 323–332, Aug. 2012.
[77] Y. LeCun et al., "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.
[78] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–14.
[79] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[80] Y. Sharma and P. Chen, "Bypassing feature squeezing by increasing adversary strength," 2018, *arXiv:1803.09868*.
[81] J. Chen, M. I. Jordan, and M. J. Wainwright, "HopSkipJumpAttack: A query-efficient decision-based attack," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1277–1294.
[82] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: From phenomena to black-box attacks using adversarial samples," 2016, *arXiv:1605.07277*.
[83] L. Schott, J. Rauber, M. Bethge, and W. Brendel, "Towards the first adversarially robust neural network model on MNIST," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–17.
[84] S. Gowal, P. Huang, A. van den Oord, T. Mann, and P. Kohli, "Self-supervised adversarial robustness for the low-label, high-data regime," in *Proc. 9th Int. Conf. Learn. Represent. (ICLR)*, 2021, pp. 1–19.
[85] C. Mao, Z. Zhong, J. Yang, C. Vondrick, and B. Ray, "Metric learning for adversarial robustness," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2019, pp. 478–489.
[86] K. Mahmood, D. Gurevin, M. van Dijk, and P. H. Nguyen, "Beware the black-box: On the robustness of recent defenses to adversarial examples," *Entropy*, vol. 23, no. 10, p. 1359, Oct. 2021.
[87] J. Buckman, A. Roy, C. Raffel, and I. J. Goodfellow, "Thermometer encoding: One hot way to resist adversarial examples," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–22.
[88] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, "PixelDefend: Leveraging generative models to understand and defend against adversarial examples," 2017, *arXiv:1710.10766*.



Boyu Zhu (Student Member, IEEE) received the B.S. degree in software engineering from Donghua University, Shanghai, China, in 2016. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Nanjing University, Nanjing, China. Her research interests include model security, data privacy, and deep learning.



Changyu Dong (Member, IEEE) received the Ph.D. degree from Imperial College London. He is currently a Professor with the Institute of Artificial Intelligence, Guangzhou University. He has authored more than 50 publications in international journals and conferences. His research interests include applied cryptography, data privacy, AI security, and blockchain. His recent work focuses mostly on designing practical secure computation protocols. The application domains include secure cloud computing and privacy-preserving data mining.



Yuan Zhang (Member, IEEE) received the B.S. degree in automation from Tianjin University, Tianjin, China, in 2005, the M.S.E. degree in software engineering from Tsinghua University, Beijing, China, in 2009, and the Ph.D. degree in computer science from The State University of New York at Buffalo, Buffalo, NY, USA, in 2013. He is currently an Associate Professor with the State Key Laboratory for Novel Software Technology, Nanjing University. His research interests include security, privacy, and economic incentives.

Yunlong Mao (Member, IEEE) received the B.S. and Ph.D. degrees in computer science from Nanjing University in 2013 and 2018, respectively. He is currently an Associate Professor with the State Key Laboratory for Novel Software Technology, Nanjing University. His current research interests include security, privacy, machine learning, and blockchain.

Sheng Zhong (Fellow, IEEE) received the B.S. and M.S. degrees in computer science from Nanjing University, Nanjing, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from Yale University, New Haven, CT, USA, in 2004. His research interests include security, privacy, and economic incentives.