

# Poisoning attack detection scheme based on data integrity sampling audit algorithm in neural network

Zhao Ningning Jiang Rui

(School of Cyber Science and Engineering, Southeast University, Nanjing 210096, China)

**Abstract:** To address the issue that most existing detection and defense methods can only detect known poisoning attacks but cannot defend against other types of poisoning attacks, a poisoning attack detecting scheme with data recovery (PAD-DR) is proposed to effectively detect the poisoning attack and recover the poisoned data in a neural network. First, the PAD-DR scheme can detect all types of poisoning attacks. The data sampling detection algorithm is combined with a real-time data detection method for input layer nodes using a neural network so that the system can ensure the integrity and availability of the training data to avoid being changed or corrupted. Second, the PAD-DR scheme can recover corrupted or poisoned training data from poisoning attacks. Cauchy Reed-Solomon (CRS) code technology can encode training data and store them separately. Once the poisoning attack is detected, the original training data is recovered, and the system may get data from any  $k$  nodes from all  $n$  stores to recover the original training data. Finally, the security objectives of the PAD-DR scheme to withstand poisoning attacks, resist forgery and tampering attacks, and recover the data accurately are formally proved.

**Key words:** poisoning attack; neural network; deep learning; data integrity sampling audit

**DOI:** 10.3969/j.issn.1003-7985.2023.03.012

Deep learning (DL)<sup>[1]</sup> is the foundation for several modern artificial intelligence applications. It has rapidly matured and entered safety-critical applications such as self-driving cars<sup>[2-3]</sup>, drones, and robots<sup>[4-5]</sup>. Deep neural network (DNN)<sup>[6]</sup> is a machine learning technique that tries to imitate the neurons of the human brain to transmit information and interpret data.

However, deep learning technology also faces some security threats due to the unique vulnerability of deep neural networks. Recent research<sup>[7]</sup> reveals that attackers can

inject malicious data into training data, called poisoning attack, to affect the learning performance of deep neural networks and produce incorrect results. Biggio et al.<sup>[8]</sup> proposed a label-flipping (LF) attack, adding some Boolean random variables as label noise to flip labels. Thus, the scheme<sup>[8]</sup> could make the model output wrong classification results and improve the attack success rate. To expand the attack range and make the label-flipping attack on the deep learning system, Muñoz-González et al.<sup>[9]</sup> designed an algorithm to generate more poisoning samples using reverse gradient optimization technology. Liu et al.<sup>[10]</sup> utilized an asymmetric vector to generate the poisoning labels and applied generative adversarial network (GAN) technology to generate the poisoning images more effectively. In 2016, Alberti et al.<sup>[11]</sup> modified a single pixel in the training image to make the neural network produce incorrect results during the test stage. In 2018, Shafahi et al.<sup>[12]</sup> proposed a targeted clean-label poisoning (TCL) attack. The authors utilized an iterative algorithm to create poisoned samples similar to the original samples being close to the target samples in feature space. Jagielski et al.<sup>[13]</sup> first considered poisoning attacks against linear regression models (R attacks) and formulated the poisoning attack as a bi-level optimization problem.

Unfortunately, the defense methods against poisoning attacks are not systematic enough. Zhang et al.<sup>[14]</sup> designed a strong defense scheme called DUTI to deal with the label-flipping attack. Given a small portion of the trusted items, the DUTI scheme could learn the difference between the distribution of trusted items and training samples to find the potentially corrupted labels and then get the corrupted labels to a domain expert for further examination. To defend against tool command language (TCL) attacks<sup>[12]</sup>, Peri et al.<sup>[15]</sup> proposed a scheme named DeepKNN to remove poisoning samples. In Ref. [15], the authors compared the class labels of each testing sample with its  $k$  neighbors. If most neighbor samples differed from the testing sample, this testing sample should be removed. Using the cooperative deep learning system, Shen et al.<sup>[16]</sup> proposed a defense method called AUROR to defend against poisoning attacks. The AUROR scheme could automatically identify and display the process of abnormal distribution features and then detect malicious users in the system according to the abnormal features. Based on a prior observation that poisoned samples may

**Received** 2023-03-10, **Revised** 2023-08-10.

**Biographies:** Zhao Ningning (1998—), female, graduate; Jiang Rui (corresponding author), male, doctor, professor, R. Jiang@seu.edu.cn.

**Foundation items:** The National Natural Science Foundation of China (No. 61372103), the Natural Science Foundation of Jiangsu Province (No. BK20201265), the Project of the National Engineering Research Center of Classified Protection and Safeguard Technology for Cyber Security (No. C21640-2).

**Citation:** Zhao Ningning, Jiang Rui. Poisoning attack detection scheme based on data integrity sampling audit algorithm in neural network[J]. Journal of Southeast University (English Edition), 2023, 39(3): 314 – 322. DOI: 10.3969/j.issn.1003-7985.2023.03.012.

exploit spare capacity in the neural network, Liu et al.<sup>[17]</sup> proposed a fine-pruning technology as a defense method to enhance the security of deep neural networks by eliminating some dormant neurons to turn off poisoning behaviors.

Considering that the previous schemes<sup>[14-17]</sup> can only defend against specific attacks, Diakonikolas et al.<sup>[18]</sup> proposed a robust algorithm named Server that could defend against poisoning attacks in classification and regression models. Chen et al.<sup>[19]</sup> proposed a generic and attack-agnostic defense approach called De-Pois. The key idea of De-Pois was to train a mimic model to imitate the behavior of the target model with clean samples.

In this study, we proposed a poisoning attack-detecting scheme with data recovery (PAD-DR). First, we designed a data sampling audit algorithm and combined it with a real-time data detection method to detect all kinds of poisoning attacks. Second, we applied Cauchy Reed-Solomon (CRS) code technology to encode training data and store them on multiple servers to recover the corrupted training data. Finally, we formally proved the security goals of our PAD-DR scheme to withstand poisoning attacks and to recover the data accurately.

## 1 Preliminaries

### 1.1 Bilinear mapping

**Definition 1** Let  $G_1$  and  $G_2$  be multiplicative cyclic groups of a large prime order  $p$ ; a pairing is a bilinear map  $e: G_1 \times G_1 \rightarrow G_2$ . It satisfies the following properties:

- 1) Bilinear.  $e(u^a, v^b) = e(u, v)^{ab}$ ,  $\forall u, v \in G_1; a, b \in \mathbb{Z}_p$ .
- 2) Non-degeneracy.  $\exists u, v \in G_1$ , thus  $e(u, v) \neq 1 \in G_1$ .
- 3) Computability.  $\forall u, v \in G_1$ ; there is a polynomial time algorithm to calculate  $e(u, v)$ .
- 4) Safety. It is difficult to calculate the discrete logarithm problem in  $G_1$  and  $G_2$ .

### 1.2 Discrete logarithm problem (DLP)

**Definition 2**<sup>[20]</sup> Let  $\alpha \in \mathbb{Z}_p^*$  and  $G_1$  be a multiplicative

cyclic group known as  $g, g^\alpha \in G_1$ , for any polynomial time adversary, the advantage of solving the value  $\alpha$  is negligible. Using a probabilistic polynomial time (PPT) algorithm  $A$  successfully solves the DLP  $\text{Adv}^{\text{DL}} = \Pr[A(g, g^\alpha) = \alpha: \alpha \in \mathbb{Z}_p^*] < \varepsilon$ , which is negligible. The probability comes from the random selection of  $\alpha$  on  $\mathbb{Z}_p^*$  and the random selection of algorithm  $A$ .

### 1.3 Computational Diffie-Hellman problem

**Definition 3**<sup>[21]</sup> Let  $a, b \in \mathbb{Z}_p^*$ , and  $G_1$  be a multiplicative cyclic group, given  $P, P^a, P^b \in G_1$ , for any polynomial time adversary, it is infeasible in calculation to solve  $P^{ab} \in G_1$ . Using a PPT algorithm  $A$  can successfully solve the CDH problem  $\text{Adv}^{\text{CDHP}} = \Pr[A(P, P^a, P^b) = P^{ab}: a, b \in \mathbb{Z}_p^*] < \varepsilon$ , which is negligible. The probability comes from the random selection of  $a, b$  on  $\mathbb{Z}_p^*$  and the random selection of the algorithm  $A$ .

### 1.4 Co-computational bilinear Diffie-Hellman problem (CO-CDH)

**Definition 4**<sup>[22]</sup> Let  $a \in \mathbb{Z}_p^*$  and  $G_1, G_2$  be two multiplicative cyclic groups, given  $P, P^a \in G_1, Q \in G_2$ , for any polynomial time adversary, it is infeasible in calculation to solve  $Q^a \in G_2$ . Using a PPT algorithm  $A$  successfully solves the Co-CDH problem  $\text{Adv}^{\text{Co-CDH}} = \Pr[A(P, P^a, Q) = Q^a: a \in \mathbb{Z}_p^*] < \varepsilon$ , which is negligible.

## 2 Proposed PAD-DR Scheme

### 2.1 System model

In our PAD-DR scheme, as shown in Fig. 1, the system consists of five types of entities: system administrator (SA), third party auditor (TPA), coded data stores (CDS<sub>*i*</sub>), training data store (TDS), and nodes of the neural network for input layer (NNs). SA is responsible for processing the original training data. TPA is responsible for generating challenges to detect whether poisoning attacks are launched.

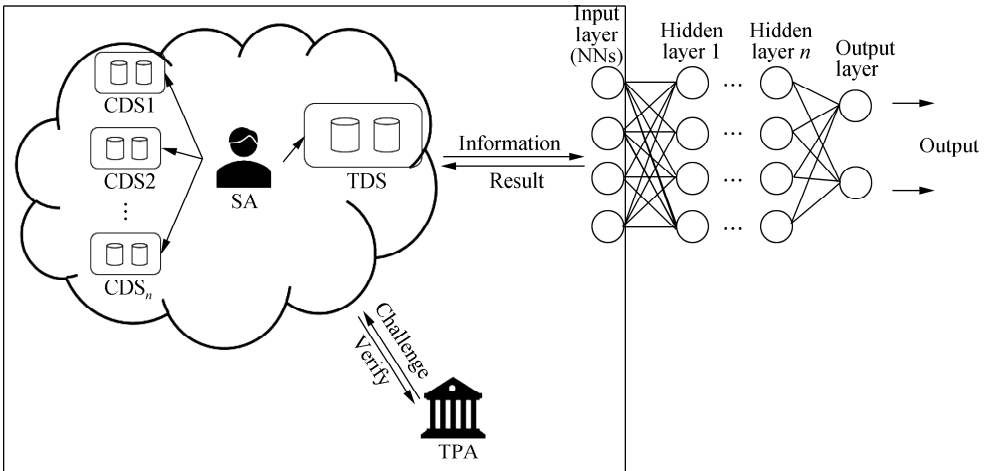


Fig. 1 System model for Our PAD-DR scheme

$CDS_i$  are cloud spaces for storing regenerating backup data, where the regenerating backup data are distributed on  $n$  multiple  $CDS_i$ . TDS stores the original training data with tags and sends them to the NNs. NNs receive the training data and feed them to the neural network. Besides, NNs are responsible for detecting the poisoning attacks in real-time with the data tags.

## 2.2 Threat model

Our PAD-DR scheme defines the threat model in terms of the SA, TPA  $CDS_i$ , TDS, NNs, and network attackers (NAs).

The SA is credible. The SA encodes the training data, generates tags for the original data, and collects regenerating data blocks to help recover the poisoned data.

The TPA is semitrusted. The TPA may run the algorithm and protocol in the system. However, the TPA is curious about the contents of training data and tries to obtain the contents of training data in the verification process.

The TDS is semitrusted. The TDS may faithfully run the algorithm and protocol in the system. However, to maintain the reputation, they may conceal the truth when poisoning samples attack the training data. At that time, TDS may attempt to forge proofs to cheat TPA for passing the auditing process.  $CDS_i$  are credible and may store encoded regenerating data for backup. NNs are credible and may run the algorithm and protocol in the system and transmit the training data to the nodes of the next layer.

NAs are malicious. NAs attempt to launch poisoning sample attacks. Furthermore, to pass the audit by TPA and NNs, NAs may attempt to launch tempering and forgery attacks by forging data and signature proofs.

## 3 Construction of PAD-DR Scheme

We proposed the detailed construction of our PAD-DR scheme. The scheme includes three phases: the setup, detection and recovery phases. Details of each phase are as follows.

### 3.1 Setup phase

The setup phase includes setup, encoding and SigGen algorithms. Among them, the setup algorithm generates system parameters, the encoding algorithm applies CRS code technology to encode the original training data for backup, and the SigGen algorithm generates tags for the original training data. Three algorithms are described in detail as follows.

#### 3.1.1 Setup algorithm

According to the provable data possession (PDP) sampling audit algorithm, we designed the setup algorithm as follows. Let  $G_1, G_2$  be two multiplicative cyclic groups of prime order  $p$  and  $e: G_1 \times G_1 \rightarrow G_2$  be a bilinear map. Let  $g$  be the generator of  $G_1$ .  $H(\cdot), H_1(\cdot)$  are secure

map-to-point hash functions:  $\{0, 1\}^* \rightarrow Z_p^*, \{0, 1\}^* \rightarrow G_1$ . At first, SA chooses a random number  $\alpha \in Z_p$  and a random number  $u \leftarrow G_1$  and computes  $v \leftarrow g^\alpha$ . Subsequently, the SA publishes the system parameters  $\pi = \{G_1, G_2, H, H_1, v, g, u\}$  publicly and saves the private key  $sk = \alpha$  secretly.

#### 3.1.2 Encoding algorithm

Suppose  $m, k, w \in \mathbb{Z}^+$ , a Galois field  $GF(2^w)$ , for data file  $F = \{m_1, m_2, \dots, m_k\}$ , SA utilizes CRS<sup>[22]</sup> technology to generate  $n = m + k$  encoded data blocks by constructing a code matrix  $C = \Psi M$ , where the encoding matrix  $\Psi$  is  $n \times k$ , and the message matrix  $M$  is  $k \times 1$ .

First, the encoding matrix  $\Psi = \begin{bmatrix} E \\ G \end{bmatrix}$  is constructed as follows. The matrix  $E$  is a  $k$ -order identity matrix with main diagonal elements of 1 and other elements of 0, and the matrix  $G$  is a  $m \times k$  Cauchy matrix over a Galois Field  $GF(2^w)$ . Let  $X = \{x_1, x_2, \dots, x_k\}$  and  $Y = \{y_1, y_2, \dots, y_m\}$  be defined such that each  $x_i$  and  $y_i$  is a distinct element of  $GF(2^w)$ , and  $X \cap Y = \emptyset$ . Then SA constructs  $G$  with  $X$  and  $Y$  as follows:

$$G = \begin{bmatrix} \frac{1}{x_1 + y_1} & \frac{1}{x_2 + y_1} & \dots & \frac{1}{x_k + y_1} \\ \frac{1}{x_1 + y_2} & \frac{1}{x_2 + y_2} & \dots & \frac{1}{x_k + y_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{x_1 + y_m} & \frac{1}{x_2 + y_m} & \dots & \frac{1}{x_k + y_m} \end{bmatrix}_{m \times k}$$

Then, the encoding matrix  $\Psi$  is designed as follows:

$$\Psi = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ \frac{1}{x_1 + y_1} & \frac{1}{x_2 + y_1} & \dots & \frac{1}{x_k + y_1} \\ \frac{1}{x_1 + y_2} & \frac{1}{x_2 + y_2} & \dots & \frac{1}{x_k + y_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{x_1 + y_m} & \frac{1}{x_2 + y_m} & \dots & \frac{1}{x_k + y_m} \end{bmatrix}_{n \times k}$$

Hence, the  $i$ -th row of  $\Psi$  is defined as the encoding vector  $\Psi_i (i \in \{1, 2, \dots, n\})$ .

Next, we defined the message matrix  $M = [m_1 \ m_2 \ \dots \ m_k]^T$  according to the data file  $F = \{m_1, m_2, \dots, m_k\}$ . The  $M = [m_1 \ m_2 \ \dots \ m_k]^T$ , which is a  $k \times 1$  matrix.

Finally, SA constructs the code matrix  $C$  by calculating  $C = \Psi M = [c_1 \ c_2 \ \dots \ c_n]^T$ . The  $i$ -th row of code matrix  $C$  is defined as  $c_i (i \in \{1, 2, \dots, n\})$ , which is named as regenerating code. SA stores  $c_i (i \in \{1, 2, \dots, n\})$  with  $\Psi_i$  on each  $CDS_i (i \in \{1, 2, \dots, n\})$ , respectively.

For example, let  $k=6$ ,  $m=4$ , and  $n=10$ , the Cauchy matrix  $G$  is with dimensions  $4 \times 6$  on  $\text{GF}(2^8)$ . Let  $X = \{0, 1, 2, 3, 4, 5\}$ ,  $Y = \{6, 7, 8, 9\}$ , the encoding process is shown as follows:

$$C = \text{Encode}(M) = \Psi M =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0+6 & 1+6 & 2+6 & 3+6 & 4+6 & 5+6 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0+7 & 1+7 & 2+7 & 3+7 & 4+7 & 5+7 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0+8 & 1+8 & 2+8 & 3+8 & 4+8 & 5+8 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0+9 & 1+9 & 2+9 & 3+9 & 4+9 & 5+9 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 122 & 186 & 71 & 167 & 142 & 244 \\ 186 & 122 & 167 & 71 & 244 & 142 \\ 173 & 157 & 221 & 152 & 61 & 170 \\ 157 & 173 & 152 & 221 & 170 & 61 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \end{bmatrix} =$$

$$\begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ 122m_1 + 186m_2 + 71m_3 + 167m_4 + 142m_5 + 244m_6 \\ 186m_1 + 122m_2 + 167m_3 + 71m_4 + 244m_5 + 142m_6 \\ 173m_1 + 157m_2 + 221m_3 + 152m_4 + 61m_5 + 170m_6 \\ 157m_1 + 173m_2 + 152m_3 + 221m_4 + 170m_5 + 61m_6 \end{bmatrix}$$

### 3.1.3 SigGen

For the data file  $F = (m_1, m_2, m_3, \dots, m_k)$ , we designed SigGen, an algorithm according to a PDP sampling audit algorithm, to generate the following signatures. First, SA utilizes the hash function  $H(\cdot)$  to calculate the hash value  $H(m_i)$  according to each data block  $m_i$ . Subsequently, SA uses the private key  $\text{sk} = \alpha$  to generate signatures as  $\sigma_i \leftarrow (H_1(W_i)u_{(m_i)}^\alpha) \in G_1$ , where  $W_i = f \parallel i$  ( $i = 1, 2, \dots, k$ ), and SA chooses the  $f$  at random from  $Z_p^*$ . Meanwhile, SA applies the RSA signature algorithm to compute  $S_{\text{ssk}}(f \parallel k)$ , where  $\text{ssk}$  is the private key in the RSA signature algorithm. Hence, the SA generates the set of signatures  $\Phi = \{\sigma_i \mid 1 \leq i \leq k\}$  and computes  $S_F = f \parallel k \parallel S_{\text{ssk}}(f$

$\parallel k)$ . Finally, SA stores  $\{F, \Phi, S_F\}$  on TDS.

## 3.2 Detection phase

The detection phase consists of TPA detection and node detection algorithms. With our proposed sampling audit algorithm, the TPA detection algorithm can detect poisoning attacks on the training data stored in TDS. In real-time, a node detection algorithm can detect poisoning attacks on the training data at the input layer for NNs.

### 3.2.1 TPA detection

First, the TPA verifies the integrity of the training data with a signature  $\text{SSig}_F$ . The TPA checks whether  $S_{\text{ssk}}(f \parallel k)$  is a valid signature with SA's public key by the RSA signature algorithm. The TPA aborts the message if the verification fails. Otherwise, according to the PDP sampling audit algorithm, to detect a poisoning attack on the training data stored in TDS, the TPA generates the challenge message  $\zeta$  as follows. The TPA randomly selects a subset  $I' \subset I$ , where  $I = \{1, 2, \dots, k\}$  is a positive integer set. For each  $i \in I'$ , the TPA chooses a random value  $v_i$  and generates  $\zeta = \{(i, v_i) \mid i \in I'\}$ . Finally, the TPA is sent  $\zeta = \{(i, v_i) \mid i \in I'\}$  to the TDS. Upon receiving a challenge  $\zeta = \{(i, v_i) \mid i \in I'\}$  from the TPA, the TDS generates a response proof as  $\mu = \sum_{i \in I'} v_i H(m_i)$ , and  $\sigma = \prod_{i \in I'} \sigma_i^{v_i} \in G_1$ . Finally, the TDS is sent  $P = \{\mu, \sigma\}$  to the TPA as the response proof.

Having received the response proof from the TDS, the TPA verifies the proof as follows. TPA checks the verification equation as

$$e(\sigma, g) \stackrel{?}{=} e(u^\mu \prod_{i \in I'} H_1(W_i)^{v_i}, v) \quad (1)$$

If Eq. (1) holds, the training data stored on TDS are securely protected. Otherwise, the training data stored on TDS should be changed or corrupted by the poisoning attacks. When the training data is detected to be poisoned, the TPA immediately makes an alarm for the poisoning attack and sends feedback  $\{\text{error}\}$  to the SA. Then, the SA performs the recovery operation to recover the poisoned data on TDS.

### 3.2.2 Node detection

Before the system model training starts, the TDS selects a random value  $v'_i (i \in I)$  in  $Z_p$ , and utilizes  $H(m_i)$  and  $\{\sigma_i \mid i \in I\}$  generated in the SigGen phase to construct a proof for all data blocks as  $\mu' = \sum_{i \in I} v'_i H(m_i)$  and  $\sigma' = \prod_{i \in I} \sigma_i^{v'_i} \in G_1$ . Then, the TDS applies a private key  $\text{tsk}$  to generate  $\text{Sig}_{\text{tsk}}(F \parallel \mu' \parallel \sigma')$  using the RSA signature algorithm. Subsequently, the TDS sends training data with signatures  $\{F, \mu', \sigma', S_{\text{tsk}}(F \parallel \mu' \parallel \sigma'), v'_i \mid i \in I\}$  to the NNs.

After receiving training data with signatures from TDS, NNs run a node detection algorithm to detect poisoning

attacks as follows. First, NNs verify the correctness of the signature  $S_{\text{isk}}(F \parallel \mu' \parallel \sigma')$  by TDS's public key according to the RSA signature algorithm. If the verification fails, the NNs abort the message and send  $\{\text{error}\}$  to SA. Otherwise, we have to design the real-time data detection method for the NNs to check the verification equation as follows:

$$e(\sigma', g) \stackrel{?}{=} e(u' \prod_{i \in I} H_1(W_i)^{v'_i}, v) \quad (2)$$

If Eq. (2) holds, the training data sent by TDS are complete and correct. Otherwise, the poisoning attacks could change or corrupt the training data. NNs may send error feedback to inform SA that the training data could be attacked and request SA to recover the training data.

### 3.3 Recovery phase

In the recovery phase, there is a data recovery algorithm. Once SA receives error feedback from TPA or NNs, implying that the training data has been poisoned and attacked, SA may execute the data recovery algorithm to recover the original training data.

With the data recovery algorithm, the original data file  $F = \{m_1, m_2, \dots, m_k\}$  can be recovered by collecting any  $k$  nodes of  $\text{CDS}_i$  for its  $c_i$  and  $\Psi_i$ . Thus, the original data file  $F$  can be recovered through linear operations on the entries of any  $k$  rows of the code matrix  $C$ .

First, SA selects  $k$  elements from a subset  $N' \subset N$ , where the positive integer set is  $N = \{1, 2, \dots, n\}$ , and collects the regenerating code  $c_i$  from  $\text{CDS}_i (i \in N')$ . Thus, all regenerating code received by the SA is  $[c_{i_1} \ c_{i_2} \ \dots \ c_{i_k}]^T$ . Then, according to the subset  $N' = \{i_1, i_2, \dots, i_k\}$ , SA chooses  $k$  encoding vectors  $\Psi_i (i \in N')$  to construct a new matrix  $\Psi_{\text{sub}} = [\Psi_{i_1} \ \Psi_{i_2} \ \dots \ \Psi_{i_k}]^T$ .

Second, according to the character of  $\Psi$  in Section 3.1.2, any  $k$ -order sub-matrix of  $\Psi$  is invertible. Thus,  $\Psi_{\text{sub}}$  is a reversible matrix, and the SA can generate the inverse matrix  $\Psi_{\text{sub}}^{-1}$ .

Finally, the SA can resolve  $[m_1 \ m_2 \ \dots \ m_k]^T = \Psi_{\text{sub}}^{-1} [c_{i_1} \ c_{i_2} \ \dots \ c_{i_k}]^T$ . Hence, the original training data file  $F$  can be recovered.

Next, based on the definition in the preliminaries, we provided a theorem indicating that the entire process of the PAD-DR scheme is correct.

**Theorem 1** Our PAD-DR scheme can correctly run to detect the poisoning attack and recover the original training data accurately.

## 4 Security Analysis

In this section, we formally proved that our PAD-DR scheme can resist tampering and forgery attacks. Also, we proved that the poisoned data can be correctly recovered.

### 4.1 Related theorems

Some theorems formally showed that our PAD-DR scheme can resist tampering and forgery attacks. Furthermore, we showed that the poisoned data can be correctly recovered.

**Theorem 2** In our PAD-DR scheme, it is computationally infeasible for the TDS and NAs to forge proof for passing the TPA's auditing. Suppose there is a  $(t, \vartheta)$ -algorithm to forge a proof. Then, if a  $(t', \vartheta')$  adversary can solve the Co-CDH problem with  $t' \leq t + q_{cG_1}$  and  $\vartheta' = \vartheta/e(1 + q_s)$ , where  $q_{cG_1}$  denotes one exponentiation time in  $G_1$ , and  $q_s$  denotes the number of requests.

**Theorem 3** In our PAD-DR scheme, NNs can detect poisoned samples in real-time training data received from TDS.

**Theorem 4** In our PAD-DR scheme, the original data file  $F$  can be recovered by collecting  $c_i$  and  $\Psi_i$  from any  $k$  nodes of  $\text{CDS}_i$ .

### 4.2 Security goals comparison

In this section, the security goals of our PAD-DR scheme are compared with that of DUTI<sup>[14]</sup>, DeepKNN<sup>[15]</sup>, Server<sup>[18]</sup> and De-Pois<sup>[19]</sup>. The comparison includes specific poisoning attack detection, arbitrary poisoning attack detection, and poisoned data recovery. In Tab. 1, “√” depicts that the security problem has been solved, “×” indicates that the problem has not been solved.

**Tab. 1** Security goals comparison

Security goal	Ref. [16]	Ref. [17]	Ref. [18]	Ref. [19]	PAD-DR
Specific detection	√	√	√	√	√
Arbitrary detection	×	×	×	×	√
Data recovery	×	×	×	×	√

According to Tab. 1, our PAD-DR scheme can realize all the security goals mentioned above, while other schemes cannot.

## 5 Experiment and Performance Analysis

This section begins by describing the experimental setup and the accuracy under different attacks. Then, we analyzed the communication overhead and computation costs in detecting poisoning attacks of training data and evaluated the performance of our PAD-DR scheme.

### 5.1 Experiment setup

#### 5.1.1 Datasets

The training data used in our scheme are from MNIST, CIFAR-10, and house pricing. MNIST is a database of handwritten digits containing 60 000 training sample sets and 10 000 test sample sets stored in binary form. The width and height of each sample image are  $28 \times 28$ . The

CIFAR-10 database contains  $32 \times 32$  color images in 10 different classes, with 50 000 training and 10 000 testing images. The 10 different classes include cars, dogs, frogs, airplanes, cars, ships, horses, birds, and trucks. The house pricing dataset utilizes predictor variables such as the number of bedrooms and lot square footage. It contains 1 460 houses and 81 features. In the experiment, we randomly split this dataset into training and testing datasets with 70% and 30% of the data, respectively.

### 5.1.2 Experimental environment and parameter setting

We evaluated the process of detecting poisoning samples on NVIDIA 2080Ti GPU and applied Ali Cloud to store training data. The operating system is Windows 10. The RAM size is 8 GB. Matrix multiplication algorithms are implemented with the open-source library Jerasure Version (1.2). Auditing algorithms are implemented on C with a pairing-based cryptography library. The curve utilized in the experiment is an MNT curve. We set the length of  $G_1$  to 175. In our experiment, we adopted a common neural network that applies three full connection layers with ReLU activation and one full connection layer with sigmoid activation, and we applied the cross-entropy loss function to calculate its loss.

### 5.1.3 Generation of poisoning training data

To verify the performance of our PAD-DR scheme, we randomly extracted 10%, 15%, 20%, 25%, and 30% of the training data to generate poisoning samples. To compare the detection rate of poisoned samples for our PAD-DR scheme, TCL<sup>[12]</sup>, LF<sup>[8]</sup>, and R<sup>[13]</sup> attacks are used to generate poisoning data. For MNIST, the number of iterations is 400, the learning rate is set to 0.1, and the initial value is set to 0.01. For CIFAR-10, the number of iterations is 6 000, the learning rate is 0.01, and the initial value is 0.01.

## 5.2 Detection rate for poisoning attacks

In this section, we evaluated the detection rate of poisoned samples for our PAD-DR scheme compared with that of TRIM<sup>[13]</sup>, DUTI<sup>[14]</sup>, Deep-kNN<sup>[15]</sup>, Server<sup>[18]</sup>, and De-Pois<sup>[19]</sup>. The results are presented in Figs. 2, 3, and 4, respectively.

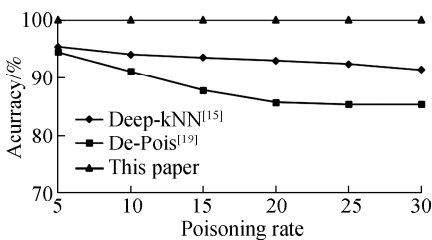


Fig. 2 Detection rate under TCL attacks

As shown in Fig. 2, for TCL attacks, the detection rate of our PAD-DR scheme is always 100%, which is higher

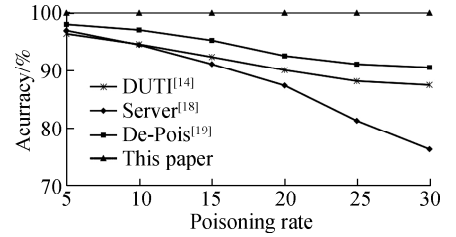


Fig. 3 Detection rate under LF attacks

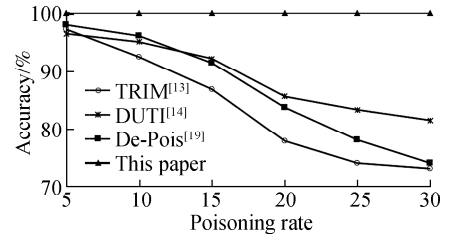


Fig. 4 Detection rate under R attacks

than that of Refs. [15, 19]. There are two reasons for the relatively low detection rate of Ref. [15]. One is that the authors only compared the samples with the surrounding  $k$  samples, which was not sufficient or accurate. The other reason is that the authors should set an artificial setting threshold to distinguish poisoned samples from clean samples. The main reason for the relatively low detection rate of Ref. [19] is that the authors adopted conditional GAN technology to expand a small part of a trusted dataset as the whole training data, which could not fully reflect the features of the real training data. In our PAD-DR scheme, we could only detect any changed or poisoned samples in the training data by verifying Eqs. (1) and (2). Hence, the detection rate of our PAD-DR scheme is always 100%.

As illustrated in Fig. 3, for TF attacks, the detection rate of our PAD-DR scheme is always 100%, which is higher than that of Refs. [14, 18–19]. In Ref. [14], the main reason for the relatively low detection rate is that the authors required a small piece of completely reliable data, which could not be ensured in the system to detect the TF attack. In Ref. [18], the detection rate decreased rapidly with the increase of the poisoning rate. The main reason to detect the TF attack is that the authors should set an artificial setting threshold, which could not accurately distinguish poisoned samples from clean samples. In Ref. [19], the main reason for the relatively low detection rate is that the authors should train the mimic model, which was ineffective in obtaining accurate prediction results.

As depicted in Fig. 4, for R attacks, the detection rate of our PAD-DR scheme is always 100%, which is higher than that of Refs. [13–14, 19]. With the increase of the poisoning rate, the detection rate of Refs. [13–14, 19] decreased rapidly. In Ref. [13], to detect R attacks, the

authors ignored the influence of poisoned samples in the lowest residual set with iteration, which may cause the failure of R-attack detection. In Ref. [14], the authors required a small piece of completely reliable data to detect an R attack, which could not be ensured in the system. In Ref. [19], the authors should train the mimic model to detect R attacks, which was not effective enough to obtain accurate prediction results. In our PAD-DR scheme, we could only detect any changed or poisoned samples in the training data using verifying Eqs. (1) and (2). Hence, the detection rate of our PAD-DR scheme for R attacks is always 100%.

### 5.3 Computation cost

We evaluated the computation cost for our PAD-DR scheme, where  $L$  denotes the length of each encoded data block,  $E_G$  denotes one exponentiation in  $G_1$  and  $G_2$ ,  $E_Z$  denotes one exponentiation in  $Z_p$ ,  $M_G$  indicates a multiplication operation on groups  $G_1$  and  $G_2$ ,  $M_Z$  indicates a multiplication operation on the number field  $Z_p$ ,  $M_F$  represents a matrix multiplication in a finite field  $F(2^n)$ ,  $M_I$  represents the cost of computing the inverse of the matrix  $M$ ,  $A_Z$  indicates an addition operation on the number field  $Z_p$ ,  $P_e$  denotes the computation cost of one pairing operation  $e$ , and  $H$  denotes the computation cost of an operation of calculating the hash value for a number. Furthermore,  $|I|$  represents the number of challenged data blocks. All the statistical results are the averages of 20 trials.

Tab. 2 depicts the computation cost of our PAD-DR scheme in the setup, detection, and recovery phases.

**Tab. 2** Computation cost of our PAD-DR scheme

Phase	Setup phase	Detection phase	Recovery phase
Computation cost	$M_F + \frac{\text{sizeof}(F)}{L} \cdot (H + E_G + M_Z)$	$2 I (M_Z + H + E_G) + 2( I  - 1)(A_Z + M_G) + 2P_e$	$M_F + M_I$

The SA encodes the original training data in the setup phase and generates signatures for every training data block. The computation cost is  $M_F$  and  $\frac{\text{sizeof}(F)}{L}(H + E_G + M_Z)$ , respectively. The total time in the setup phase is  $M_F + \frac{\text{sizeof}(F)}{L}(H + E_G + M_Z)$ .

In the detection phase, the computation cost refers to two parts: TPA detection and node detection. The computation cost of TPA detection is  $|I|(M_Z + H + E_G) + (|I| - 1)(A_Z + M_G) + P_e$ , where  $|I|(M_Z + H) + (|I| - 1)A_Z$  and  $|I|E_G + (|I| - 1)M_G$  are the computation cost of TDS to generate a linear combination of data blocks and an aggregated tag, respectively, where  $P_e$  is the computation cost of TPA to verify the proof. The computation cost of node detection is  $|I|(M_Z + H +$

$E_G) + (|I| - 1)(A_Z + M_G) + P_e$ , where  $|I|(M_Z + H) + (|I| - 1)A_Z$  and  $|I|E_G + (|I| - 1)M_G$  are the computation cost of TDS to generate a linear combination of data blocks and an aggregated tag, respectively,  $P_e$  is the computation cost of NNs to verify the proof. Hence, the computation cost in the detection phase is  $2|I|(M_Z + H + E_G) + 2(|I| - 1)(A_Z + M_G) + 2P_e$ .

In the recovery phase, the cost of original training data recovery is  $M_I + M_F$ , where  $M_I$  is the cost of SA computing the inverse matrix  $\Psi_{\text{sub}}^{-1}$ , and  $M_F$  is the cost of calculating matrix multiplication  $\Psi_{\text{sub}}^{-1} [c_1 \ c_2 \ \cdots \ c_k]^T$ .

### 5.4 Communication overhead

We assessed the communication overhead in the setup, detection, and recovery phases. In this section,  $|Z_p|$  represents the size of  $Z_p$ ,  $|G|$  represents the size of group  $G$ , and  $|GF|$  represents the size of  $GF(2^n)$ . Tab. 3 presents the communication overhead for the three phases.

**Tab. 3** Communication overhead for our PAD-DR scheme

Phase	Setup phase	Detection phase	Recovery phase
Communication overhead	$\text{sizeof}(F) + k G  + n GF $	$\text{sizeof}(F) + 2 G  + (k + 2 + 2 I ) \cdot  Z_p $	$2k GF $

In the setup phase, communication overhead mainly includes two parts. The one is SA uploads data and tags  $\{F, \Phi\}$  to TDS. The other is SA sends the encoded data  $\{c_i\}_{i=1,2,\dots,n}$  to  $\text{CDS}_i$ . Hence, the communication overhead in the setup phase is  $\text{sizeof}(F) + k|G| + n|GF|$ .

In the detection phase, the communication overhead mainly refers to two parts: TPA detection and node detection. The communication overhead for TPA detection includes two parts. One is that TPA sends a challenge  $\text{chal} = \{(i, v_i)\}$  to TDS. The other is that TDS replies as proof  $P = \{\mu, \sigma\}$ . Hence, the communication overhead for TPA detection is  $2|I||Z_p|$  and  $|G| + |Z_p|$ . The communication overhead for node detection is  $\text{sizeof}(F) + |G| + (k + 1)|Z_p|$  which arises from TDS sending  $\{F, \mu', \sigma', S_{\text{ssk}}(F \parallel \mu' \parallel \sigma'), v_i\}_{i \in I}$  to NNs. Thus, the total communication overhead in the detection phase is  $\text{sizeof}(F) + 2|G| + (k + 2 + 2|I|)|Z_p|$ .

In the recovery phase, the communication overhead is decided by the data size of  $\{c_i, \Psi_i\}$ , which arises from the  $k$  nodes of  $\text{CDS}_i$ . Hence, the communication overhead of our PAD-DR scheme is  $2k|GF|$ .

## 6 Conclusions

1) An algorithm combining data sampling audit and real-time data detection is designed, and experiments show that the algorithm can accurately detect toxic data con-

tained in the data.

2) A data recovery algorithm based on CRS encoding is designed, and experiments show that it can efficiently restore poisoned data to clean data.

3) The current algorithm cannot guarantee the security of parameters during transmission. We will conduct further research on improving the integrity and confidentiality of parameters in the future.

## References

- [1] LeCun Y, Bengio Y, Hinton G. Deep learning[J]. *Nature*, 2015, **521**(7553): 436 – 444. DOI: 10.1038/nature14539.
- [2] Ackerman E. How Drive.ai is mastering autonomous driving with deep learning [EB/OL]. (2017-12) [2022-10-16]. <https://spectrum.ieee.org/cars-that-think/transportation/self-driving/how-driveai-is-mastering-autonomous-driving-with-deep-learning>.
- [3] Class Central. Deep learning for self-driving cars [EB/OL]. (2017) [2022-11-20]. <https://www.class-central.com/mooc/8132/6-s094-deep-learning-for-self-driving-cars>.
- [4] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. *Nature*, 2015, **518** (7540): 529 – 533. DOI: 10.1038/nature14236.
- [5] Giusti A, Guzzi J, Ciresan D C, et al. A machine learning approach to visual perception of forest trails for mobile robots[J]. *IEEE Robotics and Automation Letters*, 2016, **1** (2): 661 – 667. DOI: 10.1109/lra.2015.2509024.
- [6] He K M, Zhang X Y, Ren S Q, et al. Deep residual learning for image recognition[C]//2016 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA, 2016: 770 – 778. DOI: 10.1109/CVPR.2016.90.
- [7] Chakraborty K, Bhattacharyya S, Bag R, et al. Comparative sentiment analysis on a set of movie reviews using deep learning approach[C]//*International Conference on Advanced Machine Learning Technologies and Applications*. Cham, Switzerland: Springer, 2018: 311 – 318. DOI: 10.1007/978-3-319-74690-6\_31.
- [8] Biggio B, Nelson B, Laskov P. Support vector machines under adversarial label noise[C]//*Proceedings of the Asian Conference on Machine Learning*. Taoyuan, China, 2011: 97 – 112.
- [9] Muñoz-González L, Biggio B, Demontis A, et al. Towards poisoning of deep learning algorithms with back-gradient optimization[C]//*Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. Dallas, TX, USA, 2017: 27 – 38. DOI: 10.1145/3128572.3140451.
- [10] Liu H Q, Li D X, Li Y C. Poisonous label attack: Black-box data poisoning attack with enhanced conditional DCGAN[J]. *Neural Processing Letters*, 2021, **53** (6): 4117 – 4142. DOI: 10.1007/s11063-021-10584-w.
- [11] Alberti M, Pondenkandath V, Wursch M, et al. Are you tampering with my data? [C]// *The European Conference on Computer Vision*. Munich, Germany, 2018: 296 – 312.
- [12] Shafahi A, Huang W R, Najibi M, et al. Poison frogs! Targeted clean-label poisoning attacks on neural networks [C]// *Neural Information Processing Systems 31 (NIPS)*. Montréal, Canada, 2018: 6106 – 6116.
- [13] Jagielski M, Oprea A, Biggio B, et al. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning[C]//2018 *IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA, 2018: 19 – 35. DOI: 10.1109/SP.2018.00057.
- [14] Zhang X Z, Zhu X J, Wright S. Training set debugging using trusted items[C]//*Proceedings of the AAAI Conference on Artificial Intelligence*. New Orleans, LA, USA, 2018: 1 – 8. DOI: 10.1609/aaai.v32i1.11610.
- [15] Peri N, Gupta N, Huang W R, et al. Deep k-NN defense against clean-label data poisoning attacks [EB/OL]. (2019) [2022-12-22]. <http://arxiv.org/abs/1909.13374>.
- [16] Shen S Q, Tople S, Saxena P. Auror: Defending against poisoning attacks in collaborative deep learning systems [C]//*Proceedings of the 32nd Annual Conference on Computer Security Applications*. Los Angeles, CA, USA, 2016: 508 – 519. DOI: 10.1145/2991079.2991125.
- [17] Liu K, Dolan-Gavitt B, Garg S. Fine-pruning: Defending against backdoor attacks on deep neural networks [C]//*International Symposium on Research in Attacks, Intrusions, and Defenses*. Heraklion, Crete, Greece, 2018: 273 – 294. DOI: 10.1007/978-3-030-00470-5\_13.
- [18] Diakonikolas I, Kamath G, Kane D. Sever: A robust meta-algorithm for stochastic optimization[C]//*International Conference on Machine Learning*. Long Beach, California, USA, 2019: 1596 – 1606.
- [19] Chen J, Zhang X X, Zhang R, et al. De-Pois: An attack-agnostic defense against data poisoning attacks[J]. *IEEE Transactions on Information Forensics and Security*, 2021, **16**: 3412 – 3425. DOI: 10.1109/TIFS.2021.3080522.
- [20] Krawczyk H. HMQV: A high-performance secure Diffie-Hellman protocol[C]// *25th Annual International Cryptology Conference*. Santa Barbara, CA, USA, 2005: 1 – 62.
- [21] Bao F, Deng R H, Zhu H F. Variations of Diffie-Hellman problem[C]// *Proceeding of the International Conference on Information and Communications Security*. Hohhot, China, 2003: 301 – 312.
- [22] Roth R M, Lempel A. On MDS codes via cauchy matrices[J]. *IEEE Transactions on Information Theory*, 1989, **35**(6): 1314 – 1319. DOI: 10.1109/18.45291.



# 神经网络中基于数据完整性抽样审计算法的中毒攻击检测方案

赵宁宁 蒋 睿

(东南大学网络空间安全学院, 南京 210096)

**摘要:**为了解决大多数现有的检测和防御方法只能检测已知的中毒攻击,而不能防御其他类型中毒攻击的问题,提出了一种带数据恢复的中毒攻击检测方案(PAD-DR),以有效地检测中毒攻击并恢复神经网络中毒的数据. 首先,该方案可以检测各种中毒攻击. 将数据采样检测算法与神经网络中输入层节点的实时数据检测方法相结合,使系统能够确保训练数据的完整性和可用性,避免被更改或损坏. 其次,该方案可以恢复中毒攻击中损坏或中毒的训练数据. 将柯西-里德-所罗门(CRS)码技术应用于编码训练数据,并将其单独存储,一旦检测到中毒攻击并且导致数据受损,系统就可以从所有  $n$  个存储中的任何  $k$  个节点获取数据,以恢复原始的训练数据. 最后,给出了 PAD-DR 方案的形式化证明,证明了该方案能够抵御中毒攻击、抵御临时伪造和篡改攻击,以及准确恢复数据.

**关键词:**投毒攻击;神经网络;深度学习;数据完整性审计

**中图分类号:**TP339