Peer-to-Peer Networking and Applications (2022) 15:2457–2473
https://doi.org/10.1007/s12083-022-01377-6

MP-BADNet+: Secure and effective backdoor attack detection and mitigation protocols among multi-participants in private DNNs

Congcong Chen[1] · Lifei Wei[2] · Lei Zhang[1] · Ya Peng[1] · Jianting Ning[3,4]

Received: 26 January 2022 / Accepted: 28 August 2022 / Published online: 5 September 2022

Abstract
Deep neural networks (DNNs) significantly facilitate the performance and efficiency of the Internet of Things (IoT). However, DNNs are vulnerable to backdoor attacks where the adversary can inject malicious data during the DNN model training. Such attacks are always activated when the input is stamped with a pre-specified trigger, resulting in a pre-setting prediction of the DNN model. It is necessary to detect the backdoors whether the DNN model has been injected before implementation. Since the data come from the various data holders during the model training, it is also essential to preserve the privacy of both input data and model. In this paper, we propose a framework MP-BADNet+ including backdoor attack detection and mitigation protocols among multi-participants in private deep neural networks. Based on the secure multi-party computation technique, MP-BADNet+ not only preserves the privacy of the training data and model parameters but also enables backdoor attacks detection and mitigation in privacy-preserving DNNs. Furthermore, we give the security analysis and formal security proof following the real world-ideal world simulation paradigm. Last but not least, experimental results demonstrate that our approach is effective in detecting and mitigating backdoor attacks in privacy-preserving DNNs.

Keywords Backdoor attack · Detection and mitigation · Privacy-preserving · Secure multi-party computation · Deep neural networks

1 Introduction

Deep neural networks (DNN) have been widely used in the Internet of Things (IoT) in recent years to facilitate the efficiency and performance of IoT devices, which has profoundly changed our lives. With the increasing demand for model accuracy, enormous amounts of training data are usually required for the training of DNN models [1]. Currently, the data for training models are always sourced from diverse data holders. However, due to restrictions on sensitive data in laws and regulations such as the European Union General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA), it is not allowed to share sensitive data directly, such as in the form of plaintext for DNN model training. Thus, the privacy and security issues in DNNs have attracted a great deal of attention as a real emergent issue [2].

The feasible solution to the privacy issue is that one would like to train an accurate model without revealing sensitive data of the data holder. One reliable alternative is secure multi-party computation (MPC) technique [3, 4]. MPC allows multiple participants who mutually distrust each other to cooperate in computing arbitrary functions on their private inputs without revealing anything except the output of the function. Many efficient MPC frameworks as a privacy-preserving machine learning have been proposed in the past five years such as SecureNN [5], ASTRA [6], BLAZE [7], FALCON [8].

✉ Lifei Wei
weilifei@foxmail.com

✉ Lei Zhang
lzhang@shou.edu.cn

[1] College of Information Technology, Shanghai Ocean University, Shanghai 201306, China

[2] College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China

[3] Fujian Provincial Key Laboratory of Network Security and Cryptology, College of Computer and Cyber Security, Fujian Normal University, Fuzhou 350117, China

[4] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

However, the security issues are quite hard to detect in DNNs since the weights and biases of the DNN model are not easily understood by humans, nor is it easy to exhaustively interpret all the inputs available for testing. It is always considered a black box [9], especially in the ciphertext domain. The prior work has shown that backdoors may be embedded into DNN models by malicious attackers [10, 11]. Backdoor attacks usually train DNN models using the images with triggers (also known as adversarial input or infected input) as the training dataset. When a backdoor is successfully embedded in a DNN model, it may bring designated results if activated by the infected input or normal output for other clean inputs. As a result, the backdoor in the DNN model is difficult to detect.

Consider the following IoT scenario. For an intelligent home containing a facial recognition system, a malicious attacker may inject faces with a certain special trigger into the trained DNN model so that the attacker can use the trigger to pretend to be a legal person. That is, the image of any person with the trigger is always recognized as the disguised target. For the infected DNN model, it is difficult for the owner of an intelligent home to discover that the facial recognition system may be attacked. In addition, for a clean image, the facial recognition system will recognize it normally [10]. Similar attacks can be applied to other DNN models, such as the automatic driving model [11].

We investigate related work and find that most prior work focuses on privacy-preserving machine learning or poisoning attacks. The former focuses on the privacy issues in machine learning, such as linear regression, logistic regression, and neural networks. The latter is always trying to reduce the inference accuracy on clean data [12]. For the reasons discussed above, this paper extends our earlier work (a conference version in [13]). The contributions of this paper are summarized as follows:

- **Enabling backdoor attacks detection and identification in the privacy-preserving DNN model.** We present a general solution to detect and identify the hidden backdoor attacks in a pre-trained privacy-preserving DNN model. The major computing servers and the secondary server in our system jointly perform the computation to find out whether the model has suffered backdoor attacks and recover the specific trigger deeply embedded inside the model through a reverse engineering algorithm and an outlier detection algorithm. Both algorithms run in the ciphertext domain, which preserves the privacy of the training data and the parameters of DNN models.
- **Enabling backdoor attacks mitigation in the privacy-preserving DNN model.** We use an MPC-based neuron pruning to mitigate the backdoor attack in the privacy-preserving DNN model with backdoors without revealing private information. The computing servers and the secondary server jointly execute the backdoor mitigation algorithm and finally achieve high accuracy of the backdoor mitigation approach to those in plaintext.
- **Performing the evaluation of MP-BADNet$^+$.** We evaluate MP-BADNet$^+$ on a deep neural network and train it on the MNIST dataset. For further comparison, We use BadNets [11] to add triggers to nearly 10% of the MNIST dataset and train a privacy-preserving model with backdoors. Our experiment results demonstrate that our scheme can effectively detect, identify, and mitigate backdoor in the privacy-preserving DNN model.

The rest paper is organized as follows. Section 2 introduces the related work about backdoor attacks and privacy-preserving machine learning. Section 3 describes our system architecture, security models, and design goals. Section 4 sees the notations. In Sect. 5, we propose a detail construction and give the security analysis and formal security proof. The performance comparison is given in Sect. 6. Finally, Sect. 7 makes a conclusion.

2 Related work

2.1 Backdoor attacks

Gu et al. [11] introduced a backdoor attack method called BadNets. They attached triggers to traffic signs and showed a high probability of identifying a stop sign as a speed limit sign, which is a tremendous potential threat to the application of autonomous driving. The BadNets proved that the backdoor attack on neural networks is feasible with stronger performance. BadNets injects the backdoor by modifying the clean datasets, that is, modifying part of the clean datasets so that it contains the specified trigger. Chen et al. [14] first focus on the concealment of backdoor attacks requirement, where they argue that poisoned data should be indistinguishable compared to clean data. To satisfy this requirement, they proposed a hybrid strategy by mixing backdoor triggers with clean images. Trojan Attack proposed by Liu et al. [10] shows that a malicious attacker can make a face recognition classifier identify an illegal user as a legitimate one. Trojan Attack can complete the backdoor attack without modifying the clean datasets and without modifying the initial model. It first inverses the neural network to generate a general trigger and then uses the external datasets to retrain the model to inject malicious behavior into the model. As the DNN model is not intuitive, thus it has better concealment.

Besides, some more advanced backdoor attack methods have been proposed in the past few years. Saha et al. [15] proposed a hidden random backdoor trigger injection technique, but it only achieves a better attack when the poisoning rate and trigger size are large. Shokri [16] designed an

adaptive adversarial training algorithm that optimizes the original loss function of the model and makes the poisoned data and clean data maximally indistinguishable. Salem et al. [17] proposed a dropout-based technique for triggerless backdoor attacks for DNNs (i.e., the attacker does not need to modify the inputs). Bagdasaryan et al. [18] proposed a blinded backdoor attack method that does not require access to either the training data or the model to inject backdoors in DNNs successfully.

2.2 Backdoor defenses

Researchers have also come up with some techniques to detect and mitigate backdoor attacks over the last few years. For the method of mitigating the backdoor, Liu et al. [19] proposed neuron trojan that contains three methods to defend against backdoor attacks: input outlier detection, retraining, and input preprocessing. Liu et al. [20] proposed the Fine-Pruning technique, which prunes the neurons to remove the backdoor, and this technique has almost no effect on the clean input. Fine-pruning and neuron trojan only provide mitigation methods, and they both target DNN models that the backdoor has attacked.

In 2019, Wang et al. [9] presented Neural Cleanse, a scheme to detect backdoors in DNNs models by reverse engineering the triggers corresponding to each label and using neuron pruning and unlearning techniques to mitigate backdoor attacks. In the same year, Liu et al. [21] proposed ABS by introducing different levels of stimuli to a neuron and observing how the output activation value changes to determine whether a DNNs model is attacked by a backdoor. Both Wang et al. [9] and Liu et al. [21] provide novel ideas for detecting and mitigating backdoor attacks, which can effectively detect and mitigate certain backdoor attacks (such as BadNets and Trojan Attack). However, they are only for detecting and mitigating pixel image backdoor attacks. They are not suitable for detecting infected models that have multiple backdoor triggers or multiple different triggers that lead to misclassification of the same label. Similarly, the backdoor attack defense schemes proposed by Gao et al. [22] and Chen et al. [23] only target pixel image backdoor attacks. Guo et al. [24] proposed a detection technique TABOR for Trojan Attack, which formalized the trojan detection task as a non-convex optimization problem, and formalized the trojan backdoor detection to solve the optimization problem through an objective function.

2.3 Privacy-preserving machine learning

ABY [25] utilizes efficient multiplication protocols, fast conversion techniques, and preprocessed cryptographic operations to construct secure computational schemes for two-party computing environments. SecureML [3] is a privacy-preserving method for training DNNs in two-party computing environments, which presents an efficient truncation protocol that is more efficient compared to the algorithm proposed by ABY [25]. ABY3 [26] presents a novel scheme to support semi-honest and malicious environments for MPC computation, which implements conversion between arithmetic sharing, boolean sharing, and Yao sharing [4] among three-party scenarios. SecureNN [5] improves the efficiency of the protocol and extends two-party privacy-preserving DNN training to three parties based on SecureML. Recently, Wagh et al. [8] proposed FALCON based on SecureNN and ABY3, which supports batch normalization and the training of large-capacity networks like AlexNet and VGG16. The comparison with the prior works is shown in Table 1. Prior works focus on both privacy and security rarely.

3 Overview

3.1 System architecture

Figure 1 illustrates the system framework of MP-BADNet$^+$. The system consists of three entities: the data holder, the major computing servers, and the secondary server.

Data holders The training data comes from different data holders (e.g., individuals, companies, etc.) who are reluctant to share their private data. Data holders share their private data to three computing servers for model training based on replicated secret sharing (RSS) [32] technique.

Computing servers The computing servers, named as P_0, P_1, P_2, possess a large number of computation resources and storage resources. Based on the private inputs from the data holders, we train DNN models in these three non-collusion computing servers using MPC techniques. The three computing servers hold a piece of private data; as a result, they cannot recover sensitive data.

Secondary server To detect and mitigate backdoor attacks in privacy-preserving DNN models, we set up another server to assist the computing server, named P_3. The secondary server and the computing servers jointly calculate whether the privacy-preserving DNN model has been injected with a backdoor.

The computing servers hold the datasets to train the model and recover the trigger. These datasets are provided by data holders, and shared data uses the RSS technique among the three computing servers. The computing servers and the secondary server communicate with each other to recover the trigger used by the backdoor attack. When

Table 1 Compare with prior related work

Framework	Privacy-preserving Techniques Used		Privacy-preserving Target		Supporting Backdoor Attack Detection	Implement		
	HE	MPC	Model	Data		C++	Python	Others
SecureNN [5]	✗	✓	✓	✓	✗	✓	✗	✗
ABY3 [26]	✗	✓	✓	✓	✗	✓	✗	✗
SecureML [3]	✗	✓	✓	✓	✗	✓	✗	✗
FALCON [8]	✗	✓	✓	✓	✗	✓	✗	✗
BLAZE [7]	✗	✓	✓	✓	✗	✓	✗	✗
CryptoNets [27]	✓	✗	✓	✓	✗	✗	✗	✓
CryptoDL [28]	✓	✗	✓	✓	✗	✓	✗	✗
MP2ML [29]	✓	✓	✓	✓	✗	✓	✗	✗
GAZELLE [30]	✓	✓	✓	✓	✗	✓	✗	✗
Neural Cleanse [9]	✗	✗	✗	✗	✓	✗	✓	✗
ABS [21]	✗	✗	✗	✓	✓	✗	✓	✗
STRIP [22]	✗	✗	✗	✗	✓	✗	✓	✗
TABOR [24]	✗	✗	✗	✗	✓	✗	✓	✗
PoisHygiene [31]	✗	✗	✗	✗	✓	✗	✓	✗
MP-BADNet$^+$ (This work)	✗	✓	✓	✓	✓	✓	✗	✗

detecting, identifying, and mitigating backdoor attacks involves steps requiring privacy preservation, they are computed by computing servers. Finally, the privacy of the data and model parameters is completely hidden for these non-collusion servers through the interaction between the 4-party servers.

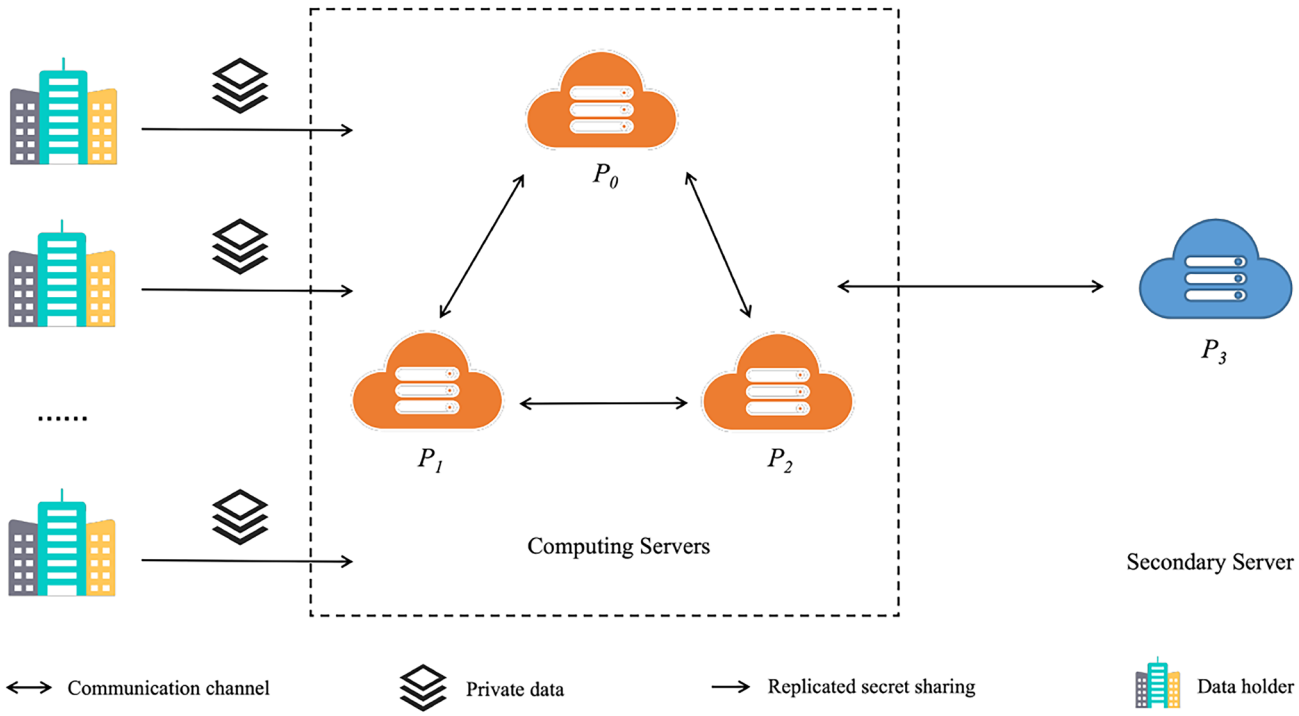


Fig. 1 The system architecture of MP-BADNet$^+$. First, data holders use the replicated secret sharing technique to share fragments of sensitive data to three computing servers for model training. Then, the three computing servers train the model based on the MPC technique. Finally, they jointly compute with the secondary servers to detect whether the privacy-preserving DNN model is injected with backdoors

3.2 Backdoor attack models

In this paper, we assume that the user outsources the model to the servers for computation due to limited computational and storage resources. These servers train the model secretly by using MPC techniques. The datasets used to train the model are provided by different data holders and are shared via RSS techniques. The infected DNN model is susceptible to backdoor attacks because it does not reduce the prediction accuracy of clean datasets and has a high probability of classifying any poisoned data as the target label regardless of its original label.

The backdoor attack in this paper assumes that each target label corresponds to only one type of trigger, and the DNN model has only one target label. To put it differently, backdoor attacks with multi-target labels and multi-triggers are beyond the scope of MP-BADNet$^+$.

3.3 Threat models and design goals

The threat model of MP-BADNet$^+$ is the same as in the previous work [3, 8, 26], considering an honest-majority adversarial setting. In the three-party computing server of this paper, we assume that at most one participant is corrupted. In addition, we assume that the communication channels between the four servers (three computing servers and a secondary server) are secure. That is, their communication private keys are securely stored and not easily leaked.

The goals of MP-BADNet$^+$ are as follows:

- Preserving privacy. We want to preserve the privacy of the training data and model parameters in the DNN model.
- Detecting backdoor. We want to detect whether the privacy-preserving DNN model is attacked by a backdoor and try to know which label is the target of the attack.
- Identifying backdoor. We want to recover the triggers used by the backdoor attack while preserving the privacy of the input data and model parameters.
- Mitigating backdoor. We want to use the recovered triggers for backdoor mitigation in the privacy-preserving DNN model.

4 Notations

Let P_0, P_1, P_2, P_3 denote the four different participants, where P_0, P_1, P_2 are computing servers, and P_3 is secondary server. The three computing servers are based on MPC for secure computing, we use P_{i+1} and P_{i-1} to denote P_i's next party and previous party, respectively. By way of example, the next party of P_0 is P_1, and the previous party of P_0 is P_2. Let $\|x\|^m = (x_1, x_2, x_3)$ to denote RSS of a secret x modulo m. In other words, $x \equiv x_1 + x_2 + x_3 \pmod{m}$, where x_1, x_2, x_3 are random. We set $\|x\|^m$ to mean (x_1, x_2) is held by P_0, (x_2, x_3) by P_1, and (x_3, x_1) by P_2.

In MP-BADNet$^+$, we use *mask* to denote a matrix, determining how many triggers can overwrite the original image. The range of *mask* is (0, 1). We use *pattern* to denote the trigger pattern, a matrix with the same size as the original image. When $mask_{i,j}$ is 1 (where i, j represent the pixel in the i-th row and j-th column), it means that the pixel value of the i-th row and j-th column of the original image is completely replaced by the pixel value of $pattern_{i,j}$. When $mask_{i,j}$ is 0, the constructed adversarial input pixel is completely the original image and does not cover any pixel of $pattern_{i,j}$.

5 Concreting protocol

Based on FALCON [8] and Neural Cleanse [9], we propose a method for backdoor attack detection, identification, and mitigation for a privacy-preserving DNN model that hides the backdoor and is trained using MPC technique. The advantages of MP-BADNet$^+$ are as follows: (1) Multiple data holders can use their data to train a model jointly; (2) It can preserve the privacy of input data and model parameters; (3) It can detect whether the model trained by MPC is embedded with a backdoor and recovers the trigger if it is, and further identify the target label of the backdoor attack; (4) It can mitigate backdoor attacks in privacy-preserving DNN model.

5.1 System design

We describe in detail the technique of four servers to jointly detect, identify, and mitigate backdoor attacks in the privacy-preserving DNN model. Our general design idea is consistent with Neural Cleanse [9]. Suppose we have a DNN model with L labels. Consider a normal label L_i and a target label L_t, where $i \neq t$. If there is a backdoor attack trigger T_t that misclassifies the label L_i as the target label L_t, then the minimum perturbation required for all inputs: $\delta_{i \to t} \leq |T_t|$.

Since all labels are to be misclassified as target labels and the trigger is a small stamp. Thus we have $\delta_{\forall \to t} \ll \min_{i, i \neq t} \delta_{\forall \to i}$, where $\delta_{\forall \to t}$ represents the minimum perturbation required from any label to target label L_t.

As described above, our system mainly includes the following steps:

- **Step 1:** For a given label, we regard it as a potential target label in a backdoor attack. We designed an optimization method that runs on four servers to recover the trigger that misclassifies all other labels as the minimum modification value of the target label.
- **Step 2:** For all labels in the DNN model, perform step 1. If the DNN model contains N labels, N potential triggers will eventually be obtained.
- **Step 3:** For N potential triggers, we analyze the size of the trigger. Any trigger significantly smaller than the other candidates is a real trigger, and its corresponding label is the target label.
- **Step 4:** Use the trigger corresponding to the target label to generate adversarial input, which is combined with clean input to mitigate the backdoor.

The reverse engineering algorithm consists of the components of step 1 and step 2. And step 3 and step 4 are the outlier detection algorithm and the backdoor mitigation algorithm, respectively. The computation of the reverse engineering algorithm is done jointly by the secondary server and computing servers, and N potential triggers are generated. Next, the secondary server runs the outlier detection algorithm to get the target trigger and the target label. Finally, the backdoor mitigation algorithm uses recovered trigger for backdoor mitigation. The schematic diagram of the detection and mitigation system is shown in Fig. 2.

5.1.1 Reverse engineering algorithm

To recover the triggers used in the backdoor attack, we designed a reverse engineering algorithm that runs on four servers. When the algorithm step involves input data and model parameters, it runs on computing servers with the MPC technique. Otherwise, it runs in plaintext on the secondary server, which mainly processes auxiliary computation in plaintext and does not involve any input data and model parameters. In this work, we assume that the secondary server and computing servers are non-collusion. In order to make the backdoor detection converge quickly, we use the Adam optimizer [33] to optimize *mask* and *pattern*.

The Algorithm 1 describes the flow of the reverse engineering algorithm in detail. $\|\cdot\|^m$ means that it is computed in ciphertext on the computing servers. All other computations of the algorithm are done in plaintext on the secondary server.

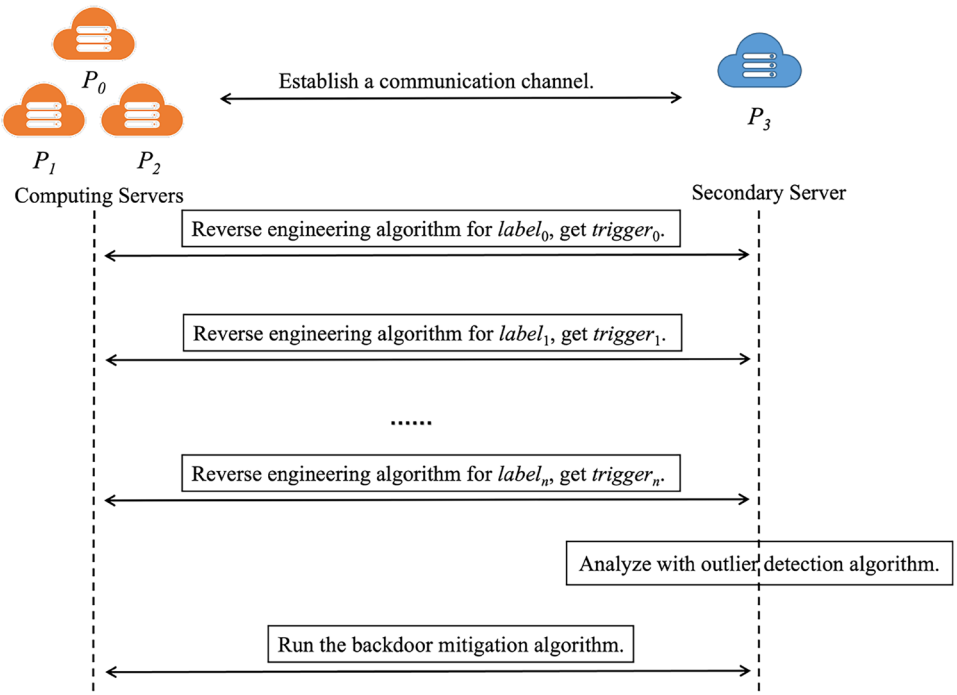


Fig. 2 The schematic diagram of the detection and mitigation system. First, reverse engineering algorithms are run for each label to generate potential triggers. Second, the size of these potential triggers is analyzed. Finally, run the backdoor mitigation algorithm to reduce or eliminate the effect of the backdoor attack

Algorithm 1 Reverse Engineering Algorithm

Input: $\|x\|^m$,$\|yTarget\|^m$, $pattern$,$mask$
Output: $maskBest$, $patternBest$, $maskUpsampleBest$

1: Initialize $maskTanh$, $patternTanh$, $maskUpsample$ and $patternRaw$ in ciphertext;
2: Initialize $maskBest, maskUnsampleBest, patternBest$;
3: Let $attackThreshold \leftarrow 0.99$, $regBest \leftarrow \infty$, $iterations \leftarrow 1000$, $iter \leftarrow 0$;
4: Perform one-hot encoding on $\|yTarget\|^m$ to generate $\|yTargetList\|^m$;
5: **while** iter $\leq$ iterations **do**
6: **for** $i := 0$ to $miniBatch$ **do**
7: Obtain $\|xBatch\|^m$ in $\|x\|^m$;
8: Use $\|xBatch\|^m$ to construct adversarial input $\|xBatchAdv\|^m$;
9: Calculate $lossReg, lossAcc$ from $\|xBatchAdv\|^m$;
10: Update $maskTanh$, $maskUpsample$ and $patternRaw$ in ciphertext;
11: Set $List_{lossReg}$.push($lossReg$), $List_{lossAcc}$.push($lossAcc$);
12: **end for**
13: Process adamLR.step();
14: Obtain $lossRegAvg \leftarrow mean(List_{lossReg})$;
15: Obtain $lossAccAvg \leftarrow mean(List_{lossAcc})$;
16: **if** $(lossAccAvg \geq attackThreshold)$ and $(lossRegAvg < regBest)$ **then**
17: $maskBest \leftarrow maskTanh$;
18: $maskUpsampleBest \leftarrow maskUpsample$;
19: $patternBest \leftarrow patternRaw$;
20: $regBest \leftarrow lossRegAvg$;
21: **end if**
22: Check $cost$ modification;
23: iter++;
24: Check whether the algorithm ends early ? **break** : **continue**;
25: **end while**
26: Return $maskBest$, $patternBest$, $maskUpsampleBest$.

For example, to optimize the computation of *pattern*, the secondary server first computes *patternTanh* and then sends it to the computing servers using the RSS technique. The computing servers construct adversarial input for prediction in the ciphertext domain and compute the gradient of the *pattern*. Finally, the computing servers send the gradient to the secondary server for *pattern* optimization.

The role of the reverse engineering algorithm is to recover the potential target label trigger and steps 5-25 are the main procedure. In the Algorithm 1, steps 1-4 are the preprocessing operations, step 5-12 is to use the input data to optimize *mask* and *pattern*, and steps 16-21 are to check whether *mask* and *pattern* are optimal. Step 22 of the reverse engineering algorithm is to modify the *cost* to optimize *mask* and *pattern*. Step 24 is to detect whether to end the algorithm early.

5.1.2 Outlier detection algorithm

The Algorithm 2 gives a detailed demonstration of the outlier detection algorithm. We use the L_1 norm to analyze the size of potential triggers and *Median Absolute Deviation* (*MAD*) technique to analyze outliers, which is flexible in the presence of multiple outliers [34]. The outlier detection algorithm first calculates *MAD* between all data points and the median. When assuming that the underlying distribution is a normal distribution, a constant 1.4826 will be used to normalize the outlier value. In this work, any label with an outlier value greater than 2 is marked as an outlier, that is, the label is the target label of a backdoor attack. $List_{mask}$ is the optimal *mask* computed by the reverse engineering algorithm. As mentioned above, step 6 in the algorithm is to normalize *MAD* under the normal distribution. Steps 7-15 are to calculate all indexes with an outlier value greater than 2.

Algorithm 2 Outlier Detection Algorithm

Input: $List_{mask}$
Output: $List_{Flag}$ and $List_{FlagL1Norm}$

Set $ConsistencyConstant \leftarrow 1.4826$;
for each $mask$ in $List_{mask}$ **do**
    Calculate its $L1$ norm to obtain $List_{L1Norm}$;
end for
Calculate $median$ and MAD of $List_{L1Norm}$;
Update $MAD \leftarrow ConsistencyConstant * MAD$;
for $i := 0$ to $NumLabels$ **do**
    if $(List_{L1Norm}[i] > median)$ **then**
        continue;
    end if
    if $abs(List_{L1Norm}[i] - median)/MAD > 2$ **then**
        $List_{Flag}$.push(i);
        $List_{FlagL1Norm}$.push$(List_{L1Norm}[i])$;
    end if
end for
Return $List_{Flag}$ and $List_{FlagL1Norm}$.

Algorithm 2 can be computed locally on the secondary server P_3. At the same time, the input data and model parameters are not leaked to the computing servers P_0, P_1 and P_2, and the secondary server P_3. However, P_3 as a plaintext server knows the trigger that is eventually recovered by the reverse engineering algorithm.

5.1.3 Backdoor mitigation algorithm

Algorithm 3 describes the details of the backdoor mitigation algorithm. The function of the backdoor attack mitigation algorithm is to mitigate the backdoor detected by the Algorithms 1 and 2. First, the trigger recovered by the reverse engineering algorithm is attached to the clean input to construct the adversarial input. Then the clean input and the adversarial input are input into the DNN model containing the backdoor in the ciphertext domain. Finally, Algorithm 3 uses the neuron pruning technique to remove neurons sensitive to triggers.

Algorithm 3 describes the specific mitigation process. Step 1 combines the triggers obtained by the reverse engineering algorithm with the clean input to construct adversarial input. Step 2 computes the neuron activation values of clean input and adversarial input to the model. Step 3 finds the difference in neuron activation values and sorts them. Steps 4 and 5 are to remove the backdoor neurons and output the modified model.

5.2 Security analysis and proofs

We give the security analysis about the aspects of channel security, data confidentiality, and privacy-preserving in Sect. 5.2.1. The details of the security proof are in Sect. 5.2.2.

5.2.1 Security analysis

According to the following steps among the computing servers and secondary server, the overall interaction diagram of MP-BADNet$^+$ is shown in Fig. 3.

Algorithm 3 Backdoor Mitigation Algorithm

Input: $\|x\|^m$ and $trigger$
Output: $model$

Attach the trigger to $\|x\|^m$ to construct adversarial input $\|x_{adv}\|^m$;
Set $a \leftarrow model(\|x\|^m)$ and $b \leftarrow model(\|x_{adv}\|^m)$;
Set $diff \leftarrow a - b$ and sort by $diff$ in ciphertext;
Remove the neurons with higher $diff$ rank in ciphertext;
Return $model$.

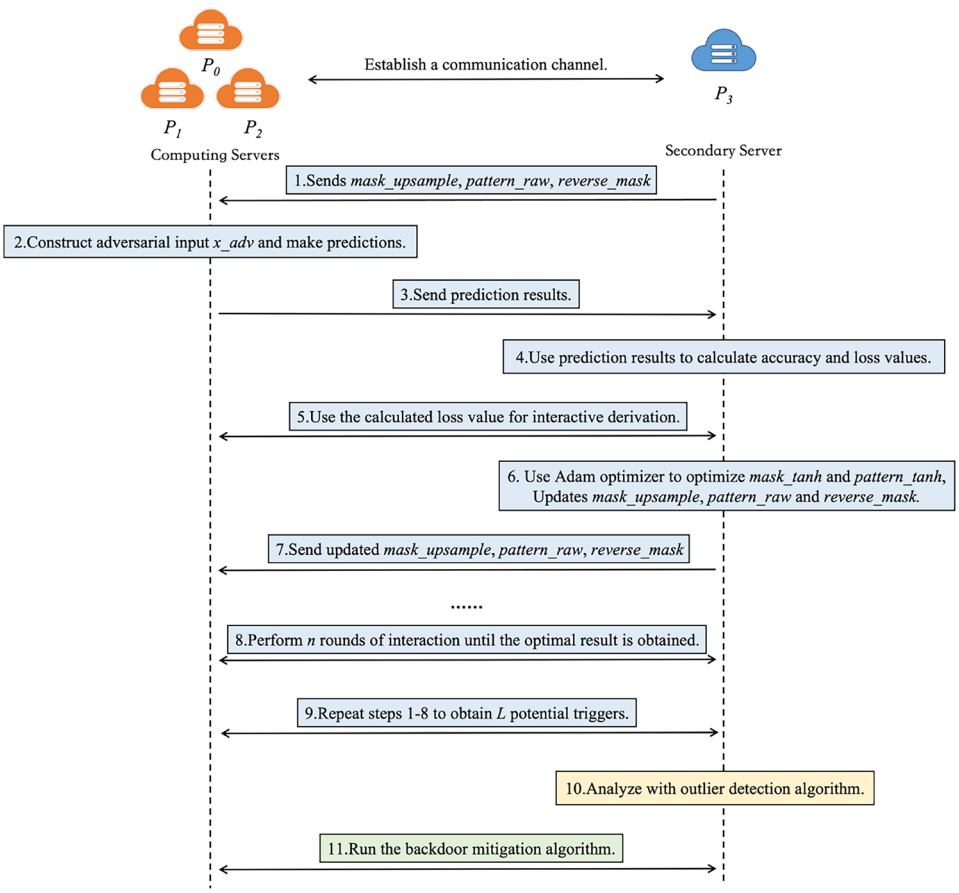


Fig. 3 The overall interaction diagram

Channel security Between the computing servers and the secondary server, it is assumed to establish a secure channel to ensure that the transmitted information is not compromised. Their communication private keys are securely stored and not easily leaked.

Data confidentiality In the three-party computing server, we assume that the security threshold is the honest majority, i.e., at most one participant is corrupted. These datasets are provided by data holders, and shared data uses the RSS technique among the three computing servers. In the scheme, the data is also secure if one of the computing servers is compromised.

Privacy-preserving The scheme achieves privacy preservation for the training data and the training model. Step 2 constructs and uses adversarial inputs for model prediction. Since the private inputs are required to construct adversarial inputs, this step is performed among the computing servers in an MPC style. Step 5 is to compute the gradient interactively among the computing servers and the secondary server, in which we use the batch technique to process the input data. Thus, only the average of each input data batch is used to calculate the gradients for *mask* and *pattern*. Step 10 is calculated by the secondary server since this step involves only the potential triggers and does not reveal private information. Other steps do not involve any private input.

5.2.2 Security proofs

We follow the real world-ideal world simulation paradigm [35–37] to prove the security. We setup hybrid interactions to prove the security of the functionality, where the sub-protocols used in the algorithm are replaced by the corresponding ideal functionalities which are indistinguishable.

Theorem 1 *Algorithm* 1 *securely realizes the ideal functionality* $\mathcal{F}_{REA}$ (*Fig.* 4) *with abort in the presence of one malicious party, given the ideal functionality of the protocols* (*i.e.*, $\mathcal{F}_{Mult}$, $\mathcal{F}_{Trunc}$, $\mathcal{F}_{Reconst}$, $\mathcal{F}_{PC}$, $\mathcal{F}_{WA}$, $\mathcal{F}_{ReLU}$, $\mathcal{F}_{Maxpool}$, $\mathcal{F}_{Pow}$, $\mathcal{F}_{Div}$ *and* $\mathcal{F}_{BN}$) *proposed in Falcon* [8].

Input: The parameters in Algorithm 1: $\|x\|^m$, $\|yTarget\|^m$, $pattern$, $mask$
Output: The random shares.

1. Reconstruct x and $yTarget$.
2. Complete steps 1-5 of Algorithm 1.
3. Send random shares of $maskTanh$, $patternTanh$, $maskUpsample$, $patternRaw$ and $yTargetList$ to P_0, P_1 and P_2.
4. For each iteration t:
   - For each $miniBatch$:
     - Compute $xBatchAdv$ from x, $pattern$ and $mask$.
     - Send random shares of $xBatchAdv$ to P_0, P_1 and P_2.
     - Calculate $lossReg$ and $lossAcc$ from $xBatchAdv$.
     - Send random shares of $lossReg$ and $lossAcc$ to P_0, P_1 and P_2.
     - Update $maskTanh$, $maskUpsample$ and $patternRaw$.
     - Send random shares of $maskTanh$, $maskUpsample$ and $patternRaw$ to P_0, P_1 and P_2.
     - Complete step 11 of Algorithm 1.
   - Complete steps 13-24 of Algorithm 1.
5. Return random shares of the output.

Fig. 4 The reverse engineering algorithm ideal functionality $\mathcal{F}_{REA}$

Proof We assume that P_2 is corrupted without loss of generality. Simulator $\mathcal{S}$ simulates the view of corrupt P_2. The process of $\mathcal{S}$ in Algorithm 1 is listed as follows. Steps 1-5 and Steps 13-24 are computed by P_3 and therefore do not need to be simulated. Step 8 can be simulated using the simulator for $\mathcal{F}_{Mult}$, $\mathcal{F}_{Trunc}$ and $\mathcal{F}_{Reconst}$. Steps 9 and 10 can be simulated using the simulators for $\mathcal{F}_{Mult}$, $\mathcal{F}_{Trunc}$, $\mathcal{F}_{Reconst}$, $\mathcal{F}_{PC}$, $\mathcal{F}_{WA}$, $\mathcal{F}_{ReLU}$, $\mathcal{F}_{Maxpool}$, $\mathcal{F}_{Pow}$, $\mathcal{F}_{Div}$ and $\mathcal{F}_{BN}$. Step 11 is calculated interactively by P_0, P_1, P_2 and P_3 and can be simulated using the simulators for $\mathcal{F}_{Trunc}$ and $\mathcal{F}_{Reconst}$. Other steps do not leak any information. We show the simulation by proceeding with a series of replacements by the ideal functionalities from the real view to the ideal view, in which we find that the neighboring views are indistinguishable. From the discussion above, Algorithm 1 securely realizes the ideal functionality $\mathcal{F}_{REA}$ (Fig. 4) with abort in the presence of one malicious party. □

Discussion Algorithm 2 does not need to be simulated. Since the input of Algorithm 2 is only the potential triggers computed by the reverse engineering algorithm and does not involve any private data and model privacy, it can be computed in plaintext by P_3.

Theorem 2 *Algorithm* 3 *securely realizes the ideal functionality* $\mathcal{F}_{BMA}$ (*Fig.* 5) *with abort in the presence of one malicious party, given the ideal functionality of the protocol* (*i.e.*, $\mathcal{F}_{Mult}$, $\mathcal{F}_{Trunc}$, $\mathcal{F}_{Reconst}$, $\mathcal{F}_{PC}$, $\mathcal{F}_{WA}$, $\mathcal{F}_{ReLU}$, $\mathcal{F}_{Maxpool}$, $\mathcal{F}_{Pow}$, $\mathcal{F}_{Div}$ *and* $\mathcal{F}_{BN}$) *proposed in Falcon* [8].

Proof Similar to the proof of Theorem 1, the process of $\mathcal{S}$ in Algorithm 3 is listed as follows. Step 1 and Step 2 can be simulated by using the simulators for $\mathcal{F}_{Mult}$, $\mathcal{F}_{Trunc}$, $\mathcal{F}_{Reconst}$, $\mathcal{F}_{PC}$, $\mathcal{F}_{WA}$, $\mathcal{F}_{ReLU}$, $\mathcal{F}_{Maxpool}$, $\mathcal{F}_{Pow}$, $\mathcal{F}_{Div}$ and $\mathcal{F}_{BN}$. Step 3 and Step 4 are calculated interactively by computing servers and the secondary server and can be simulated using the simulators for $\mathcal{F}_{Trunc}$ and $\mathcal{F}_{Reconst}$. Other steps do not leak any information. We also show the simulation by proceeding with a series of

Table 2 The Network Structure of MP-BADNet$^+$

Layer	Input Size	Output Size	Description
Fully Connected Layer	28×28	128	Fully connected layer
ReLU Activation	128	128	ReLU on each input
Fully Connected Layer	128	128	Fully connected layer
ReLU Activation	128	128	ReLU on each input
Fully Connected Layer	128	10	Fully connected layer
ReLU Activation	10	10	ReLU on each input

Fig. 5 The backdoor mitigation algorithm ideal functionality $\mathcal{F}_{BMA}$

Input: The parameters in Algorithm 3: $\|x\|^m$ and $trigger$
Output: The random shares.

1. Reconstruct x.
2. Construct adversarial input x_{adv}.
3. Compute $a \leftarrow model(x)$, $b \leftarrow model(x_{adv})$.
4. Send random shares of a and b to P_0, P_1 and P_2.
5. Complete steps 3-4 of Algorithm 3.
6. Return random shares of the output.

replacements by the ideal functionalities from the real view to the ideal view, in which we find that the neighboring views are indistinguishable. From the discussion above, Algorithm 1 securely realizes the ideal functionality $\mathcal{F}_{BMA}$ (Fig. 5) with abort in the presence of one malicious party. □

6 Experimental evaluation

6.1 Experiment setup

We implement our MP-BADNet$^+$ framework using the communication backend of FALCON [8]. We run our experiments on Ubuntu 16.04 LTS with an Intel Xeon CPU @ 2.20 GHz processor and 16 GB RAM. Moreover, four processes are used to simulate four different participants. We evaluate MP-BADNet$^+$ on the MNIST dataset. It consists of 60,000 training samples and 10,000 test samples. Each sample is a 28×28 pixel gray-scale hand written digital image. The network used in this paper is a 3-layer fully connected layer neural network proposed like Mohassel's scheme [3] and uses rectified linear unit(ReLU) function as an activation function in each layer. The softmax function is used to obtain the output probability after the final output layer. The network structure is as follows Table 2. The network structure of it is shown in Fig. 6. Our protocol uses fixed-point encoding with 11 bits of precision and three different moduli $L = 2^l$, a prime number p and 2. In particular, we set $l = 2^5$ and $p = 67$.

We reproduce the attack method of BadNets [11] to poison the MNIST dataset. As shown in Fig. 7, (a) is the original trigger, (b) is the clean input, and (c) is the infected input. We poisoned about 10% of the training dataset with

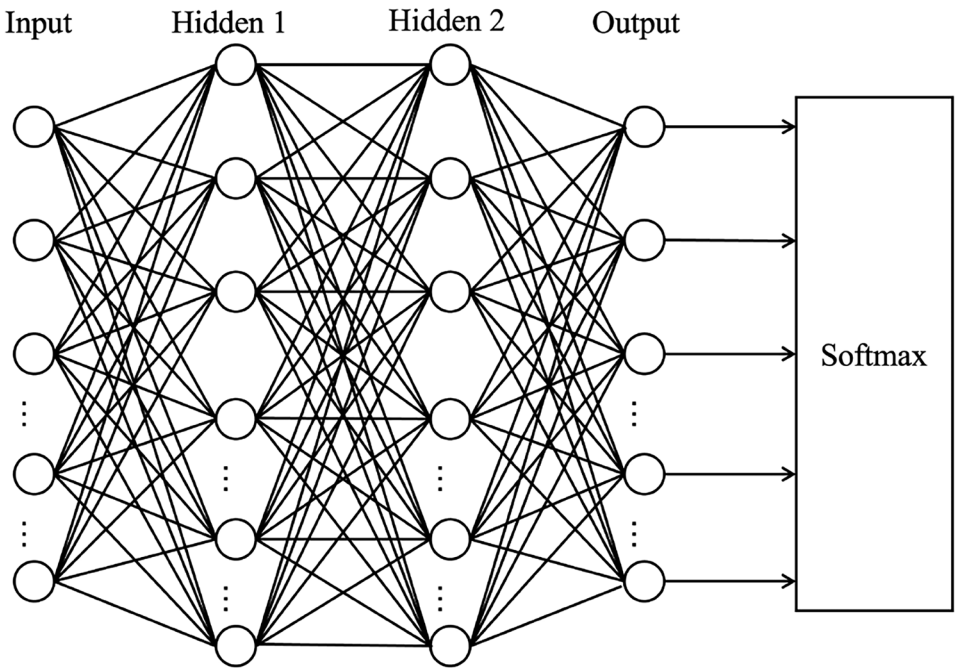
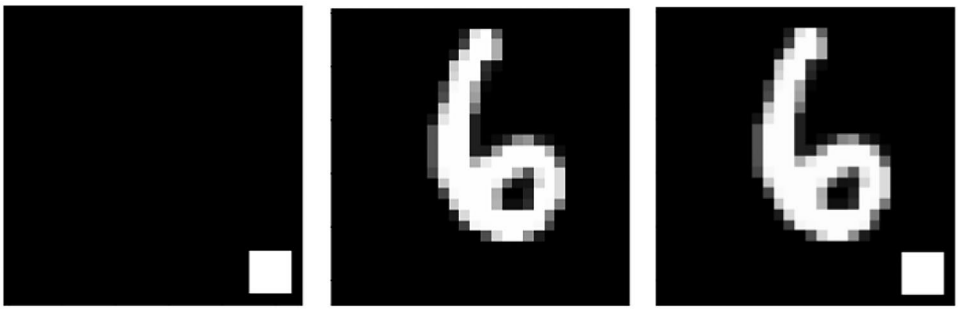


Fig. 6 The network structure of MP-BADNet$^+$. It consists of 3 fully connected layers with the ReLU activation function and is finally normalized by the softmax function



(a) Trigger **(b)** Clean input **(c)** Infected input

Fig. 7 Backdoor attacks in MP-BADNet$^+$. **a** shows an original trigger, **b** shows a clean input, and **c** shows an infected input

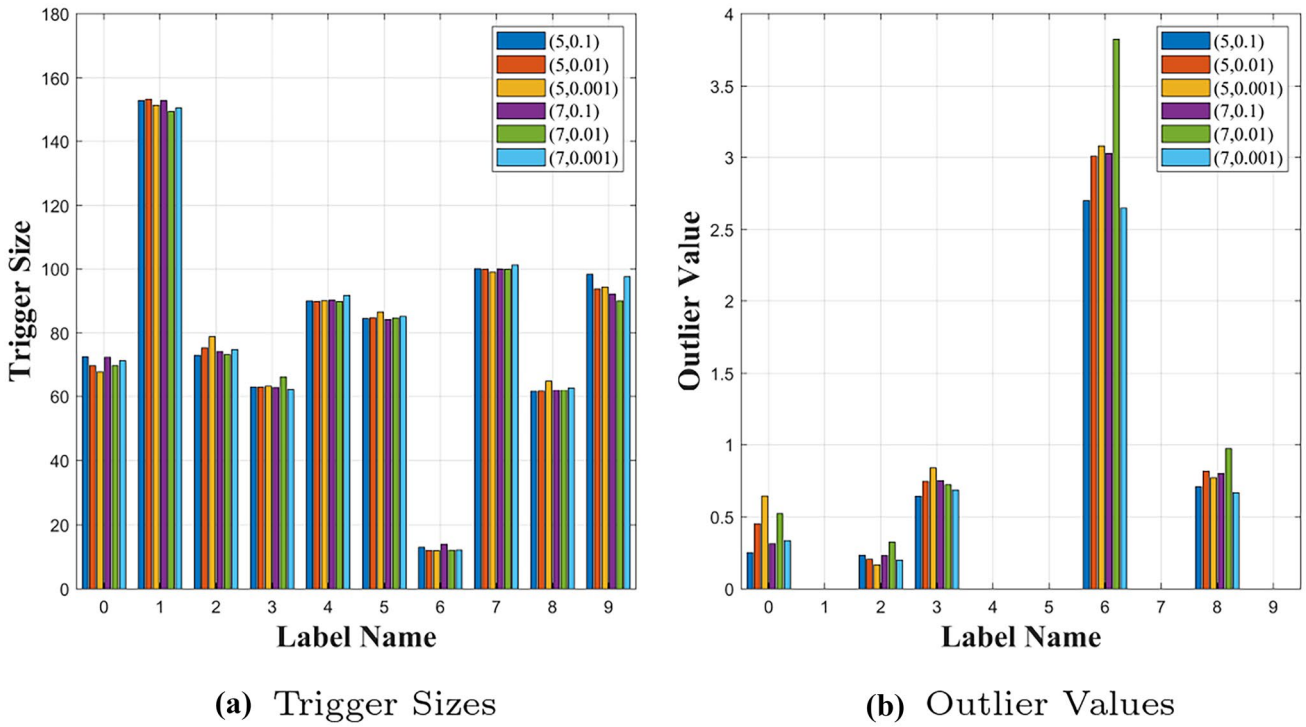
Table 3 The performance comparison in plaintext and ciphertext

Type	*Pat*	*LR*	Steps	Time(hour)	Comm.cost(GB)
Ciphertext	5	0.001	413	12.6373	14.9417
Ciphertext	5	0.01	291	8.9394	10.6276
Ciphertext	5	0.1	291	9.0391	10.6306
Ciphertext	7	0.01	404	12.3477	14.6202
Ciphertext	7	0.1	408	12.2586	14.7707
Plaintext	5	0.001	545	1.4385	–
Plaintext	5	0.01	346	0.8879	–
Plaintext	5	0.1	276	0.7281	–
Plaintext	7	0.01	425	1.0219	–
Plaintext	7	0.1	385	0.9996	–

target labels set to 6 and each target label with a size of 4×4 pixels. We then use the infected dataset for model training. Our experimental results suggest that the trained DNN model can successfully classify clean inputs, while the infected inputs are identified as target labels with high probability.

After 15 epochs of model training, we found that the prediction accuracy of plaintext and ciphertext is about 97.17% for the clean dataset, while the prediction accuracy of plaintext is 100% and 99.97% for ciphertext for the infected dataset.

It is worth noting that the size of the dataset used in our reverse engineering algorithm is 60,000 unless otherwise specified.



(a) Trigger Sizes **(b)** Outlier Values

Fig. 8 Backdoor attack detection on *infected* model in ciphertext. **a** shows the trigger size recovered by the reverse engineering algorithm, and **b** gives the result after executing the outlier detection algorithm

Table 4 Detection cost comparison of reverse engineering algorithms using different dataset sizes in the infected model

Type	Dataset Size	Time(hour)	textbfComm. cost(GB)	Infected Label Trigger Size
Ciphertext	1280	0.4824	0.5744	11.8418
Ciphertext	2560	0.6820	0.9712	12.1855
Ciphertext	5120	1.0295	1.3229	11.5117
Ciphertext	12,800	1.1225	2.7156	11.8154
Plaintext	1280	0.0469	–	12.8625
Plaintext	2560	0.0783	–	12.4059
Plaintext	5120	0.1226	–	11.6409
Plaintext	12,800	0.2673	–	11.7198

Table 5 Comparison of reverse engineering algorithms with clean models in ciphertext

Model Type	*Pat*	*LR*	Steps	Time(hour)	Comm.cost(GB)
Infected	5	0.1	291	9.0391	10.6306
Infected	5	0.01	291	8.9394	10.6276
Infected	5	0.001	413	12.6373	14.9417
Infected	7	0.1	408	12.2586	14.7707
Infected	7	0.01	404	12.3477	14.6202
Infected	7	0.001	442	13.5132	15.9610
Clean	5	0.1	271	8.3806	9.9276
Clean	5	0.01	273	8.3535	9.9917
Clean	5	0.001	346	9.9613	12.5687
Clean	7	0.1	395	12.0529	14.3057
Clean	7	0.01	401	12.3012	14.5187
Clean	7	0.001	589	17.8112	21.1561

6.2 Detection results

We analyze the performance of the algorithm by setting different patience (*Pat*) and learning rate (*LR*). The former is to adjust the weight of the optimization target, and the latter is the learning rate of the Adam optimizer. In our experiment, *Pat* is set at 5 and 7, and the *LR* is set at 0.1, 0.01, and 0.001, respectively. The performance comparison between plaintext and ciphertext is shown in Table 3. In ciphertext, the *LR* is 0.01 and the *Pat* is 5, the time spent and the communication are the lowest.

Figure 8(a) presents the size of the triggers after the reverse engineering algorithm for each label. As we can observe from Fig. 8, the trigger size of label 6 is significantly smaller than that of other labels. Furthermore, Fig. 8(b) shows the results of the outlier detection algorithm, which indicates that the outliers of label 6 are greater than 2. Since the trigger sizes of labels 1, 4, 5, 7, and 9 are larger than the median, it is impossible for these labels to be the target labels. We show the outliers as 0 in

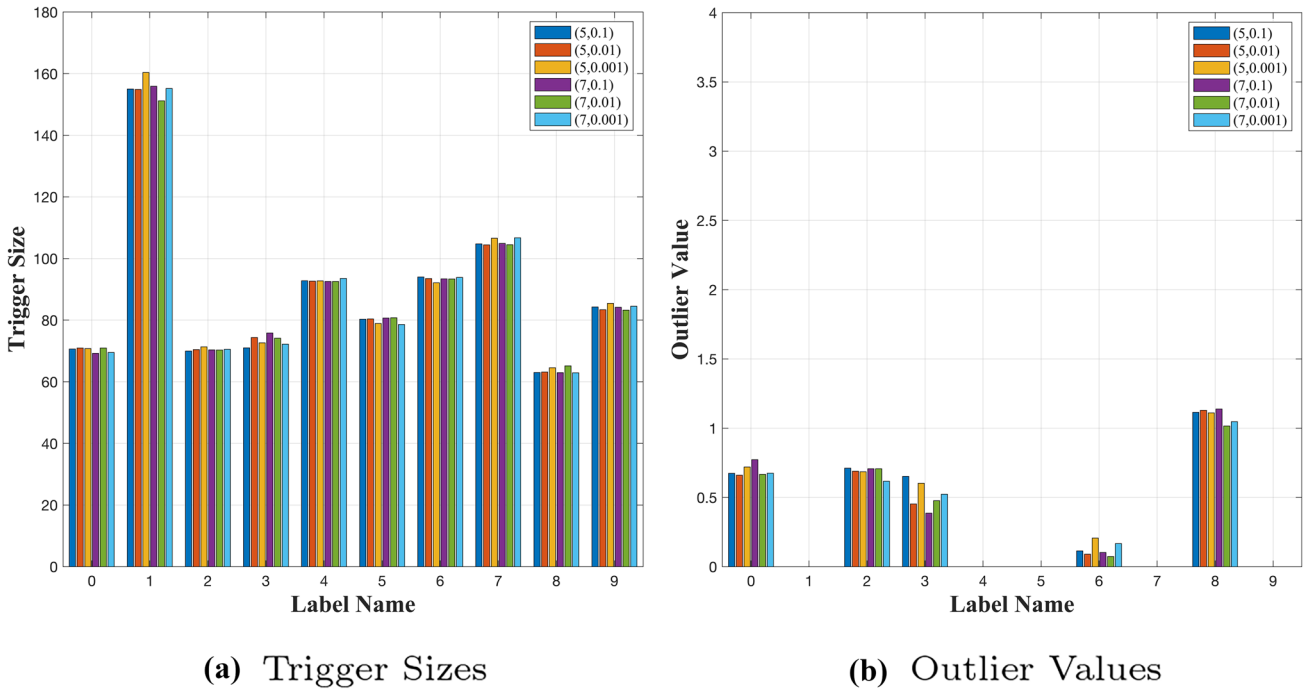


Fig. 9 Backdoor attack detection on *clean* model in ciphertext. **a** shows the trigger size recovered by the reverse engineering algorithm, and **b** gives the result after the executing the outlier detection algorithm

Table 6 The comparison of reverse engineering algorithm using different datasets sizes with the infected model and the clean model in ciphertext

Type	Dataset Size	Time(hour)	Comm.cost(GB)
Infected	1280	0.4824	0.5744
Infected	2560	0.6820	0.9712
Infected	5120	1.0295	1.3229
Infected	12,800	1.1225	2.7156
Clean	1280	0.5200	0.6100
Clean	2560	0.7750	1.0317
Clean	5120	0.9588	1.2986
Clean	12,800	1.1400	2.8142

Fig. 8(b). Note that the parameters (5, 0.1) mean that the *Pat* is 5 and the *LR* is 0.1.

Table 4 shows the detection cost of the reverse engineering algorithm for different datasets sizes in the infected model. We conducted experiments with dataset sizes 1,280, 2,560, 5,120, and 12,800, respectively. *Pat* and *LR* are set to 5 and 0.01. We can conclude from the experimental results that the smaller the dataset, the lower the time and communication overhead. Despite the smaller dataset used by the reverse engineering algorithm, the trigger of the attacked label can be recovered.

The experimental results described above show that MP-BADNet$^+$ can successfully detect backdoors in poisoned models and identify target labels while preserving the privacy of input data and model parameters. The efficiency of backdoor detection in the ciphertext domain is approximately $4\times - 12\times$ slower than in the plaintext domain.

6.3 Comparison with clean model

Figure 9(a) shows the trigger size recovered by the reverse engineering algorithm in the clean model. In contrast to Fig. 8, the trigger size of label 6 tends to be normal. Figure 9(b) shows the values after the execution of the outlier detection algorithm, where the outlier values are less than 2 when the *Pat* is 5 and 7. The results in Fig. 9(b) mean no backdoor in the model.

Table 5 shows the experimental comparison between the infected model (with backdoor) and the clean model (without backdoor) in the ciphertext domain. The size of the dataset used for the reverse engineering algorithm of this experiment is 60,000. And Table 6 shows the detection cost of the reverse engineering algorithm for different datasets sizes with the infected model and the clean model in the ciphertext domain. We conducted experiments with dataset sizes 1,280, 2,560, 5,120, and 12,800, respectively. *Pat* and *LR* are set to 5 and 0.01.

The experimental results show that the time and communication cost spent on the infected model during the reverse engineering algorithm is similar to that on the clean model in the ciphertext domain.

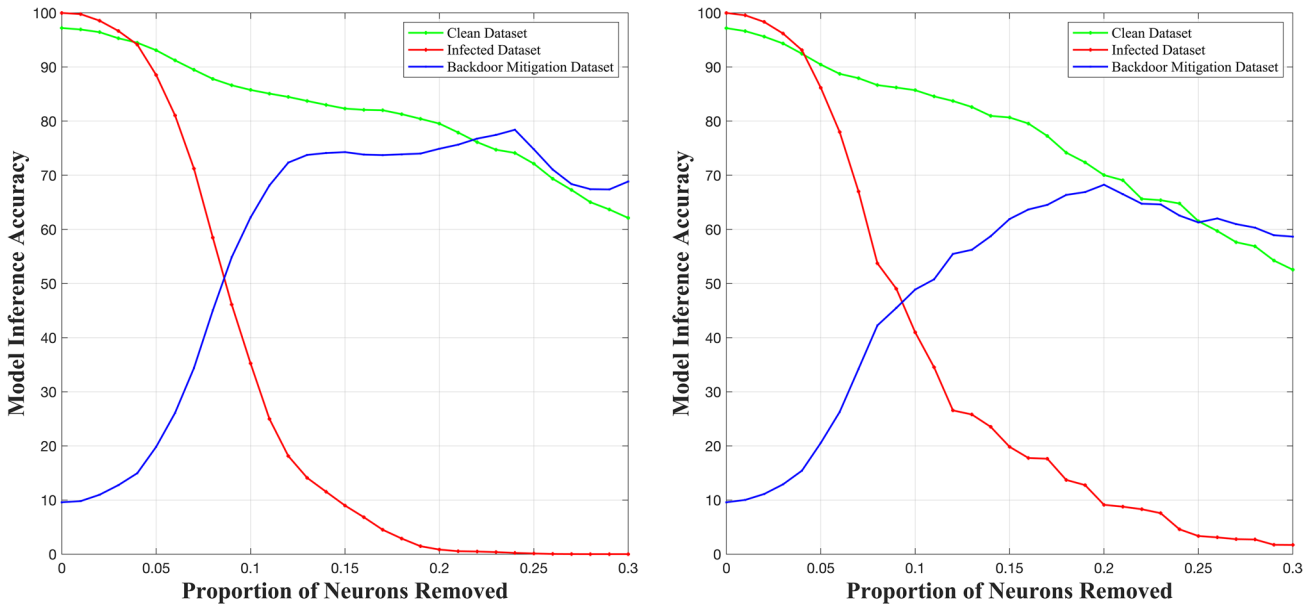


Fig. 10 Backdoor mitigation result

6.4 Mitigation results

We use reverse engineered triggers with (*Pat*, *LR*) values at (5, 0.01) to construct the backdoor mitigation experiments. We compute the neuron activation values corresponding to the clean and infected datasets in the DNN model and remove the neurons with a high difference between activation values.

As shown in Fig. 10, we give the model inference accuracy among the different types from the same dataset in the ciphertext domain and the plaintext domain, respectively. The green line is the accuracy in the clean dataset, the red line is the success rate of the backdoor attack after mitigation in the infected dataset, and the blue line is the accuracy of correctly identifying the original label in the infected dataset instead of the false target label.

The experimental results show that the backdoor mitigation effect in ciphertext rarely differs from that in plaintext. That means the backdoor mitigation can achieve those aims in ciphertext as well as in plaintext and also provide the property of privacy-preserving. We also find that the model performed best with the neuron removal proportion of 0.15-0.2. It is noted that the backdoor mitigation algorithm achieves a trade-off between accuracy and neuron removal proportion since the backdoor mitigation algorithm incurs a loss in the inference accuracy.

7 Conclusion and future work

In this paper, we have proposed MP-BADNet$^+$, a novel framework for detecting, identifying, and mitigating backdoor attacks in a privacy-preserving DNN model. The privacy of input data and model parameters has been preserved using the MPC technique with honest-majority adversarial settings. We have given the security analysis and formal security proof following the real world-ideal world simulation paradigm. We have demonstrated through our experiments that MP-BADNet$^+$ can effectively detect backdoors, identify target labels, and mitigate the impact of backdoors both in the plaintext and ciphertext domain. In the future, we will focus on privacy preservation backdoor attack detection and mitigation algorithms with high efficiency suitable for IoT devices.

Acknowledgements The authors are grateful to the anonymous reviewers for their invaluable suggestions and comments. This work is supported in part by National Natural Science Foundation of China under Grant 61972241 and 61972094, Natural Science Foundation of Shanghai under Grant 22ZR1427100 and 18ZR1417300, and Luo-Zhaorao College Student Science and Technology Innovation Foundation of Shanghai Ocean University.

Declarations

Conflict of interest We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

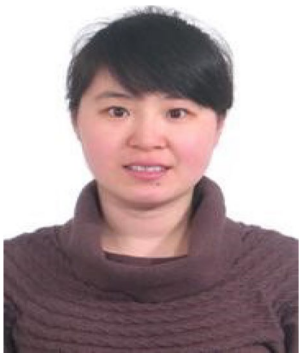
References

1. Zhu X, Vondrick C, Fowlkes C, Ramanan D (2016) Do we need more training data? Int J Comput Vision 119(1):76–92. https://doi.org/10.1007/s11263-015-0812-2
2. Stoica I, Song D, Popa A, Patterson D, Mahoney M, Katz R, Joseph A, Jordan M, Hellerstein J, Gonzalez J, et al. (2017) A berkeley view of systems challenges for ai. arXiv preprint arXiv:1712.05855
3. Mohassel P, Zhang Y (2017) Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy (SP), IEEE, Piscataway, NJ, pp 19–38. https://doi.org/10.1109/SP.2017.12
4. Yao A (1986) How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986), IEEE, Piscataway, NJ, pp 162–167. https://doi.org/10.1109/SFCS.1986.25
5. Wagh S, Gupta D (2019) Chandran N (2019) Securenn: 3-party secure computation for neural network training. Proceedings on Privacy Enhancing Technologies 3:26–49. https://doi.org/10.2478/popets-2019-0035
6. Chaudhari H, Choudhury A, Patra A, Suresh A (2019) Astra: High throughput 3pc over rings with application to secure prediction. In: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, Association for Computing Machinery, New York, pp 81–92. https://doi.org/10.1145/3338466.3358922
7. Patra A, Suresh A (2020) Blaze: blazing fast privacy-preserving machine learning. Proceedings 2020 Network and Distributed System Security Symposium. https://doi.org/10.14722/ndss.2020.24202
8. Wagh S, Tople S, Benhamouda F, Kushilevitz E, Mittal P (2021) Rabin T (2021) Falcon: Honest-majority maliciously secure framework for private deep learning. Proceedings on Privacy Enhancing Technologies 1:188–208. https://doi.org/10.2478/popets-2021-0011
9. Wang B, Yao Y, Shan S, Li H, Viswanath B, Zheng H, Zhao B (2019) Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: 2019 IEEE Symposium on Security and Privacy (SP), IEEE, Piscataway, NJ, pp 707–723. https://doi.org/10.1109/SP.2019.00031
10. Liu Y, Ma S, Aafer Y, Lee WC, Zhang X (2017a) Trojaning attack on neural networks. In: Network and Distributed System Security Symposium
11. Gu T, Liu K, Dolan-Gavitt B, Garg S (2019) Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access 7:47230–47244. https://doi.org/10.1109/ACCESS.2019.2909068
12. Huang L, Joseph A, Nelson B, Rubinstein B, Tygar D (2011) Adversarial machine learning. In: Proceedings of the 4th ACM workshop on Security and artificial intelligence, Association for Computing Machinery, New York, NY, USA, pp 43–58. https://doi.org/10.1145/2046684.2046692
13. Chen C, Wei L, Zhang L, Ning J (2021) MP-BADNet: A Backdoor-Attack Detection and Identification Protocol among Multi-Participants in Private Deep Neural Networks, Association for Computing Machinery, New York, NY, USA, p 104-109. https://doi.org/10.1145/3472634.3472660
14. Chen X, Liu C, Li B, Lu K, Song D (2017) Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526
15. Saha A, Subramanya A, Pirsiavash H (2020) Hidden trigger backdoor attacks. Proceedings of the AAAI Conference on Artificial Intelligence 34:11957–11965
16. Shokri R, et al. (2020) Bypassing backdoor detection algorithms in deep learning. In: 2020 IEEE European Symposium on Security and Privacy (EuroS &P), IEEE, pp 175–183

17. Salem A, Backes M, Zhang Y (2020) Don't trigger me! a triggerless backdoor attack against deep neural networks. arXiv preprint arXiv:2010.03282
18. Bagdasaryan E, Shmatikov V (2021) Blind backdoors in deep learning models. In: 30th USENIX Security Symposium (USENIX Security 21), pp 1505–1521
19. Liu Y, Xie Y, Srivastava A (2017b) Neural trojans. In: 2017 IEEE International Conference on Computer Design (ICCD), IEEE, Piscataway, NJ, pp 45–48. https://doi.org/10.1109/ICCD.2017.16
20. Liu K, Dolan-Gavitt B, Garg S (2018) Fine-pruning: Defending against backdooring attacks on deep neural networks. In: International Symposium on Research in Attacks, Intrusions, and Defenses, Springer, Springer International Publishing, Cham, pp 273–294. https://doi.org/10.1007/978-3-030-00470-5_13
21. Liu Y, Lee W, Tao G, Ma S, Aafer Y, Zhang X (2019) Abs: Scanning neural networks for back-doors by artificial brain stimulation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, Association for Computing Machinery, New York, NY, USA, pp 1265–1282. https://doi.org/10.1145/3319535.3363216
22. Gao Y, Xu C, Wang D, Chen S, Ranasinghe DC, Nepal S (2019) Strip: A defence against trojan attacks on deep neural networks. In: Proceedings of the 35th Annual Computer Security Applications Conference, Association for Computing Machinery, New York, NY, USA, pp 113–125. https://doi.org/10.1145/3359789.3359790
23. Chen H, Fu C, Zhao J, Koushanfar F (2019) Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In: IJCAI, pp 4658–4664
24. Guo W, Wang L, Xing X, Du M, Song D (2019) Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. arXiv preprint arXiv:1908.01763
25. Demmler D, Schneider T, Zohner M (2015) Aby-a framework for efficient mixed-protocol secure two-party computation. In: NDSS, San Diego, CA
26. Mohassel P, Rindal P (2018) Aby3: A mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Association for Computing Machinery, New York, NY, USA, pp 35–52. https://doi.org/10.1145/3243734.3243760
27. Gilad-Bachrach R, Dowlin N, Laine K, Lauter K, Naehrig M, Wernsing J (2016) Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. International Conference on Machine Learning, New York, New York, USA 48:201–210
28. Hesamifard E, Takabi H, Ghasemi M (2018) Wright R (2018) Privacy-preserving machine learning as a service. Proceedings on Privacy Enhancing Technologies 3:123–142. https://doi.org/10.1515/popets-2018-0024
29. Boemer F, Cammarota R, Demmler D, Schneider T, Yalame H (2020) Mp2ml: a mixed-protocol machine learning framework for private inference. In: Proceedings of the 15th International Conference on Availability, Reliability and Security, Association for Computing Machinery, New York, NY, USA, pp 1–10. https://doi.org/10.1145/3407023.3407045
30. Juvekar C, Vaikuntanathan V, Chandrakasan A (2018) {GAZELLE}: A low latency framework for secure neural network inference. In: 27th {USENIX} Security Symposium ({USENIX} Security 18), {USENIX} Association, Baltimore, MD, pp 1651–1669. https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar
31. Guo J, Kong Z, Liu C (2020) Poishygiene: Detecting and mitigating poisoning attacks in neural networks. arXiv preprint arXiv:2003.11110
32. Araki T, Furukawa J, Lindell Y, Nof A, Ohara K (2016) High-throughput semi-honest secure three-party computation with an honest majority. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Association for Computing Machinery, New York, NY, USA, pp 805–817. https://doi.org/10.1145/2976749.2978331
33. Kingma D, Ba J (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980
34. Hampel F (1974) The influence curve and its role in robust estimation. J Am Stat Assoc 69(346):383–393. https://doi.org/10.1080/01621459.1974.10482962
35. Canetti R (2000) Security and composition of multiparty cryptographic protocols. J Cryptol 13(1):143–202
36. Canetti R (2001) Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, IEEE, pp 136–145
37. Goldreich O, Micali S, Wigderson A (2019) How to Play Any Mental Game, or a Completeness Theorem for Protocols with Honest Majority, Association for Computing Machinery, New York, NY, USA, p 307-328. https://doi.org/10.1145/3335741.3335755

Congcong Chen received the BE degree in software engineering from Jiangxi Agricultural University, Nanchang, China, in 2018. He is currently a candidate master student of computer Engineering degree at the College of Information Technology, Shanghai Ocean University, China. His main research interests include AI security, privacy preserving, secure multi-party computation, and information security.

Lifei Wei received the BSc and MSc degrees in applied mathematics from the University of Science and Technology Beijing, Beijing, China, in 2005 and 2007, respectively and the PhD degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2013. He is currently a Professor with College of Information Engineering, Shanghai Maritime University, Shanghai, China. He has published papers in major journals, such as IEEE Transactions on Information Forensics and Security (TIFS), IEEE Transactions on Dependable and Secure Computing (TDSC), IEEE Transactions on Services Computing, and Information Sciences. His main research interests include information security, AI security, privacy preserving, blockchain and applied cryptography.



Lei Zhang received the BSc degree in applied mathematics from Yantai Normal University, Yantai, China, in 2005, MSc degrees in computational mathematics and PhD degree in automatic control science and Engineering from the University of Science and Technology Beijing, Beijing, China, in 2007 and 2011, respectively. She is currently an Assistant Professor with the College of Information Technology, Shanghai Ocean University, China. Her main research interests include applied cryptography, big data security and access control.

Ya Peng received the BE degree in computer science and technology from Anhui Agricultural University, Hefei, China, in 2020. She is currently a candidate master student of computer science and technology at the College of Information Technology, Shanghai Ocean University, China. Her main research interests include AI security and information security.

Jianting Ning received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, in 2016. He is currently a Professor with Fujian Provincial Key Laboratory of Network Security and Cryptology, College of Computer and Cyber Security, Fujian Normal University, China. Previously, he was a Research Scientist at the School of Computing and Information Systems, Singapore Management University. He has published papers in major conferences/journals, such as ACM CCS, ASIACRYPT, ESORICS, ACSAC, IEEE Transactions on Information Forensics and Security (TIFS), and IEEE Transactions on Dependable and Secure Computing (TDSC). His research interests include applied cryptography and information security.