Focusing on Pinocchio's Nose: A Gradients Scrutinizer to Thwart Split-Learning Hijacking Attacks Using Intrinsic Attributes

Jiayun Fu*, Xiaojing Ma[1]*, Bin B. Zhu†, Pingyi Hu*, Ruixin Zhao*, Yaru Jia*, Peng Xu*, Hai Jin*, and Dongmei Zhang†
*Huazhong University of Science and Technology, Wuhan, China
†Microsoft Research Asia, Beijing, China
Emails: {fujiayun, lindahust}@hust.edu.cn, binzhu@microsoft.com, {pingyihu, M202071386, yrjia, xupeng, hjin}@hust.edu.cn, dongmeiz@microsoft.com

***Abstract*—Split learning is privacy-preserving distributed learning that has gained momentum recently. It also faces new security challenges. FSHA [37] is a serious threat to split learning. In FSHA, a malicious server hijacks training to trick clients to train the encoder of an autoencoder instead of a classification model. Intermediate results sent to the server by a client are actually latent codes of private training samples, which can be reconstructed with high fidelity from the received codes with the decoder of the autoencoder. SplitGuard [10] is the only existing effective defense against hijacking attacks. It is an active method that injects falsely labeled data to incur abnormal behaviors to detect hijacking attacks. Such injection also incurs an adverse impact on honest training of intended models.**

In this paper, we first show that SplitGuard is vulnerable to an adaptive hijacking attack named SplitSpy. SplitSpy exploits the same property that SplitGuard exploits to detect hijacking attacks. In SplitSpy, a malicious server maintains a shadow model that performs the intended task to detect falsely labeled data and evade SplitGuard. Our experimental evaluation indicates that SplitSpy can effectively evade SplitGuard. Then we propose a novel passive detection method, named Gradients Scrutinizer, which relies on intrinsic differences between gradients from an intended model and those from a malicious model: the expected similarity among gradients of same-label samples differs from the expected similarity among gradients of different-label samples for an intended model, while they are the same for a malicious model. This intrinsic distinguishability enables Gradients Scrutinizer to effectively detect split-learning hijacking attacks without tampering with honest training of intended models. Our extensive evaluation indicates that Gradients Scrutinizer can effectively thwart both known split-learning hijacking attacks and adaptive counterattacks against it.

*Jiayun Fu, Xiaojing Ma, Pingyi Hu, Ruixin Zhao, Yaru Jia, and Peng Xu are with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Hubei Engineering Research Center on Big Data Security, Hubei Key Laboratory of Distributed System Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology. Hai Jin is with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology.

[1]Corresponding author: Xiaojing Ma (lindahust@hust.edu.cn).

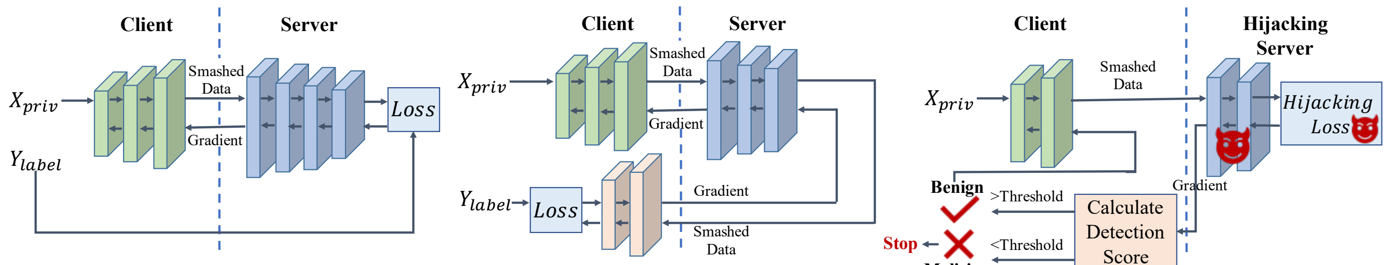
Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24874
www.ndss-symposium.org

I. Introduction

Deep Neural Networks (DNNs) have achieved state-of-the-art performance for many computer vision and other tasks. A well-performing DNN model requires a large number of training samples to train. Privacy concerns and privacy protection laws such as GDPR [40] and HIPAA [3] make collecting a large set of training data a great challenge. To address this problem, distributed deep learning, such as federated learning [7], [22], [26], [30] and split learning [15], [50], has been proposed. In distributed learning, a group of data holders (clients) collaboratively train a DNN model without sharing their private data, facilitated by a server to aggregate model updates in federated learning or train a portion of the DNN network with more powerful computing power in split learning. We focus on the latter in this paper.

In split learning, a deep neural network is split into two or three parts, as shown in Fig.1, depending on whether the labels are shared with the server. In label-sharing split learning, a client computes initial layers of the DNN model and sends outputs along with their labels to the server, while the server computes the remaining layers of the DNN model, calculates loss values with the received labels, and then backward propagates gradients to the client. In label-protected split learning, the client calculates loss values and backward propagates to the server, and the server then backward propagates back to the client to update the first portion of the model.

Split learning faces new security challenges. A malicious server can launch active inference attacks on clients' training data. The *Feature-Space Hijacking Attack* (FSHA) [37] is a recently proposed active attack on split learning. In FSHA, a malicious server, say Bob, hijacks split-learning training to train the client-side model, in a manner like training a *Generative Adversarial Network* (GAN) [14], [20], [42], to approximate the encoder of an autoencoder [19], [24] that Bob trains simultaneously. When a client, say Alice, uses her model to compute outputs and sends them to Bob, she actually sends



(a) Label-sharing Split Learning (b) Label-protected Split Learning (c) Overview of Gradient Scrutinizer

Fig. 1: Two split learning scenarios and an overview of Gradient Scrutinizer. (a) Label-sharing split learning: clients send intermediate outputs along with their labels to the server. (b) Label-protected split learning: clients do not send data labels to the server. (c) An overview of Gradient Scrutinizer.

codes of her private training samples to Bob, who can apply the decoder of the autoencoder to recover the private training samples from the received codes. This active inference attack can deduce private training samples with high fidelity. It is a serious threat to split learning.

To thwart FSHA, SplitGuard [10] exploits different behaviors of hijacked training and honest training on falsely labeled samples, referred to as *fake samples*, to detect training-hijacking attacks on split learning. During training with SplitGuard, a client intermittently feeds intentionally fake samples and compares their gradients, referred to as *fake gradients*, with those of regular training samples received from the server, referred to as *regular gradients*. Fake gradients tend to deviate from regular gradients in honest split-learning training but not so in hijacked split-learning training. This property is exploited by SplitGuard to thwart split-learning hijacking attacks.

In this paper, we first show that SplitGuard is vulnerable to an adaptive hijacking attack, named *SplitSpy*, which exploits the same property exploited by SplitGuard to detect hijacking attacks. In SplitSpy, an adversary server, Bob, maintains a legitimate model that performs the intended task and is trained in a speedup manner. Bob then uses the legitimate model to detect fake samples and to generate deceivable gradients to send to clients, making SplitGuard unable to detect the hijacked training.

Then we propose a novel detection method, named *Gradients Scrutinizer*, which can effectively thwart both FSHA and SplitSpy as well as adaptive counterattacks. Unlike SplitGuard that tampers with split-learning training through injecting fake training samples, Gradients Scrutinizer is a passive detection method that passively scrutinizes gradients received from the server to detect hijacking attacks, and thus incurs no adverse impact on honest training of intended models. It relies on an intrinsic difference between gradients from an intended model and those from autoencoder: the expected similarity of gradients of same-label samples is different from that of gradients of cross-label samples for an intended model, while they are the same for a malicious model. This intrinsic distinguishability enables Gradients Scrutinizer to effectively detect split-learning hijacking attacks. We present experimental evaluation to show Gradients Scrutinizer's effectiveness in detecting FSHA and SplitSpy as well as 4 adaptive counterattacks that aim to enable an autoencoder to produce label-dependent gradients to evade Gradients Scrutinizer.

This paper includes the following major contributions:

- We propose SplitSpy, which can effectively evade SplitGuard, the only existing effective defense against split-learning hijacking attacks to the best of our knowledge.
- We propose Gradients Scrutinizer, a passive detector that can effectively thwart split-learning hijacking attacks.
- We conduct extensive experiments to demonstrate the effectiveness of Gradients Scrutinizer in detecting FSHA, SplitSpy, and 4 adaptive counterattacks.
- We present a theoretical security analysis to demonstrate that Gradients Scrutinizer relies on the intrinsic distinguishability between honest training and hijacked training in thwarting hijacking attacks.

The paper is organized as follows. We briefly review related work in Section II, describe FSHA and SplitGuard in Section III. Our SplitSpy and its performance evaluation against SplitGuard are presented in Section IV. Gradients Scrutinizer is described in Section V, and its performance against FSHA and SplitGuard is extensively evaluated in Section VI. We evaluate the performance of Gradients Scrutinizer against four adaptive counterattacks in Section VII. The paper concludes with Section VIII.

The Gradient Scrutinizer code is available at: https://github.com/CGCL-codes/GradientsScrutinizer.

II. Related Work

A. Split Learning

Split learning [2], [15], [39], [50] is a type of distributed learning that enables multiple clients and a server to jointly train a global model without sharing clients' raw data. The global model is partitioned into multiple portions. Clients and the server each maintain a portion of the model. Clients and the server share only intermediate forward and backward results of the model. Split learning is particularly suitable for scenarios that clients have limited computing resources. In this case, it allows offloading most computational work to the more powerful server with a proper partition of the model.

SplitNN [15], [50] is the first split-learning method. It supports both label-sharing and label-protected split learning, as shown in Fig. 1. In the former case, clients keep the first several layers of the network and share their data labels with the server while the server keeps the remaining layers of the network. In the latter case, clients keep first and last several layers and do not share their data labels with the server while the server keeps the middle portion of the network. SplitNN applies a round-robin training protocol to train multiple clients. Clients agree on a sequential circular list, and interact with the server in turn. When it is the turn of a client, Alice, she performs one or multiple rounds (i.e., steps) of training process with the server, Bob. In each round of training process, Alice feeds her private samples to the first portion of the network, forward propagates through it, and sends outputs of its last layer to Bob. In label-sharing split learning, Alice also sends the labels to Bob. Upon receiving the outputs from Alice, Bob feeds and forward propagates through his portion of the network. In label-sharing split learning, Bob calculates loss values and backward propagates gradients to update his portion of the network. In label-protected split learning, Bob sends outputs of his last layer back to Alice, who then forward propagates through the last portion of the network, calculates loss values, and sends the output gradients of the first layer of the last portion of the network back to Bob, who then backward propagates through his portion of the network. In both cases, Bob sends the output gradients of his first layer back to Alice to backward propagate through the first portion of the network. At the end of her turn, Alice sends her updated client model to the next client in the sequential circular list to repeat the training process with the server as Alice just did.

The sequential training process in SplitNN is inefficient since only a single client is trained with the server each time. Many split-learning methods have been proposed to address this issue [1], [16], [21], [25], [44], [46], [48]. Typical methods include Splitfed [46], Multi-head Split Learning [21], and FedGKT [16]. Splitfed [46] conducts split learning in the same way as federated learning: the server receives forward propagation results from clients in parallel and updates the server-side portion of the network, while clients receive backward propagation gradients from the server and update their local portions of the network in parallel. At the end of some interaction rounds, federated-learning aggregation is applied to clients' models to produce a global clients' model to update all clients' models. This aggregation process is conducted at an aggregation server that is different from the split learning server. MhSP [21] modifies Splitfed by removing the aggregation of clients' models since its authors find out that removing the aggregation process has only a little effect on the model performance. FedGKT [16] is similar to MhSP except that it allows clients to maintain different network architectures while sharing the same server-side model. Unlike MhSP and other split-learning methods, each client in FedGKT trains its client model on its own by adding additional layers, no backward propagated gradients from the server are used. It is straightforward to modify FedGKT to use backward propagated gradients from the server to update clients' models like other split-learning methods. This modified version of FedGKT is referred to as *FedGKT-SP* in this paper.

B. Inference Attacks on Split Learning

The risk of inference attacks in split learning has been studied, typically under the assumption that the server is honest but curious [11], [18], [29], [56], [56]. Most studies focus on the risk that the server infers labels of clients' private data in label-protected split learning. Some also study the risk that the server reconstructs private data of clients in some simple cases. A norm-based label-inference attack on two-party split learning is proposed in [29], which allows the party without labels to accurately recover private ground-truth labels owned by the other party. Zhao et al. [56] exploit the correlation of signs of gradients at the logit layer and gradients of its connected weights to reveal data labels. Liu et al. [31] cluster gradients based on the cosine similarity to determine whether data samples are with the same label. A *Gradient Inversion Attack* (GIA) is proposed in [23] that can uncover private labels with very high accuracy in two-party split learning. It leverages obtained gradients to train a shadow model and optimizes labels to make the shadow model gradients closest to the obtained gradients. Unsplit [11] and also the attack proposed in [18] aim to infer training data by using a shadow model of clients' initial portion of the split model, which is trained alternatively to predict and to approximate the received intermediate results sent by clients.

Many defense methods have also been proposed. Some apply distance correlation to reduce the irrelevant information contained in intermediate results to alleviate the risk of inference [43], [45], [49], [51]. Others defend against inference attacks by adding random noise to intermediate results [9], [13], [33], [38], [41], [47], [53].

An active attack, the *Feature-Space Hijacking Attack* (FSHA) [37], is proposed recently to enable a malicious server in split learning to reconstruct clients' private data. In FSHA, a malicious server, Bob, hijacks the training process to trick clients to train the client-side model to approximate the encoder of an autoencoder that Bob trains simultaneously. The autoencoder is trained with public data that performs the same task as the intended task of split learning but can be very different from clients' private data. Forward propagating results sent by a client to Bob are actually latent codes of private training samples. By applying the decoder of the autoencoder to the received codes, Bob can reconstruct the private training samples with high fidelity. More details are provided in Section III-B.

C. Defense against Training-hijacking Attacks

FSHA is a significant privacy risk for split learning. Researchers have actively searched for solutions. Gawron et al. [12] study the performance of FSHA in split learning enhanced with *Differential Privacy* (DP), in which clients add random noise to received gradients from the server to update the client-side model. They find out that DP can not provide enough protection against FSHA. SplitGuard [10] exploits different behaviors of a benign model and a hijacking model to detect FSHA. More details are provided in Section III-C. To the best of our knowledge, it is the only effective defense against FSHA. Our analysis of SplitGuard reveals that the very same property exploited by SplitGuard to detect FSHA can also be exploited to evade SplitGuard, which leads to SplitSpy. Our Gradients Scrutinizer can thwart both FSHA and SplitSpy.

III. Existing Hijacking Attacks and Defense

A. Threat Model for Hijacking Attacks and Defenses

The threat model for both split-learning hijacking attacks and their defenses including ours is similar to that in [37]. More specifically, the server in split learning, Bob, can be malicious and can alter his task in split learning training to reconstruct private training data and evade detection. Bob has no information on the clients' architecture or its weights but has access to a dataset X_{pub} that captures the same domain of clients' private training datasets. No intersection between X_{pub} and the private datasets is required. On the other hand, clients are all honest and follow the split learning training process.

B. Feature-Space Hijacking Attacks

The *Feature-Space Hijacking Attack* (FSHA) [37] is a powerful active inference attack on split learning. In FSHA, a malicious server, Bob, trains an autoencoder, which comprises an encoder $\tilde{f}$ and a decoder $\tilde{f}^{-1}$, on public dataset X_{pub} by minimizing the reconstruction error. As a result, decoder $\tilde{f}^{-1}$ can invert an output of encoder $\tilde{f}$ with high fidelity. Simultaneously, Bob also trains with clients a discriminator model D, which tries to distinguish clients' output $f(X_{priv})$ on private dataset X_{priv} from the encoder's output $\tilde{f}(X_{pub})$ on public dataset X_{pub}. Specifically, discriminator D is updated at each iteration to minimize the following loss function

$$L_D = \log(1 - D(\tilde{f}(X_{pub}))) + \log(D(f(X_{priv}))) \quad (1)$$

Then Bob directs the client-side model f towards maximizing the discriminator's error rate, i.e., minimizing the following loss function:

$$L_f = \log(1 - D(f(X_{priv}))) \quad (2)$$

This interactive training with clients is actually training a *Generative Adversarial Network* (GAN) [14]. In the end, the output space of the client-side model f and Bob's encoder $\tilde{f}$ are expected to overlap to a great extent, making it possible for $\tilde{f}^{-1}$ to invert clients' outputs.

C. SplitGuard

SplitGuard [10] exploits different behaviors of honest training and hijacked training on fake examples. In SplitGuard, after first few steps of honest training, clients send fake batches with probability P_F to server. A *fake batch* is a batch with a percentage $B_F \in [0, 1]$ of fake samples. Clients put fake gradients to set F and regular gradients to set R. Set R is randomly split into two subsets, R_1 and R_2, where $R = R_1 \cup R_2$. Fake gradients differ from regular gradients in magnitude and directions for honest training. Such differences vanish for hijacked training, and thus split-learning hijacking attacks can be detected.

Formally, let $d(A, B)$ represent the absolute difference between average magnitudes of vectors in sets A and B:

$$d(A, B) = \Big|\frac{1}{|A|}\sum_{a\in A}\|a\| - \frac{1}{|B|}\sum_{b\in B}\|b\|\Big| \quad (3)$$

and let $\theta(A, B)$ be the angle between sums of vectors in A and B:

$$\theta(A, B) = \arccos\left(\frac{\Sigma_A \cdot \Sigma_B}{\|\Sigma_A\| \cdot \|\Sigma_B\|}\right) \quad (4)$$

where

$$\Sigma_A = \sum_{a\in A} a \quad (5)$$

is a sum of vectors in A. SplitGuard collects fake gradients and computes a *SplitGuard* (*SG*) *score* as follows,

$$SG\ Score = Sig(\alpha \cdot s)^{\beta} \in (0, 1) \quad (6)$$

with

$$s = \frac{\theta(F, R) \cdot d(F, R) - \theta(R_1, R_2) \cdot d(R_1, R_2)}{d(F, R) + d(R_1, R_2) + \varepsilon} \quad (7)$$

where ε is a small value to avoid division by zero, α and β are two hyperparameters, and Sig is the sigmoid function. An honest server should have a high SG score. If a SG score is smaller than a threshold, SplitGuard determines that the training is a hijacked training.

Fake samples have an adverse impact on honest split-learning training. Clients know fake samples and do not update the client-side model with fake gradients to avoid their adverse impact on the client-side model. An honest server does not have such information and inevitably applies fake samples to update the server-side model, leading to degraded accuracy. To control the adverse impact on honest split-learning training, fake samples should be controlled to take a small percentage of training data.

IV. Our Attack against SplitGuard

SplitGuard exploits distinctive behaviors of honest training and hijacked training on fake samples to detect hijacking attacks: fake gradients are highly distinctive from regular gradients in honest training but almost indistinguishable from regular gradients in hijacked training. In this section, we present our attack, named *SplitSpy*, that exploits the distinctive behaviors of fake samples to effectively evade SplitGuard. We firstly present SplitSpy for label-sharing split learning and then present it for label-protected split learning.

A. SplitSpy for Label-sharing Split Learning

1) ***Description of SplitSpy for Label-sharing****:* In label-sharing split learning, the server has access to data labels. In addition to the models in FSHA, a malicious server in SplitSpy, Bob, maintains a legitimate model that performs the intended task of split learning with clients. It is used to detect fake samples and generate gradients for detected fake samples to send back to clients to disrupt SplitGuard detection.

To detect fake samples, Bob forward propagates intermediate results received from a client through the legitimate model to produce prediction results and compares with received labels. SplitSpy detects fake samples by exploiting the same property exploited by SplitGuard: fake samples tend to produce larger prediction errors than regular samples. More specifically, for a sample x, Bob applies the legitimate model to produce a probability $P_y(x)$ to predict the received true label y_x and calculates its prediction error:

$$P_{err}(x) = 1 - P_y(x) \quad (8)$$

Samples are sorted in a descending order of their prediction errors. Top samples are likely to be fake samples. SplitSpy removes top λ percentage of samples. The value of λ is related to but can be very different (see Section IV-A2) from the ratio of fake samples used in SplitGuard. Its value also depends on the accuracy of the legitimate model: when the accuracy is low, λ is set high to ensure an enough number of fake samples removed. When the accuracy improves with the progress of training, λ decreases to a fixed value. Note that the ratio of fake samples used in SplitGuard should be sufficiently low to avoid incurring a significant adverse impact on model accuracy of honest training.

Removed samples are not used to train the legitimate model. Their gradients that Bob sends back to clients are from the legitimate model. Survived samples are used to train the legitimate model. Their gradients that Bob sends to clients are from discriminator D, the same as in FSHA, to drive the client-side model to approximate encoder $\tilde{f}$. Since a significant portion of fake gradients a client receives are from the legitimate model, distinctions between fake gradients and regular gradients are retained, and SplitGuard is circumvented.

We also apply an n-to-1 speedup training on the legitimate model to enhance evasion of SplitGuard: the legitimate model is updated n rounds for one round of updates from clients. The value of n should not be too large to avoid training the legitimate model with data of heavily skewed distribution, which may result in lowered accuracy. Our experimental evaluation indicates that the legitimate model maintains good accuracy even for n at value 20.

2) ***Evaluation of SplitSpy for Label-sharing:*** SplitGuard is evaluated on MNIST, FMINST, and Cifar10 in [10]. The first two datasets are too simple. We choose Cifar10, Cifar100, and CelebA to evaluate the effectiveness of SplitSpy against SplitGuard and compare it with FSHA. In our experiments, SplitGuard is applied as described in [10], with the same values of parameters: hyperparameters α and β in SplitGuard are set to 5 and 2, ϵ is set to $1e^{-10}$ to avoid division by zero, and the batch size is set to 64. For SplitSpy, unless stated otherwise, we empirically set n to 20 and λ to 40%, 20%, and 10% when the legitimate model's accuracy is below 15%, between 15% and 30%, and above 30%, respectively, to ensure an enough number of faked samples are removed. The legitimate model's accuracy is estimated with all samples, including fake samples, for each batch, and batches with abnormally lower accuracy are replaced with predicted accuracy based on preceding batches. In removing samples, all the samples in recent batches are used, and the λ percentage of samples with the largest prediction errors are removed.

Figs. 2-4 show SplitGuard's detection results for SplitSpy, FSHA, and honest training on different datasets, with P_F and B_F in SplitGuard set to $\{0.1, 0.2, 0.3\}$ and $\{16/64, 32/64, 64/64\}$, respectively. The results shown in these figures start at the 50th step since SplitGuard requires the honest model to have a certain level of classification capability to produce detectable distinction between fake gradients and regular gradients and does not send fake samples before the 50th step. We do not show results after the 150th step since SplitSpy's SG scores are generally very close to 1 after the 150th step. From the three figures, we can see that FSHA has low scores on all the datasets and thus can be detected. On the other hand, SplitSpy has very high scores in general, lying in the range of honest training, on all the datasets, which means that the SplitSpy attack is indistinguishable from honest training for SplitGuard.

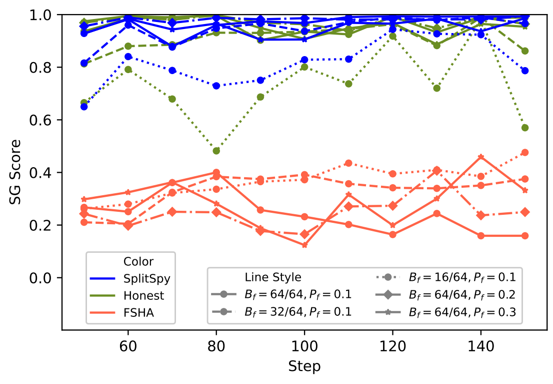


Fig. 2: On Cifar10: SG scores of SplitSpy (in blue), FSHA (in red), and honest training (in green) for $P_F = \{0.1, 0.2, 0.3\}$ and $B_F = \{16/64, 32/64, 64/64\}$ in SplitGuard.

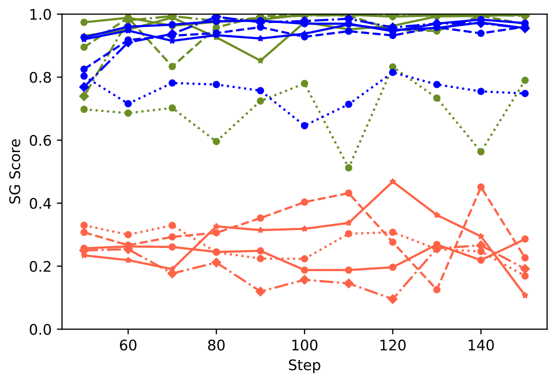


Fig. 3: On Cifar100: SG scores of SplitSpy (in blue), FSHA (in red), and honest training (in green) for $P_F = \{0.1, 0.2, 0.3\}$ and $B_F = \{16/64, 32/64, 64/64\}$ in SplitGuard. The legend inside Fig. 2 is also applied here.

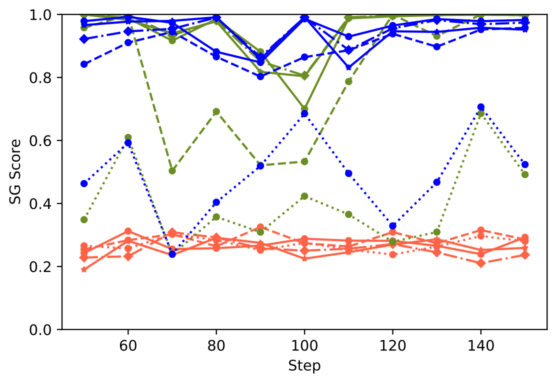


Fig. 4: On CelebA: SG scores of SplitSpy (in blue), FSHA (in red), and honest training (in green) for $P_F = \{0.1, 0.2, 0.3\}$ and $B_F = \{16/64, 32/64, 64/64\}$ in SplitGuard. The legend inside Fig. 2 is also applied here.

SplitGuard processes fake samples at the batch level: if a batch contains fake samples, the batch is considered as a fake batch, with all its samples treated as fake samples and their

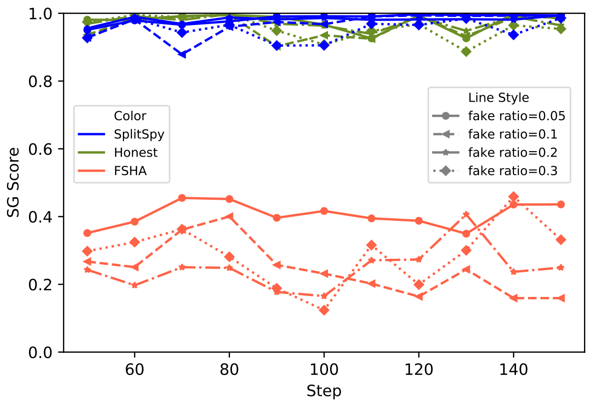


Fig. 5: On Cifar10: SG scores of SplitSpy (in blue), FSHA (in red), and honest training (in green) for 5%, 10%, 20%, and 30% of fake samples in SplitGuard, while λ in SplitSpy is fixed at 10%.

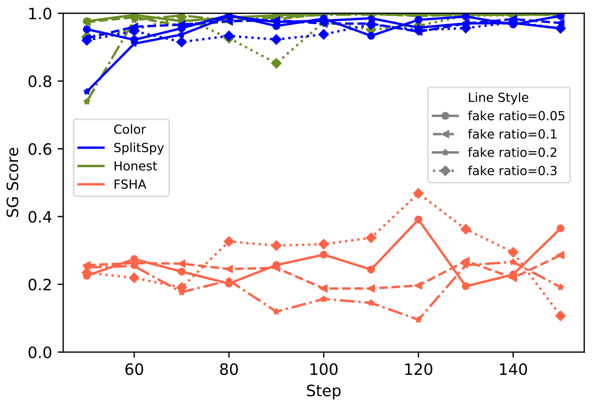


Fig. 6: On Cifar100: SG scores of SplitSpy (in blue), FSHA (in red), and honest training (in green) for 5%, 10%, 20%, and 30% of fake samples in SplitGuard, while λ in SplitSpy is set to 40% during 50-100th step and 20% during 100-150th step.

gradients are put into the set of fake gradients. This batch-level detection reduces the SG score for a benign server when B_F is small, i.e., fake samples take a small percentage of samples in a batch, since most samples in a fake batch are regular samples in this case, yet they are treated as fake samples. We can perceive this effect from Figs. 2-4: the SG scores of honest training and SplitSpy for $B_F = 16/64$ are generally lower than those for $B_F = 32/64$, which in turn are generally lower than those for $B_F = 64/64$. For CelebA's results shown in Fig. 4, some SG scores of honest training even reside in the range of FSHA's SG scores when $B_F = 16/64$.

This shortcoming of SplitGuard can be addressed by the following sample-level detection: regular samples in a fake batch are still considered as regular samples, and their gradients are put into the set of regular gradients instead of the set of fake gradients as in SplitGuard's batch-level detection. Figs. 5-7 show the sample-based detection results on the three datasets with the percentage of fake samples in SplitGuard set to 5%, 10%, 20%, and 30%, while λ in SplitSpy, according to our experimental setting, is fixed at 10% for both Cifar10 and CelebA and is set to 40% from the 50th step to the 100th step and 20% from the 100th step to the 150th step for Cifar100. We can see from these figures that the SG scores of honest training are all high, well above FSHA's SG scores. On the other hand, SplitSpy's SG scores still remain high and are indistinguishable from those of honest training.

We have also evaluated the reconstruction performance of SplitSpy and compared it with FSHA. Fig. 8 shows some reconstructed images by SplitSpy with λ set to 10% and 20% and by FSHA on Cifar10, along with their values of *Mean Squared Error* (MSE), SSIM [52], and LPIPS [55] with the original images. We can see that their reconstructed images of successful attacks are of similarly high fidelity. We can also conclude that SplitSpy's reconstruction capability is robust to different values of λ. On the other hand, a large value of λ requires more training steps to reach the same reconstruction capability than a small value of λ. This is simply because fewer samples are effectively used to train the client-side model to approximate the encoder of a malicious server's autoencoder.

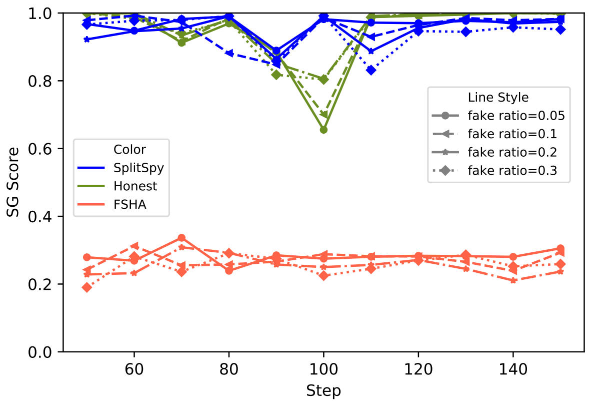
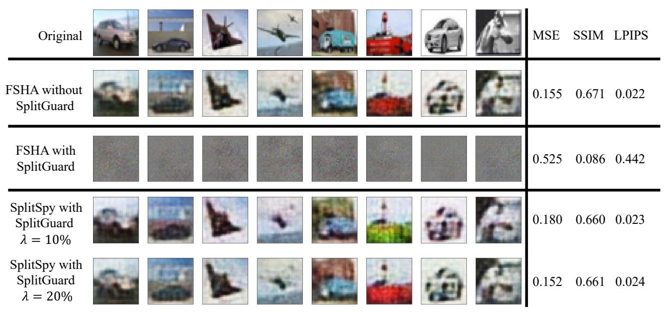


Fig. 7: On CelebA: SG scores of SplitSpy (in blue), FSHA (in red), and honest training (in green) for 5%, 10%, 20%, and 30% of fake samples in SplitGuard, while λ in SplitSpy is fixed at 10%.

From the above results, we conclude that SplitSpy can effectively evade SplitGuard while retaining the similar inference power as FSHA in reconstructing clients' private data.

B. SplitSpy for Label-protected Split Learning

In this case, the server has no access to data labels. This is not a big hurdle in evading SplitGuard detection. As mentioned in Section II-B, many label-inference attacks on split learning have been proposed [23], [29], [31], [56]. Among them, the *Gradient Inversion Attack* (GIA) [23] can infer labels of private data in two-party split learning with high accuracy. In this attack, the party without labels trains a surrogate model by treating data labels as a variable. When it receives a round of data gradients from the party with labels, the malicious party trains the surrogate model many rounds by applying the following two steps alternatively: train parameters of the surrogate model with fixed labels and train label prediction with fixed parameters of the surrogate model. It achieves label-



	MSE	SSIM	LPIPS
Original			
FSHA without SplitGuard	0.155	0.671	0.022
FSHA with SplitGuard	0.525	0.086	0.442
SplitSpy with SplitGuard $\lambda = 10\%$	0.180	0.660	0.023
SplitSpy with SplitGuard $\lambda = 20\%$	0.152	0.661	0.024

Fig. 8: Reconstructed Cifar10 images of FSHA with and without SplitGuard detection and of SplitSpy with λ set to 10% and 20% under SplitGuard detection, along with their MSE, SSIM, and LPIPS values with the original images. Higher SSIM and smaller MSE and LPIPS indicate better reconstruction.

inference accuracy of 97.5% after one training epoch of the split model, when the split model achieves prediction accuracy around 40%. This method can be combined with SplitSpy described in Section IV-A for label-sharing split learning to attack SplitGuard for label-protected split learning: Bob trains a surrogate model with GIA to infer data labels, which are then used to attack SplitGuard with SplitSpy as described in Section IV-A. As a result, SplitSpy can also evade SplitGuard detection for label-protected split learning.

V. Our Defense against Hijacking Attacks

A. Intuition behind Our Defense

SplitGuard detects split-learning hijacking attacks by injecting falsely labeled data into split learning training to incur abnormal behaviors for honest training that hijacked training lacks. This active approach tampers with split-learning training, resulting in degraded accuracy for an honestly trained global model. More critically, the very same property can be exploited by adversaries to detect falsely labeled data and evade SplitGuard detection with an honestly trained server-side model, as our SplitSpy shows.

We advocate that an effective yet hard to evade defense against hijacking attacks should not tamper with split-learning training, such as artificially creating differences between honest training and hijacked training in SplitGuard. Instead, it should rely on intrinsic differences of received gradients between the two types of training. Such a passive approach is more difficult to design but also generally harder to evade than an active approach. An additional benefit of such a passive detection approach is that detection does not incur any accuracy degradation for honest training.

Split-learning hijacking attacks aim to train the client-side model as the encoder of an autoencoder in order to reconstruct private training data. There is a fundamental difference between training a model to perform intended classification task and an encoder: labels are used in the former case but not in the latter case, as we can see from Eqs. 1 and 2 in hijacking attacks. This intrinsic distinction leads to the following observable behavior differences between received gradients produced by an intended classification model and those by a malicious autoencoder model. For a classification model, gradients of samples with the same label tend to be more similar than those of samples with different labels. This intrinsic property of a classification model has been exploited to infer data labels in inference attacks on split learning [31]. On the other hand, gradients from an autoencoder do not have such distinctness between same-label samples and different-label samples. Our Gradients Scrutinizer exploits this intrinsic difference to differentiate hijacked training from honest training.

Discriminator model D in hijacking attacks is trained as the discriminator in a GAN, as described in Section III-B. It is known that training a GAN is less stable than training a classification model [4], [6]. Our Gradients Scrutinizer also exploits this intrinsic difference between the intended model and an attacking model to detect split-learning hijacking attacks.

B. Theoretical Analysis

In this subsection, we provide a theoretical analysis on the intrinsic distinguishability between honest training and hijacked training. It lays the foundation for Gradients Scrutinizer to thwart split-learning hijacking attacks.

We assume that training dataset $\mathbb{X}$ in the split-learning contains n classes, and $\boldsymbol{X}$ denotes a sample in $\mathbb{X}$. Let C_y denote the set of samples with label y, and let $g(\boldsymbol{X})$ represent a received gradient of sample $\boldsymbol{X}$ from the server. Given three random samples $\boldsymbol{X_0}$, $\boldsymbol{X_1}$, and $\boldsymbol{X_2}$ with given labels y_0, y_1, and y_2, respectively, with $y_1 = y_0$ and $y_2 \neq y_0$, we want to calculate the expected difference of same-label and different-label inner products:

$$\begin{aligned}&\mathbb{E}_{\{\boldsymbol{X_0},\boldsymbol{X_1},\boldsymbol{X_2}\}\in\{C_{y_0},C_{y_1},C_{y_2}\}}\left[g(\boldsymbol{X_0})\cdot g(\boldsymbol{X_1}) - g(\boldsymbol{X_0})\cdot g(\boldsymbol{X_2})\right]\\ &\quad= \mathbb{E}_{\boldsymbol{X_0}\in C_{y_0}}\left[g(\boldsymbol{X_0})\cdot \mathbb{E}_{\boldsymbol{X_1}\in C_{y_1},\boldsymbol{X_2}\in C_{y_2}}\left[g(\boldsymbol{X_1}) - g(\boldsymbol{X_2})\right]\right] \qquad (9)\end{aligned}$$

where $\mathbb{E}$ denotes the expectation operation.

Let us consider the inner expectation in Eq. 9, $\mathbb{E}_{\boldsymbol{X_1}\in C_{y_1},\boldsymbol{X_2}\in C_{y_2}}\left[g(\boldsymbol{X_1}) - g(\boldsymbol{X_2})\right]$, which is the expected gradient difference of $\boldsymbol{X_1}$ with label $y_1 = y_0$ and $\boldsymbol{X_2}$ with label $y_2 \neq y_0$ given $\boldsymbol{X_0}$ with label y_0.

Proposition 1. If gradient $g(\boldsymbol{X})$ of $\boldsymbol{X}$ is from a GAN model in Eqs. 1 and 2, then $\mathbb{E}_{\boldsymbol{X_1}\in C_{y_1},\boldsymbol{X_2}\in C_{y_2}}\left[g(\boldsymbol{X_1}) - g(\boldsymbol{X_2})\right] = 0$.

Proof Let $Z_{y_1}^{l-1}$ denote the output of $(l-1)$-th layer from an input with label y_1 before activation. According to the process of back propagation, then the gradient of Z^{l-1} will be:

$$\frac{\partial L_f}{\partial Z^{l-1}} = (W^l)^T \frac{\partial L_f}{\partial Z^l} \odot \sigma'_{l-1}(Z^{l-1}) \qquad (10)$$

where W^l is the parameter of the l-th layer, σ_{l-1} is the activation function for the $(l-1)$-th layer, and $\odot$ is the Hadamard product. Suppose that the network has m layers and splits at s-th layer, the received gradient of s-th layer is

$$\begin{aligned}\frac{\partial L_f}{\partial Z^s} &= [\,(W^{s+1})^T \cdots [\,(W^{m-1})^T\,[\,(W^m)^T \frac{\partial L_f}{\partial Z^m}\\ &\quad \odot\sigma'_{m-1}(Z^{m-1})\,]\cdots \odot \sigma'_{s+1}(Z^{s+1})\,]\odot \sigma'_s(Z^s)\,] \qquad (11)\end{aligned}$$

If gradient $g(\boldsymbol{X})$ is calculated from a GAN model, then as illustrated in Section III-B, the received gradient is calculated with loss function in Eq. 2:

$$L_f = \log(1 - D(f(X_{priv}))) = \log(1 - Sig(Z^m)) \quad (12)$$

where Sig is the sigmoid function and Z^m is the logit layer before the sigmoid function. From Eq. 12, the gradient at the logit layer $\frac{\partial L_f}{\partial Z^m}$ in Eq. 11 is:

$$\begin{aligned}\frac{\partial L_f}{\partial Z^m} &= \frac{1}{1 - Sig(Z^m)}(-\frac{\partial Sig(Z^m)}{\partial Z^m}) \\ &= -\frac{1}{1 - Sig(Z^m)} Sig(Z^m)(1 - Sig(Z^m)) \\ &= -Sig(Z^m) = -\frac{1}{1 + e^{-Z^m}}\end{aligned} \quad (13)$$

Substitute Eq. 13 to Eq. 11, we have

$$\begin{aligned}\frac{\partial L_f}{\partial Z^s} = &-[\,(W^{s+1})^T \cdots [\,(W^{m-1})^T\,[\,(W^m)^T \frac{1}{1 + e^{-Z^m}} \\ &\odot \sigma'_{m-1}(Z^{m-1})\,] \cdots \odot \sigma'_{s+1}(Z^{s+1})\,] \odot \sigma'_s(Z^s)\,]\end{aligned} \quad (14)$$

From Eq. 14, we can see that the gradient does not depend on any label, and its expectation under different labels should be the same, i.e., $\mathbb{E}_{y_1}\left[\frac{\partial L_f}{\partial Z^s}\right] = \mathbb{E}_{y_2}\left[\frac{\partial L_f}{\partial Z^s}\right]$. Thus we have $\mathbb{E}_{\boldsymbol{X_1} \in C_{y_1}}\left[\frac{\partial L_f(\boldsymbol{X_1})}{\partial Z^s}\right] = \mathbb{E}_{\boldsymbol{X_2} \in C_{y_2}}\left[\frac{\partial L_f(\boldsymbol{X_2})}{\partial Z^s}\right]$, and

$$\begin{aligned}&\mathbb{E}_{\boldsymbol{X_1} \in C_{y_1}, \boldsymbol{X_2} \in C_{y_2}}[g(\boldsymbol{X_1}) - g(\boldsymbol{X_2})] = \\ &\mathbb{E}_{\boldsymbol{X_1} \in C_{y_1}}\left[\frac{\partial L_f(\boldsymbol{X_1})}{\partial Z^s}\right] - \mathbb{E}_{\boldsymbol{X_2} \in C_{y_2}}\left[\frac{\partial L_f(\boldsymbol{X_2})}{\partial Z^s}\right] = 0\end{aligned} \quad (15)$$

Therefore Proposition 1 holds. □

Proposition 2. If gradient $g(\boldsymbol{X})$ of $\boldsymbol{X}$ is from a classification model, then $\mathbb{E}_{\boldsymbol{X_1} \in C_{y_1}, \boldsymbol{X_2} \in C_{y_2}}[g(\boldsymbol{X_1}) - g(\boldsymbol{X_2})] = \Theta \neq 0$, where Θ is a parameter depending on the model and training data.

Proof In normal classification training process, model learns the mapping from $X \rightarrow label$. As stated in [54] and [31], in a well learned classification model, latent features of samples from different labels are discriminative. When training a classification model, these features become increasingly distinguishable in general. In other words, during the training process, the expected value of the Euclidean distance between feature vector Z_{y_1} of a sample with label y_1 and feature vector Z_{y_2} of a sample with label y_2, denoted as $\mathbb{E}[\|Z_{y_1} - Z_{y_2}\|_2]$ will get smaller when $y_1 = y_2$ and get larger when $y_1 \neq y_2$.

In this case, if we assume the opposite, i.e., $\mathbb{E}_{\boldsymbol{X_1} \in C_{y_1}}[g(\boldsymbol{X_1})] = \mathbb{E}_{\boldsymbol{X_2} \in C_{y_2}}[g(\boldsymbol{X_2})]$. Let $\bar{Z}_{y_1}$ and $\bar{Z}_{y_2}$ denote the centroids of samples with label y_1 and y_2 in the feature space, respectively. After updating with gradients $\mathbb{E}_{\boldsymbol{X_1} \in C_{y_1}}[g(\boldsymbol{X_1})]$ and $\mathbb{E}_{\boldsymbol{X_2} \in C_{y_2}}[g(\boldsymbol{X_2})]$, the centroids will move to $\bar{Z}'_{y_1}$, and $\bar{Z}'_{y_2}$, where $\bar{Z}'_{y_1} = \bar{Z}_{y_1} + \Delta_{y_1}$ and $\bar{Z}'_{y_2} = \bar{Z}_{y_2} + \Delta_{y_2}$. In one step, the update $\Delta_{y_1} = -\mathbb{E}_{\boldsymbol{X_1} \in C_{y_1}}[g(\boldsymbol{X_1})] \times lr$ and $\Delta_{y_2} = -\mathbb{E}_{\boldsymbol{X_2} \in C_{y_2}}[g(\boldsymbol{X_2})] \times lr$, where lr is the learning rate.

As $\mathbb{E}_{\boldsymbol{X_1} \in C_{y_1}}[g(\boldsymbol{X_1})] = \mathbb{E}_{\boldsymbol{X_2} \in C_{y_2}}[g(\boldsymbol{X_2})]$ and $\Delta_{y_1} = \Delta_{y_2}$, the Euclidean distance between the two centroids does not change with the training, which means that the model cannot learn the classification capability to distinguish samples of label y_1 from samples of label $y_2 \neq y_1$. This is in contradiction to the fact that the classification model learns the classification capability to recognize the label of a sample. So in normal classification training, $\mathbb{E}_{\boldsymbol{X_1} \in C_{y_1}, \boldsymbol{X_2} \in C_{y_2}}[g(\boldsymbol{X_1}) - g(\boldsymbol{X_2})] = \Theta \neq 0$, where Θ is a parameter depending on the model and training data. □

Based on **Proposition 1** and **Proposition 2**, we have the following proposition, which certifies existence of intrinsic distinguishability between honest training and hijacked training.

Proposition 3. $\mathbb{E}_{\{\boldsymbol{X_0}, \boldsymbol{X_1}, \boldsymbol{X_2}\} \in \{C_{y_0}, C_{y_1}, C_{y_2}\}}[g(\boldsymbol{X_0}) \cdot g(\boldsymbol{X_1}) - g(\boldsymbol{X_0}) \cdot g(\boldsymbol{X_2})]$ is 0 for GAN but a non-zero value that depends on the model and training data for a classification model, where $y_1 = y_0$ and $y_2 \neq y_0$.

C. Gradients Scrutinizer

We adopt cosine similarity to measure similarity between two gradients. For two vectors v_1 and v_2, cosine similarity, denoted as cos in this paper, is defined as follows

$$\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\| \cdot \|v_2\|} \quad (16)$$

We use a combination of three measures, set gap, fitting error, and overlapping ratio, to differentiate hijacking servers from honest servers. These measures are described next.

*1) **Set Gap**:* At each training step i, a client maintains two sets of cosine similarity values: $S_s(i)$ for cosine similarity of gradients with the same label and $S_d(i)$ for cosine similarity of gradients of different labels. When a client, Alice, receives a batch of gradients from the server, she classifies them according to their labels. Then she calculates cosine similarity for each pair of gradients with the same label and puts the resulting value into set $S_s(i)$. She also computes cosine similarity for each pair of gradients with different labels and put the resulting value into set $S_d(i)$. After computing cosine similarity, Alice calculates a gap score G_i between the two sets for the step:

$$G_i = \overline{S_s(i)} - \overline{S_d(i)} \quad (17)$$

where

$$\overline{S} = \frac{1}{\|S\|}\sum_{x \in S} x \quad (18)$$

is the average of set S, and $\|S\|$ is the cardinality of S. Since the cosine similarity of same-label gradients has a higher value than that of different labels, and we have observed that cosine similarity of different labels is all non-negative, the gap score in Eq. 17 is a positive value, and there is no need to use the absolute value operator in Eq. 18. It is a measure of the distance between averaged same-label cosine similarity and averaged different-label cosine similarity for the current training step. Honest training should have a larger gap score than hijacked training.

*2) **Fitting Error**:* We use a low-order polynomial to fit the average cosine similarity for each set of $S_s(i)$ and $S_d(i)$ along different training steps. A more stable training process should produce smaller fitting errors. In our detection, we use a quadratic function to fit the average of each set of $S_s(i)$ and $S_d(i)$ along with steps:

$$\begin{aligned} P_s(i) = &\quad a_s \cdot i^2 + b_s \cdot i + c_s \\ P_d(i) = &\quad a_d \cdot i^2 + b_d \cdot i + c_d \end{aligned} \tag{19}$$

We use the average of the normalized L_2 errors of both sets $S_s(i)$ and $S_d(i)$ as the fitting error E_n at n-th step:

$$E_n = \frac{1}{2}\left(\sqrt{\frac{\sum_{i=1}^{n}(P_s(i) - \overline{S_s(i)})^2}{n}} + \sqrt{\frac{\sum_{i=1}^{n}(P_d(i) - \overline{S_d(i)})^2}{n}}\right) \tag{20}$$

*3) **Overlapping Ratio**:* $S_s(i)$ and $S_d(i)$ tend to be separated for honest training and overlapped for hijacked training. For a set S, we treat samples falling into the top and bottom γ percentiles as potential outliers, remove them, and calculate the range of the values remained in the set, denoted as $R_\gamma(S)$. The overlapping ratio at step i is defined as the *Intersection over Union* (IoU) [35] of the ranges of $S_s(i)$ and $S_d(i)$:

$$V_i = \frac{|R_\gamma(S_i^s) \cap R_\gamma(S_i^d)|}{|R_\gamma(S_i^s) \cup R_\gamma(S_i^d)|} \tag{21}$$

*4) **Detection Score**:* We combine the three measures to get a normalized detection score $DS_n \in [0,1]$ at n-th step:

$$DS_n = Sig(\lambda_{ds}(\hat{G_n} \cdot \hat{E_n} \cdot \hat{V_n} - \alpha)) \tag{22}$$

where $\hat{G_n} = (G_n)^{\lambda_g}$, $\hat{E_n} = -log(\lambda_e E_n + \varepsilon_e)$, $\hat{V_n} = -log(\lambda_v V_n + \varepsilon_v)$, λ_{ds}, α, λ_g, λ_e, ε_e, λ_v, and ε_v are hyperparameters, and Sig is the sigmoid function that normalizes the detection score to the range of $[0,1]$.

Hyperparameters λ_g, λ_e, and λ_v control the sensitivity of G_n, E_n, and V_n, respectively, in the detection score, and λ_{ds} controls the overall sensitivity of G_n, E_n, and V_n, α is a bias to adjust the range of the detection score to cover a large range in $[0,1]$, and $\varepsilon_e, \varepsilon_v \in (0,1]$ control the ranges of $\hat{E_n}$ and $\hat{V_n}$: $\hat{E_n} \leq -log(\varepsilon_e)$ and $\hat{V_n} \leq -log(\varepsilon_v)$.

If the detection score is less than a threshold, T_{thres}, we determine that the training is a hijacked training. Otherwise, it is an honest training. Fig. 1(c) shows an overview of our defense. When a client, Alice, detects hijacked training, she stops training and informs other clients immediately.

VI. Experimental Evaluation

A. Experimental Setup

We evaluate the performance of our proposed defense on several popular image classification datasets: MNIST [28], Cifar10 [27], CelebA [32], and ImageNet [8]. FSHA is from its published code [36]. In our study, we follow the experimental settings that the authors of FSHA use in evaluating FSHA in [37] for both FSHA and SplitSpy. Specifically, the training set of a dataset is used as clients' dataset X_{priv}, and the validation set is used as X_{pub} in FSHA to train the autoencoder. We use the same ResNet network [17] used in [37] for classification and split it to two parts as client and server in the same way as in [37] (see Appendix A for details). We use the same training settings as in [36] unless stated otherwise: models are trained with Adam optimizer, the batch size is set to 64, and the learning rate is set to 10^{-4} for training discriminator D and 10^{-5} for training all other models.

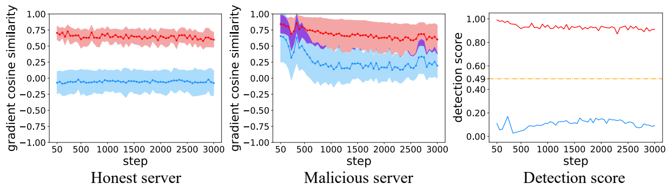


Fig. 9: Gradients Scrutinizer's detection performance on FMNIST. FMNIST is used to determine our hyperparameters (see the caption of Fig. 10 for more information).

We use toy dataset FMNIST to determine the hyperparameters of Gradients Scrutinizer on the hardest case, as shown in Fig 9, among all the attacks presented in this paper, including the adaptive attacks to be described in Section VII. We first determine the hyperparameters in the detection score given in Eq. 22 by requiring similar contributions of the gap score and the fitting error to the detection score and tolerance of a small overlapping ratio. Then we determine threshold T_{thres} in a similar way as *Support Vector Machine* (SVM) determines its decision boundary. The resulting hyperparameter values are: $\lambda_{ds} = 6$, $\lambda_g = 0.8$, $\lambda_e = 9$, $\lambda_v = 0.1$, $\epsilon_e = e^{-3}$, $\epsilon_v = e^{-1}$, and $\alpha = 0.8$. Threshold T_{thres} is set to 0.49. These values are used on our experimental datasets.

Like SplitGuard [10], we start detection at the 50th step and make a detection for each subsequent step. As a result, we have at least 50 data points to fit the quadratic function in Eq. 19 at each detection step. We use a sliding window of 10 steps to make a decision in our experiments.

B. Detecting FSHA with SplitNN

The basic split-learning method, SplitNN [15], [50], uses a round-robin training protocol to train clients. The performance of Gradients Scrutinizer against FSHA on different datasets is shown in Fig. 10. The first two columns show the results of honest training and hijacked training, respectively, and the last column is their detection scores. We can see from the figure that honest training and hijacked training are highly distinctive, and Gradients Scrutinizer can detect a hijacked training within the first 100 training steps, more precisely at the 59th step when the first sliding window is applied to make a decision, for all the datasets.

Figs. 11-14 show reconstructed images of different datasets when FSHA is detected by Gradients Scrutinizer and when no detection is used, along with wFSHA, an enhanced version of FSHA to be described in Section VI-D. Their values of *Mean Squared Error* (MSE), SSIM [52], and LPIPS [55] with the original images are also shown in the figures. We can see that FSHA is detected well before it can infer any meaningful information about private training data. In Appendix B, we show reconstructed images of FSHA when Differential Privacy (DP) is used and conclude that DP is ineffective in thwarting FSHA, which agrees with the conclusion in [12].

To show how each of the three measures that Gradients Scrutinizer uses performs, Fig. 15 shows the experimental

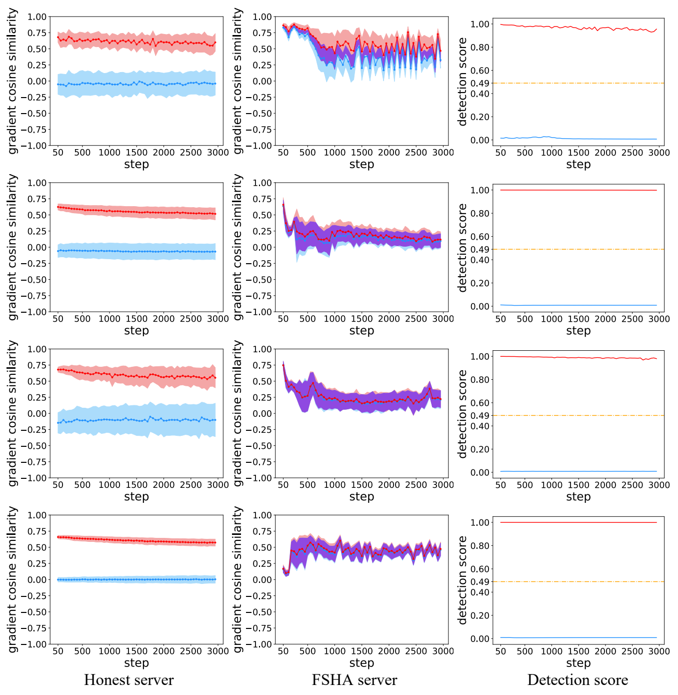
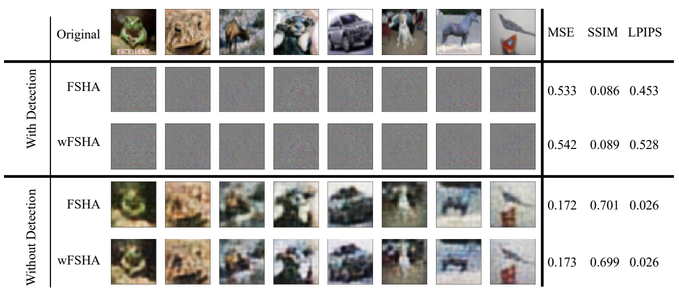


Fig. 10: Gradients Scrutinizer's detection performance with SplitNN on MNIST, Cifar10, CelebA, and ImageNet (from top to bottom). The left two columns show the average (solid line) and the $\pm\xi$ range of cosine similarity (ξ is the standard deviation) for same-label gradients (red) and different-label gradients (blue), with their overlapped region in purple. The right column shows the detection scores of honest training (red) and hijacked training (blue) and the threshold (yellow).



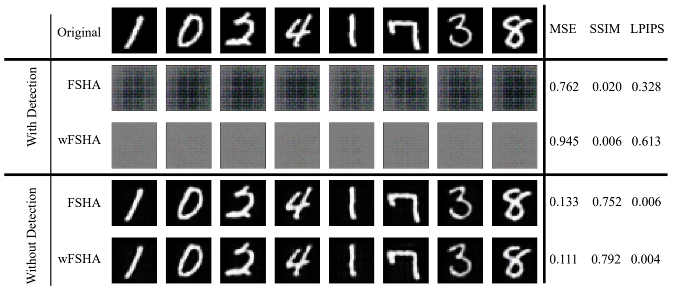
		MSE	SSIM	LPIPS
With Detection	FSHA	0.533	0.086	0.453
	wFSHA	0.542	0.089	0.528
Without Detection	FSHA	0.172	0.701	0.026
	wFSHA	0.173	0.699	0.026

Fig. 11: Reconstructed Cifar10 images of FSHA and wFSHA when detected by Gradients Scrutinizer (middle section) and when no detection is applied (bottom section). Higher SSIM and smaller MSE and LPIPS indicate better reconstruction.

results of the three measures on Cifar10. We can see that all of them are highly effective in detecting FSHA.

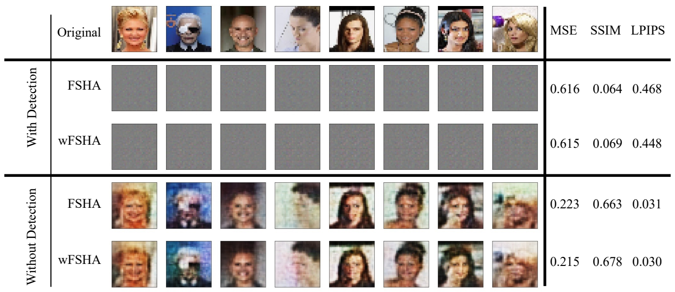
C. Detecting FSHA with More Split Learning Methods

We have also evaluated the performance of Gradients Scrutinizer with other split learning methods described in Section II-A: Splitfed [46], MhSP [21], and FedGKT-SP, a variant of FedGKT [16] for split learning. In our experiments, we assume 200 clients randomly communicating with the server for MhSP and FedGKT-SP. For Splitfed, we assume



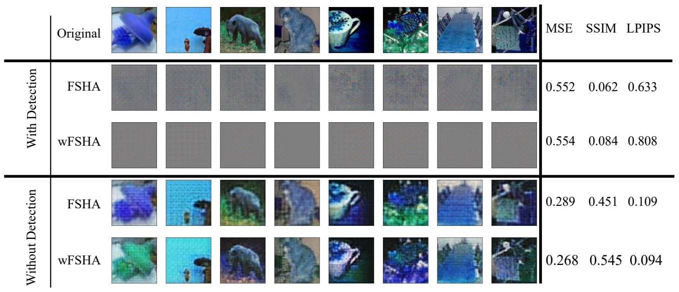
		MSE	SSIM	LPIPS
With Detection	FSHA	0.762	0.020	0.328
	wFSHA	0.945	0.006	0.613
Without Detection	FSHA	0.133	0.752	0.006
	wFSHA	0.111	0.792	0.004

Fig. 12: Reconstructed MNIST images of FSHA and wFSHA when detected by Gradients Scrutinizer (middle section) and when no detection is applied (bottom section).



		MSE	SSIM	LPIPS
With Detection	FSHA	0.616	0.064	0.468
	wFSHA	0.615	0.069	0.448
Without Detection	FSHA	0.223	0.663	0.031
	wFSHA	0.215	0.678	0.030

Fig. 13: Reconstructed CelebA images of FSHA and wFSHA when detected by Gradients Scrutinizer (middle section) and when no detection is applied (bottom section).



		MSE	SSIM	LPIPS
With Detection	FSHA	0.552	0.062	0.633
	wFSHA	0.554	0.084	0.808
Without Detection	FSHA	0.289	0.451	0.109
	wFSHA	0.268	0.545	0.094

Fig. 14: Reconstructed ImageNet images of FSHA and wFSHA when detected by Gradients Scrutinizer (middle section) and when no detection is applied (bottom section).

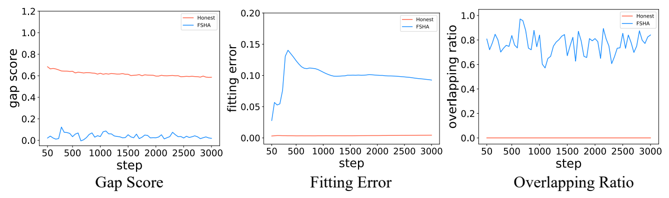


Fig. 15: The performance of the three measures on Cifar10. Honest training is in red, FSHA is in blue.

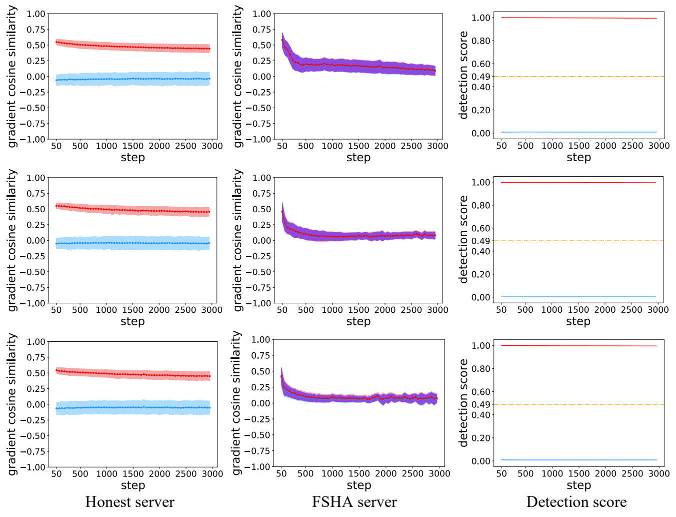


Fig. 16: Gradients Scrutinizer's detection performance on Cifar10 with Splitfed (top), FedGKT-SP (middle), and MhSL (bottom). See the caption of Fig. 10 for more information.

there are 100 clients, and client model aggregation occurs when each client has trained 10 steps with the server. At each aggregation, 10 client models are randomly selected to aggregate, and the aggregated model is sent to all clients to use subsequently. Fig. 16 shows the experimental results with these split methods on Cifar10. We can see that Gradients Scrutinizer has similar detection performance as with SplitNN.

D. Detecting SplitSpy and Enhanced FSHA

We have evaluated the performance of Gradients Scrutinizer against SplitSpy. Fig. 17 shows the detection results with SplitNN on Cifar10 when λ in SplitSpy is set to 10%, 20%, and 30%. We can see that Gradients Scrutinizer can effectively detect SplitSpy.

The discriminator in FSHA is trained with Eqs. 1 and 2. Such training is less stable than using Wasserstein loss [6],

$$L_{DW} = D(f(X_{priv})) - D(\tilde{f}(X_{pub})), \tag{23}$$

to train the discriminator and

$$L_{fW} = -D(f(X_{priv})) \tag{24}$$

to train the client-side model. According to [6], to make GAN more stable, weights of the discriminator should be kept in a compact range such as clamped into a fixed box, and training should avoid using momentum optimizer such as Adam optimizer. A more stable GAN generates less training variations and thus enhances the power of evading Gradients Scrutinizer. This Wasserstein loss-based FSHA is referred to as *wFSHA*. We have evaluated the detection performance of Gradients Scrutinizer against wFSHA. In our experiments, RMSprop [5] is used as the optimizer, and discriminator weights are clamped into $[-0.01, 0.01]$. The experimental results with SplitNN on Cifar10 are shown in Fig. 18 and reconstructed images are shown in Fig. 11. We can see that Gradients Scrutinizer is still effective in thwarting wFSHA.

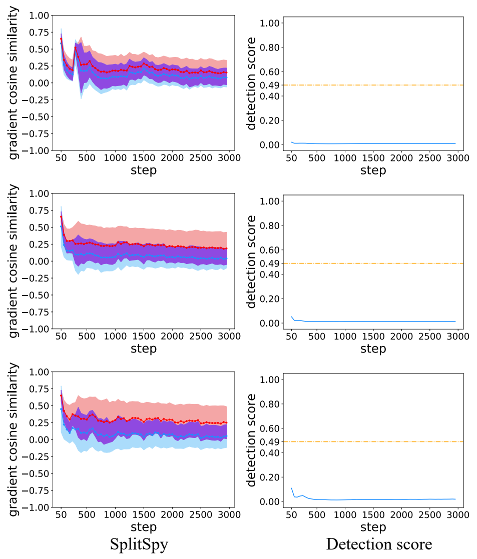


Fig. 17: Gradients Scrutinizer's detection performance against SplitSpy with SplitNN on Cifar10, with λ in SplitSpy set to 10% (top), 20% (middle), 30% (bottom). The left column shows the average (solid line) and the $\pm\xi$ range of cosine similarity (ξ is the standard deviation) for same-label gradients (red) and different-label gradients (blue), with their overlapped region in purple. The right column shows the detection score (blue) and the threshold (yellow).

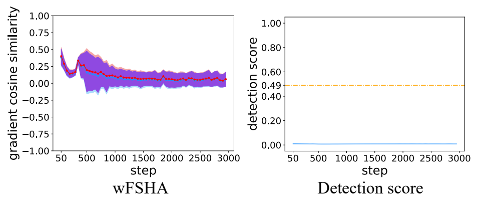


Fig. 18: Gradients Scrutinizer's detection performance against wFSHA with SplitNN on Cifar10. See the caption of Fig. 17 for more information.

E. Impact of Different Settings and Data Distributions

The aforementioned experimental results are based on the default settings and the original data distributions of the given datasets. In this subsection, we study the detection performance of our proposed defense under different settings and data distributions.

1) ***Impact of Batch Size:*** Gradients Scrutinizer relies on statistical measures of the same-label cosine similarity set and the different-label cosine similarity set at each step, such as computing set gap and overlapping ratio. A small batch size makes these measures less reliable, potentially leading to degraded performance of Gradients Scrutinizer. We have studied the impact of different batch sizes on Gradients Scrutinizer's detection performance. Fig. 19 shows the experimental results when the batch size is 32 and 16, smaller than the default batch size 64. Comparing the results for batch size 16 and 32 shown in Fig. 19 with those of default batch size 64 shown in Fig. 10, we can see that for batch size 16 the cosine similarity fluctuates more and the detection performance is degraded, but there is

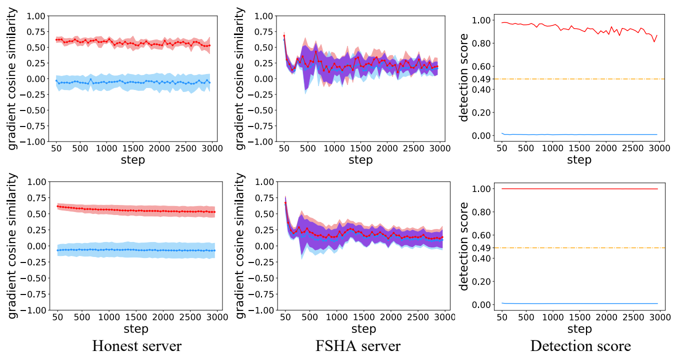


Fig. 19: Gradients Scrutinizer's detection performance on Cifar10 with batch size = 16 (top) and 32 (bottom). See the caption of Fig. 10 for more information.

no obvious difference for batch size 32 and 64. We conclude that Gradients Scrutinizer can still detect FSHA effectively for the two batch sizes.

2) ***Impact of Uneven Dataset:*** Uneven data may increase variations of gradients generated by a classification model, leading to degraded detection performance of Gradients Scrutinizer. CelebA is an uneven dataset. We have evaluated Gradients Scrutinizer with more uneven datasets. Fig. 20 shows the experimental results on uneven MNIST and Cifar10 sampled with Dirichlet distribution $\alpha = 1$. Comparing Fig. 20 with Fig. 10, we can see that the cosine similarity fluctuates more, with less separation between same-label cosine similarity and different-label cosine similarity, for both uneven datasets, but the detection performance is degraded only on the uneven MNIST. There is no obvious degradation for the detection performance (i.e., the detection score) on the uneven Cifar10. We conclude that Gradients Scrutinizer is still effective for uneven datasets.

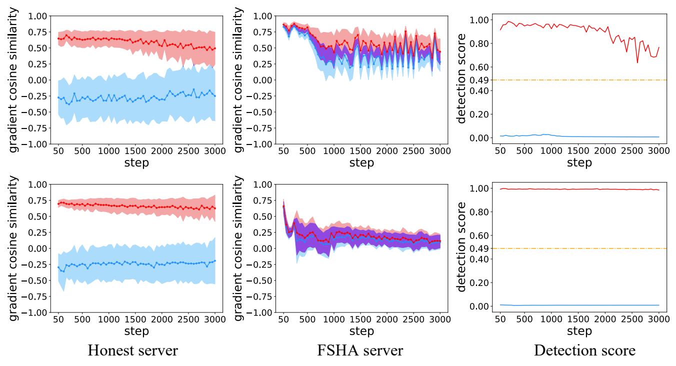


Fig. 20: Gradients Scrutinizer's detection performance against the attack on different uneven MNIST (top) and Cifar10 (bottom), both are sampled with Dirichlet distribution $\alpha = 1$. See the caption of Fig. 10 for more information.

3) ***Impact of Split Layer:*** In split learning, a neural network can be split at different layers. To evaluate its impact on the detection performance of Gradients Scrutinizer, we have used the same four different split layers as in [37] in our experiments. The detail of the split layers is provided in Appendix A. The experimental results with three distinct split layers different from the default one are shown in Fig. 21. We can see that Gradients Scrutinizer is robust to split layer positions.

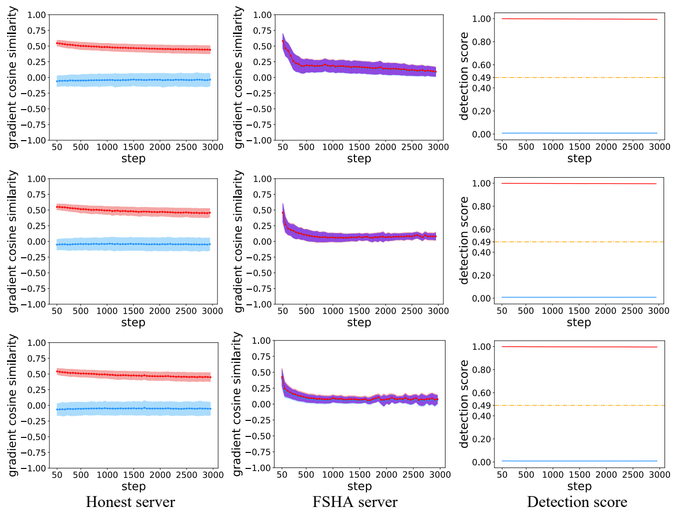


Fig. 21: Gradients Scrutinizer's detection performance on Cifar10 with three different positions of the split layer. See the caption of Fig. 10 for more information.

4) ***Impact of Three Measures:*** Our detection score is a combination of the three measures: gap score, fitting error, and overlapping ratio. In some scenarios, as shown in Fig. 15, each measure is able to distinguish FSHA training from honest training. In such a case, any full or partial combination of the three measures can be chosen for detection. However, for some worst cases such as shown in Fig. 22, a measure may be ineffective to detect. In those conditions, our Gradients Scrutinizer still works as it takes all the three measures into consideration. This ensemble detection makes it much harder for hijacking attacks to evade.

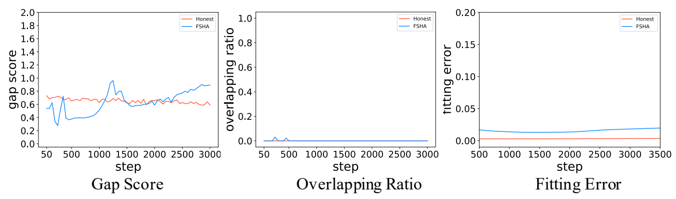


Fig. 22: The performance of the three measures in their worst case. Gap score and overlapping ratio: label-aware GAN training (see Section VII for details) on MNIST. Fitting error: wFSHA training on Cifar10. Honest training is in red, hijacked training is in blue.

VII. Adaptive Counterattacks

We have studied Gradients Scrutinizer's detection performance against known split-learning hijacking attacks. What potential counterattacks adversaries can develop against Gradients Scrutinizer if they have full knowledge of our defense? We study adaptive counterattacks in this section.

Our defense relies on the intrinsic difference between a benign model that trains with labels and an attack model that does not use labels. An adversary may attempt to inject labels into the learning process of a malicious model so that the resulting malicious model has learned the capability to send label-dependent gradients to clients to blur the difference with a benign model. This can be potentially done by adding gradients from a classification model to the gradients sent to clients, training a discriminator model that generates label-dependent gradients, alternatively sending gradients from a classification model, or using conditional GAN. These four adaptive counterattacks on Gradients Scrutinizer will be described and evaluated in following subsections. The evaluation will show that none of them can evade Gradients Scrutinizer.

For the experiments reported in this section, we use Wasserstein loss and the RMSprop optimizer as in wFSHA, described in Section VI-D, since it is more stable than the original GAN used in FSHA and thus should be a more powerful threat to Gradients Scrutinizer, although the detection results on wFSHA, reported in VI-D, indicate that it does not improve much the attack power against Gradients Scrutinizer.

A. *Attack with Mixed Gradients*

An adversary can follow the split learning protocol to train a server-side classification model C like a benign server, and mix its gradients with gradients generated from the discriminator to send to clients:

$$Gradients^T = \gamma \cdot Gradients^C + (1-\gamma) \cdot Gradients^D \quad (25)$$

where $Gradients^T$ is the total gradients sent to a client, $Gradients^D$ is the gradients from discriminator D, $Gradients^C$ is the gradients from classification model C, and γ is a weight of the gradients from classification model C in the total gradients sent to a client.

Gradients from classification model C interfere with our defense as well as with the reconstruction capability of a hijacking attack. When γ increases from 0 to 1, the classification model contributes more and more to the gradients sent to a client, and we can expect that it becomes harder and harder for our defense to detect a malicious server, and at the same time, the adversary loses more and more power to reconstruct clients' training data. When $\gamma = 0$, this attack is the same as FSHA, and Gradients Scrutinizer can effectively detect it. When $\gamma = 1$, the malicious server behaves identically to a benign server to train a benign global model, and there is no hijacking attack that needs to be detected. What about an intermediate value for γ?

We have evaluated the attack with different values of γ. Fig. 23 shows the detection results when $\gamma = 0.8$. Figs. 24-27 show the reconstructed images of the 4 datasets at this γ value for both when hijacking is detected and when there is no detection (i.e., the eventual reconstruction results the attack can achieve). We can see from the figures that Gradients Scrutinizer can still detect the hijacking attack quickly, well before it can infer any meaningful information of private training data.

We can make the following two observations. The first is that the reconstructed images without our defense shown in Figs. 24-27 are degraded as compared with those of FSHA shown in Figs 11-14 but still reasonably good. The second is that, as shown in Fig. 23, the initial behavior of the attack is similar to that of an honest server for both MNIST and

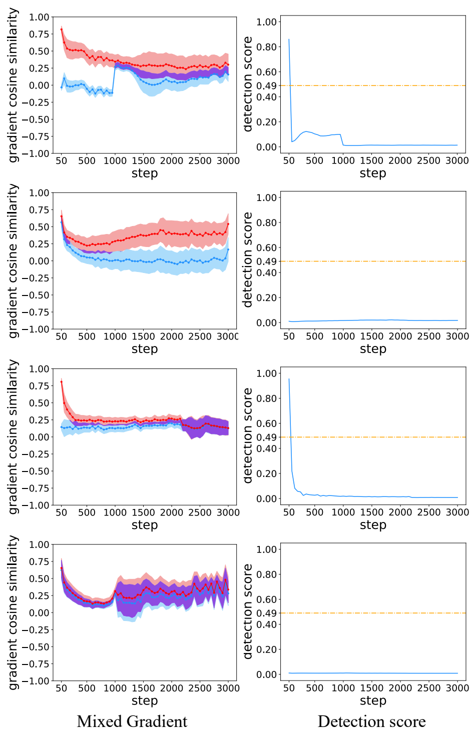


Fig. 23: Gradients Scrutinizer's detection performance with the attack of mixed gradients ($\gamma = 0.8$). From the top to bottom: MNIST, Cifar10, CelebA, and ImageNet. See the caption of Fig. 17 for more information.

CelebA, and then it changes to a typical behavior of FSHA. This can be explained that the classification model learns faster than the discriminator, and thus absolute values of its gradients decrease much faster than those of the discriminator. Eventually, gradients sent to clients are dominated by those from the discriminator, and the impact of the classification model is gradually diminished.

B. *Attack Using Label-aware GAN*

Instead of artificially mixing gradients from a classification model trained by following the split learning protocol, an adversary can intrinsically introduce label-dependent behaviors into the discriminator and the encoder, which can also potentially interfere Gradients Scrutinizer's detection. We evaluate the impact of this attack on our defense.

We can introduce label-dependence to both the discriminator and the encoder to make each of them produce label-dependent results. First, we train the encoder of an autoencoder with the intended task (classification task), while the decoder is still trained like before to minimize the reconstruction error. In this way, the encoder behaves like a classification model.

In FSHA, the discriminator produces a scalar value $D(f(X))$ that does not depend on any label. To make the discriminator label-independent, we use a discriminator that produces a N-dimensional vector $D_N(f(X))$, where N is the number of distinct labels of the intended classification task. We assign a larger weight to the component associated with the label of the data and take the mean of all components as

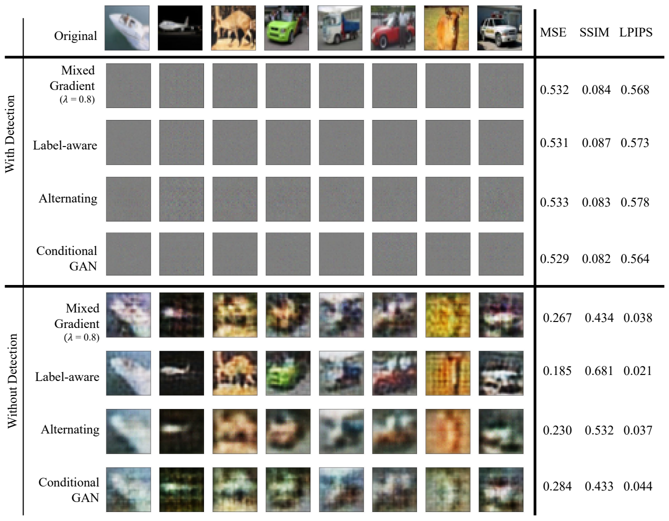

Original
MSE SSIM LPIPS
With Detection
Mixed Gradient (λ = 0.8) 0.532 0.084 0.568
Label-aware 0.531 0.087 0.573
Alternating 0.533 0.083 0.578
Conditional GAN 0.529 0.082 0.564
Without Detection
Mixed Gradient (λ = 0.8) 0.267 0.434 0.038
Label-aware 0.185 0.681 0.021
Alternating 0.230 0.532 0.037
Conditional GAN 0.284 0.433 0.044

Fig. 24: Reconstructed images of different split-learning hijacking attacks on Cifar10: when they are detected by Gradients Scrutinizer (middle section) and when no detection is applied (bottom section). Higher SSIM and smaller MSE and LPIPS indicate better reconstruction.

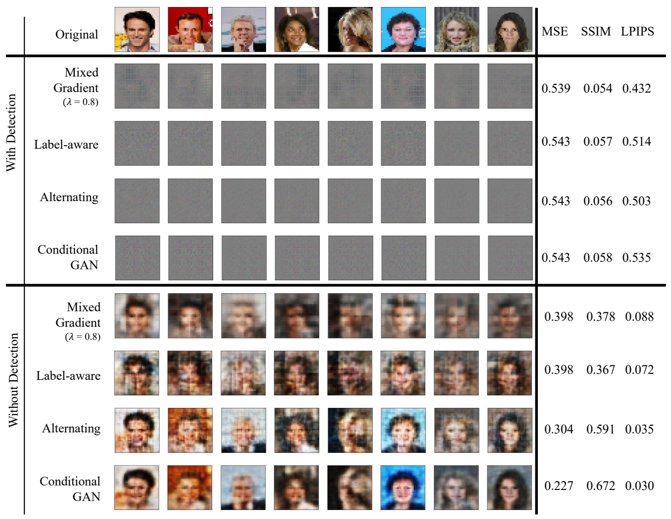

Original
MSE SSIM LPIPS
With Detection
Mixed Gradient (λ = 0.8) 0.539 0.054 0.432
Label-aware 0.543 0.057 0.514
Alternating 0.543 0.056 0.503
Conditional GAN 0.543 0.058 0.535
Without Detection
Mixed Gradient (λ = 0.8) 0.398 0.378 0.088
Label-aware 0.398 0.367 0.072
Alternating 0.304 0.591 0.035
Conditional GAN 0.227 0.672 0.030

Fig. 26: Reconstructed images of different split-learning hijacking attacks on CelebA: when they are detected by Gradients Scrutinizer (middle section) and when no detection is applied (bottom section).

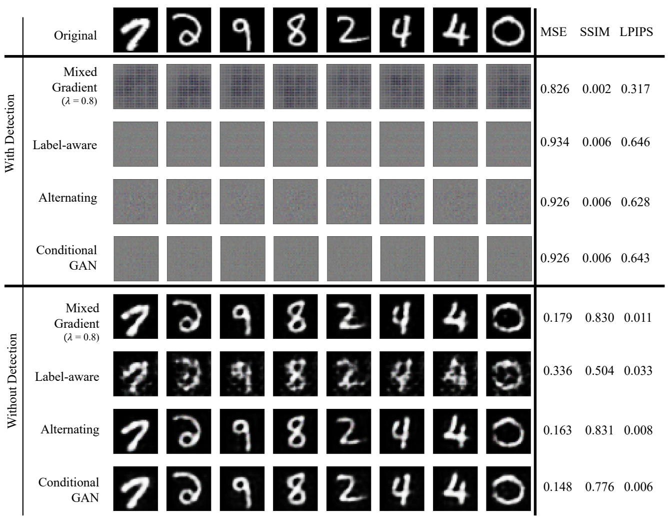

Original
MSE SSIM LPIPS
With Detection
Mixed Gradient (λ = 0.8) 0.826 0.002 0.317
Label-aware 0.934 0.006 0.646
Alternating 0.926 0.006 0.628
Conditional GAN 0.926 0.006 0.643
Without Detection
Mixed Gradient (λ = 0.8) 0.179 0.830 0.011
Label-aware 0.336 0.504 0.033
Alternating 0.163 0.831 0.008
Conditional GAN 0.148 0.776 0.006

Fig. 25: Reconstructed images of different split-learning hijacking attacks on MNIST: when they are detected by Gradients Scrutinizer (middle section) and when no detection is applied (bottom section).

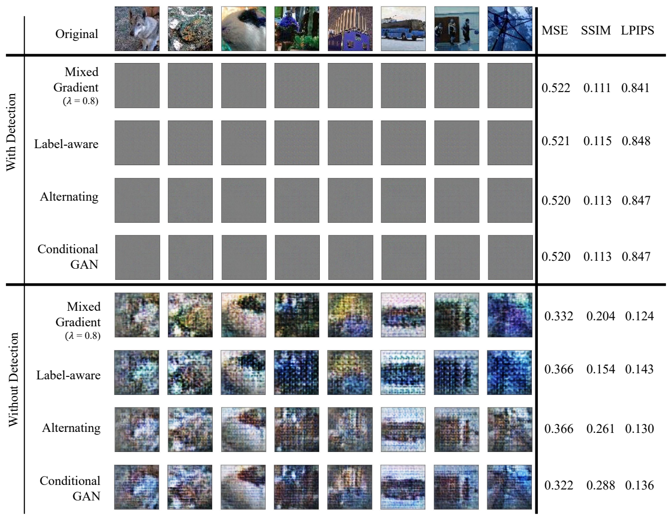

Original
MSE SSIM LPIPS
With Detection
Mixed Gradient (λ = 0.8) 0.522 0.111 0.841
Label-aware 0.521 0.115 0.848
Alternating 0.520 0.113 0.847
Conditional GAN 0.520 0.113 0.847
Without Detection
Mixed Gradient (λ = 0.8) 0.332 0.204 0.124
Label-aware 0.366 0.154 0.143
Alternating 0.366 0.261 0.130
Conditional GAN 0.322 0.288 0.136

Fig. 27: Reconstructed images of different split-learning hijacking attacks on ImageNet: when they are detected by Gradients Scrutinizer (middle section) and when no detection is applied (bottom section).

the loss function to train the client-side model:

$$L_{af} = \frac{1}{n}\sum_{i=0}^{n-1} D_n(f(X))[i] \cdot w_i \tag{26}$$

where

$$w_i = \begin{cases} 1 & \text{i} \neq \text{y} \\ w_d > 1 & \text{i=y} \end{cases} \tag{27}$$

y is the label of sample X, and w_d is the weight controlling the significance of the sample's label. When $w_d = 1$, all components equally contribute to L_{af}, and the discriminator reduces to the traditional label-unaware discriminator. The larger the value of w_d, the closer the discriminator behaves like a classifier.

Detection results of Gradients Scrutinizer against the attack using label-aware GAN are shown in Fig. 28, and reconstructed images of the 4 datasets at the step that the attack is detected and also at the end of training if there is no defense are shown in Figs. 24-27. We can see from these figures that this attack is ineffective in evading Gradients Scrutinizer and cannot infer any meaningful information about private training data when the attack is detected.

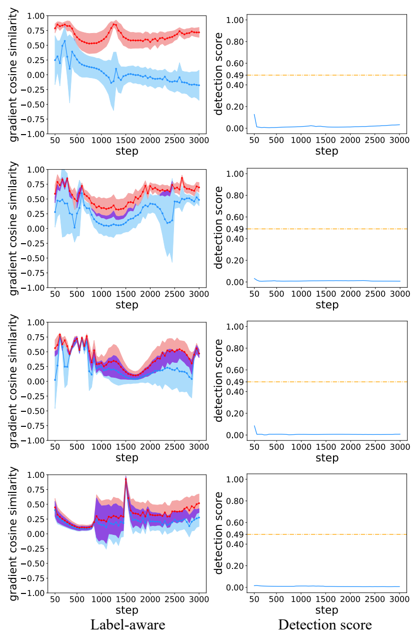


Fig. 28: Gradients Scrutinizer's detection performance against the attack with label-dependent GAN on different datasets. From top to bottom: MNIST, Cifar10, CelebA, and ImageNet. See the caption of Fig. 17 for more information.

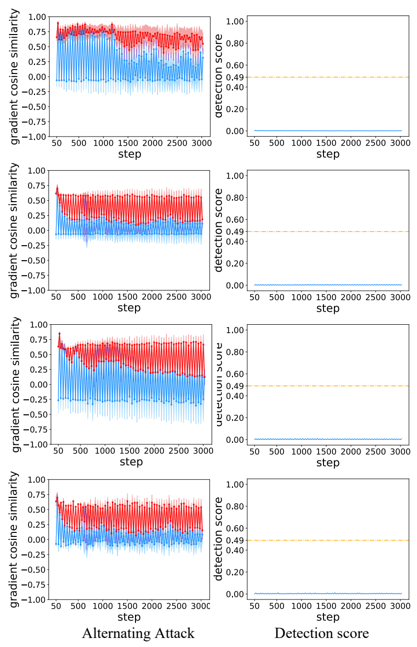


Fig. 29: Gradients Scrutinizer's detection performance against the alternating attack on MNIST, Cifar10, CelebA, and ImageNet (from top to bottom). See the caption of Fig. 17 for more information.

C. Alternating Attack

As mentioned before, an adversary, Bob, can train a server-side classification model. Instead of mixing the benign gradients, Bob can alternatively back propagate benign gradients and attack gradients to evade detection. For each $2n$ steps, Bob first sends benign gradients from the classification model

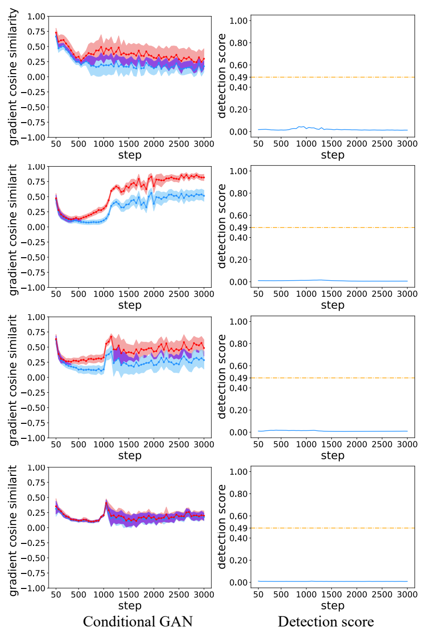


Fig. 30: Gradients Scrutinizer's detection performance against the attack with conditional GAN on MNIST, Cifar10, CelebA, and ImageNet (from top to bottom). See the caption of Fig. 17 for more information.

for n steps and then sends FSHA attack gradients for next n steps. Detection results with $n = 10$ are shown in Fig. 29 when gradients are checked step by step, and reconstructed images of the 4 datasets at the step that the attack is detected and also at the end of training if there is no defense are shown in Figs. 24-27. We can see that the alternating attack can be effectively detected by Gradients Scrutinizer, mainly due to large fitting errors, and cannot infer any meaningful information about private training data when the attack is detected.

D. Attack with Conditional GAN

Conditional GAN depends on labels. It can be used to enhance FSHA against our defense. In our evaluation, we add an embedding layer in server's training to embed label information before the input to server's autoencoder. Then we apply the following optimization, which is proposed in [34]:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D_d(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D_c(G(\mathbf{z}|\mathbf{y})))] \tag{28}$$

Detection results are shown in Fig. 30, and reconstructed images of the 4 datasets at the step that the attack is detected and also at the end of training if there is no defense are shown in Figs. 24-27. We can see that Gradients Scrutinizer can still effectively detect the attack based on conditional GAN, and no meaningful information about private training data can be inferred when the attack is detected.

VIII. Conclusion

In this paper, we first propose SplitSpy, an effective attack against SplitGuard, the only existing effective defense against split-learning hijacking attacks, to the best of our knowledge.

In SplitSpy, a malicious server maintains a legitimate model that performs the intended task and is trained in a speedup manner. It facilitates detection of fake training samples used in SplitGuard, and is used to calculate gradients of detected fake samples to send to clients. Our experimental evaluation indicates that SplitSpy can effectively evade SplitGuard detection. Then we propose Gradients Scrutinizer, a novel passive detection method that relies on the intrinsic distinguishability between honest training and hijacked training for the expected similarity among gradients of same-label samples and that among gradients of different-label samples. This intrinsic distinguishability makes Gradients Scrutinizer an effective detector against split-learning hijacking attacks. Our extensive evaluation indicates that Gradients Scrutinizer can effectively thwart both known split-learning hijacking attacks and adaptive counterattacks.

Acknowledgements

This work was partially supported by National Natural Science Foundation of China under Grants No. 62272175, 62272186, and U1936211, Hubei Province Key R&D Technology Special Innovation Project under Grant No. 2021BAA032, the Wuhan Applied Foundational Frontier Project under Grant No. 2020010601012188, and Ant Group.

References

[1] A. Abedi and S. S. Khan, "FedSL: Federated split learning on distributed sequential data in recurrent neural networks," *arXiv preprint arXiv:2011.03180*, 2020.

[2] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Camtepe, Y. Gao, H. Kim, and S. Nepal, "Can we use split learning on 1D CNN models for privacy preserving training?" in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, pp. 305–318.

[3] G. J. Annas, "Hipaa regulations - a new era of medical-record privacy?" *The New England Journal of Medicine*, vol. 348, no. 15, pp. 1486–1490, April 2003.

[4] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," in *Proceedings of the International Conference on Learning Representations*, 2017.

[5] M. Arjovsky, S. Chintala, and L. Bottou, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent manitude." in *Technical Report: Neural Networks of Machine Learning*, 2012, pp. 26–31.

[6] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70. PMLR, 06–11 Aug 2017, pp. 214–223.

[7] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," in *Proceedings of Machine Learning and Systems*, vol. 1, 2019, pp. 374–388.

[8] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li, "Imagenet: A large-scale hierarchical image database," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 248–255.

[9] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy." *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3-4, pp. 211–407, 2014.

[10] E. Erdoğan, A. Küpçü, and A. E. Çiçek, "Splitguard: Detecting and mitigating training-hijacking attacks in split learning," in *Proceedings of the 21st Workshop on Privacy in the Electronic Society*, ser. WPES'22. New York, NY, USA: Association for Computing Machinery, 2022, p. 125–137.

[11] E. Erdoğan, A. Küpçü, and A. E. Çiçek, "Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning," in *Proceedings of the 21st Workshop on Privacy in the Electronic Society*, ser. WPES'22. New York, NY, USA: Association for Computing Machinery, 2022, p. 115–124.

[12] G. Gawron and P. Stubbings, "Feature space hijacking attacks against differentially private split learning," *arXiv preprint arXiv:2201.04018*, 2022.

[13] B. Ghazi, N. Golowich, R. Kumar, P. Manurangsi, and C. Zhang, "Deep learning with label differential privacy," in *Proceedings of Neural Information Processing Systems*, vol. 34, 2021, pp. 27 131–27 145.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of Neural Information Processing Systems*, vol. 27, 2014, pp. 1–9.

[15] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[16] C. He, M. Annavaram, and S. Avestimehr, "Group knowledge transfer: Federated learning of large CNNs at the edge," in *Proceedings of Neural Information Processing Systems*, vol. 33, 2020, pp. 14 068–14 080.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[18] Z. He, T. Zhang, and R. B. Lee, "Attacking and protecting data privacy in edge–cloud collaborative inference systems," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9706–9716, 2020.

[19] G. E. Hinton and R. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Proceedings of the Neural Information Processing Systems*, vol. 6, 1993, pp. 3–10.

[20] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5967–5976.

[21] P. Joshi, C. Thapa, S. Camtepe, M. Hasanuzzamana, T. Scully, and H. Afli, "Splitfed learning without client-side synchronization: Analyzing client-side split network portion size to overall performance," *arXiv preprint arXiv:2109.09246*, 2021.

[22] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. Oliveira, H. Eichner, S. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[23] S. Kariyappa and M. K. Qureshi, "ExPLoit: Extracting private labels in split learning," *arXiv preprint arXiv:2112.01299*, 2021.

[24] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[25] Y. Koda, J. Park, M. Bennis, K. Yamamoto, T. Nishio, M. Morikura, and K. Nakashima, "Communication-efficient multimodal split learning for mmwave received power prediction," *IEEE Communications Letters*, vol. 24, no. 6, pp. 1284–1288, 2020.

[26] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, pp. 32–33, 2009.

[28] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[29] O. Li, J. Sun, X. Yang, W. Gao, H. Zhang, J. Xie, V. Smith, and C. Wang, "Label leakage and protection in two-party split learning," *arXiv preprint arXiv:2102.08504*, 2021.

[30] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[31] J. Liu and X. Lyu, "Clustering label inference attack against practical split learning," *arXiv preprint arXiv:2203.05222*, 2022.

[32] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*. IEEE Computer Society, December 2015, pp. 3730–3738.

[33] F. Mireshghallah, M. Taram, A. Jalali, A. T. Elthakeb, D. Tullsen, and H. Esmaeilzadeh, "A principled approach to learning stochastic representations for privacy in deep neural inference," *arXiv preprint arXiv:2003.12154*, 2020.

[34] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[35] A. H. Murphy, "The finley affair: A signal event in the history of forecast verification," *Weather and forecasting*, vol. 11, no. 1, pp. 3–20, 1996.

[36] D. Pasquini, "Unleashing the tiger: Inference attacks on split learning," https://github.com/pasquini-dario/SplitNN_FSHA, 2021, accessed on May 1st, 2022.

[37] D. Pasquini, G. Ateniese, and M. Bernaschi, "Unleashing the tiger: Inference attacks on split learning," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021, pp. 2113–2129.

[38] M. Pathak, S. Rane, and B. Raj, "Multiparty differential privacy via aggregation of locally trained classifiers," in *Proceedings of the Neural Information Processing Systems*, vol. 23, 2010, pp. 1876–1884.

[39] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split learning for collaborative deep learning in healthcare," *arXiv preprint arXiv:1912.12115*, 2019.

[40] P. Regulation, "General Data Protection Regulation," *Official Journal of the European Union*, vol. 25, pp. 1–88, 2018.

[41] J. Ryu, D. Won, and Y. Lee, "A study of split learning model to protect privacy," *Convergence Security Journal*, vol. 21, no. 3, pp. 49–56, 2021.

[42] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," in *Proceedings of the Neural Information Processing Systems*, vol. 29, 2016, pp. 2226–2234.

[43] M. Samragh, H. Hosseini, A. Triastcyn, K. Azarian, J. Soriaga, and F. Koushanfar, "Unsupervised information obfuscation for split inference of neural networks," *arXiv preprint arXiv:2104.11413*, 2021.

[44] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed comparison of communication efficiency of split learning and federated learning," *arXiv preprint arXiv:1909.09145*, 2019.

[45] G. J. Székely and M. L. Rizzo, "Partial distance correlation with methods for dissimilarities," *The Annals of Statistics*, vol. 42, no. 6, pp. 2382–2412, 2014.

[46] C. Thapa, M. A. P. Chamikara, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," *arXiv preprint arXiv:2004.12088*, 2020.

[47] T. Titcombe, A. J. Hall, P. Papadopoulos, and D. Romanini, "Practical defences against model inversion attacks for split neural networks," *arXiv preprint arXiv:2104.05743*, 2021.

[48] V. Turina, Z. Zhang, F. Esposito, and I. Matta, "Combining split and federated architectures for efficiency and privacy in deep learning," in *Proceedings of the 16th International Conference on Emerging Networking Experiments and Technologies*, 2020, pp. 562–563.

[49] P. Vepakomma, O. Gupta, A. Dubey, and R. Raskar, "Reducing leakage in distributed deep learning for sensitive health data," *arXiv preprint arXiv:1812.00564*, 2019.

[50] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[51] P. Vepakomma, T. Swedish, R. Raskar, O. Gupta, and A. Dubey, "No peek: A survey of private distributed deep learning," *arXiv preprint arXiv:1812.03288*, 2018.

[52] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[53] X. Yang, J. Sun, Y. Yao, J. Xie, and C. Wang, "Differentially private label protection in split learning," *arXiv preprint arXiv:2203.02073*, 2022.

[54] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *Proceeding of European Conference of Computer Vision(ECCV)*. Springer, 2014, pp. 818–833.

[55] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 586–595.

[56] B. Zhao, K. R. Mopuri, and H. Bilen, "iDLG: Improved deep leakage from gradients," *arXiv preprint arXiv:2001.02610*, 2020.

Appendix

A. DNN Used in Our Experiments and Split Layers

The same ResNet and four split-layer settings used in evaluating FSHA in [37] are used in our experiments. The detail is provided in Table I. Split 3 in the table is used as our default split layer.

B. Reconstructed Images of FSHA when DP is Used

As mentioned in Section II-C, Differential Privacy (DP) is studied in [12] to defend against FSHA. Fig. 31 shows reconstructed images of FSHA on MNIST and Cifar10 when DP is used. We can see that the private training data can be reconstructed, esp. when privacy budget ϵ is large. The results agree with the conclusion in [12] that DP is ineffective in thwarting FSHA.

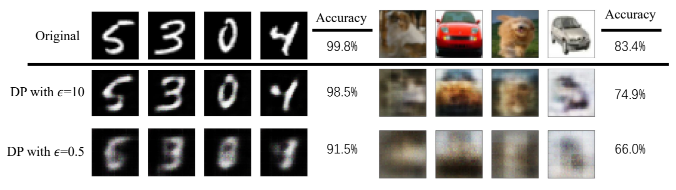


Fig. 31: Reconstructed images of FSHA with Differential Privacy (DP) training (privacy budget ϵ is set to 0.5 and 10) of 100,000 steps on MNIST and Cifar10. The accuracy column shows the accuracy of the resulting honest training model.

TABLE I: Architectures and split layers in our experiments. They are identical to those used in evaluating FSHA in [37]. The Split 3 is used as the default split.

Split	f	$\tilde{f}$	$\tilde{f}^{-1}$	D

1	`2D-Conv(64, 3, (1,1), ReLU)`
	`batch-normalization`
	`ReLU`
	`maxPolling((2,2))`
	`resBlock(64, 1)`
		`2D-Conv(64, 3, (2,2), linear)`
		`2D-Conv(64, 3, (1,1), linear)`
			`2D-ConvTrans(256, 3, (2,2), linear)`
			`2D-Conv(3, 3, (1,1), tanh)`
				`2D-Conv(128, 3, (2,2), ReLU)`
				`2D-Conv(128, 3, (2,2))`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`2D-Conv(256, 3, (2,2), ReLU)`
				`dense(1)`

2	`2D-Conv(64, 3, (1,1), ReLU)`
	`batch-normalization`
	`ReLU`
	`maxPolling((2,2))`
	`resBlock(64, 1)`
	`resBlock(128, 2)`
		`2D-Conv(64, 3, (2,2), linear)`
		`2D-Conv(128, 3, (2,2), linear)`
		`2D-Conv(128, 3, (1,1)`
			`2D-ConvTrans(256, 3, (2,2), linear)`
			`2D-ConvTrans(128, 3, (2,2), linear)`
			`2D-Conv(3, 3, (1,1), tanh)`
				`2D-Conv(128, 3, (2,2))`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`2D-Conv(256, 3, (2,2), ReLU)`
				`dense(1)`

3	`2D-Conv(64, 3, (1,1), ReLU)`
	`batch-normalization`
	`ReLU`
	`maxPolling((2,2))`
	`resBlock(64, (1,1))`
	`resBlock(128, 2)`
	`resBlock(128, 1)`
		`2D-Conv(64, 3, (2,2), linear)`
		`2D-Conv(128, 3, (2,2), linear)`
		`2D-Conv(128, 3, (1,1)`
			`2D-ConvTrans(256, 3, (2,2), linear)`
			`2D-ConvTrans(128, 3, (2,2), linear)`
			`2D-Conv(3, 3, (1,1), tanh)`
				`2D-Conv(128, 3, (2,2))`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`2D-Conv(256, 3, (2,2), ReLU)`
				`dense(1)`

4	`2D-Conv(64, 3, (1,1), ReLU)`
	`batch-normalization`
	`ReLU`
	`maxPolling((2,2))`
	`resBlock(64, 1)`
	`resBlock(128, 2)`
	`resBlock(128, 1)`
	`resBlock(256, 2)`
		`2D-Conv(64, 3, (2,2), linear)`
		`2D-Conv(128, 3, (2,2), linear)`
		`2D-Conv(256, 3, (2,2), linear)`
		`2D-Conv(256, 3, (1,1))`
			`2D-ConvTrans(256, 3, (2,2), linear)`
			`2D-ConvTrans(128, 3, (2,2), linear)`
			`2D-ConvTrans(3, 3, (2,2), tanh)`
				`2D-Conv(128, 3, (1,1))`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`resBlock(256, 1)`
				`2D-Conv(256, 3, (2,2), ReLU)`
				`dense(1)`