A Privacy-Preserving Federated Learning System for Android Malware Detection Based on Edge Computing

Ruei-Hau Hsu
Department of Computer Science and Engineering, National Sun Yat-sen University
Kaohsiung, Taiwan
rhhsu@mail.cse.nsysu.edu.tw

Yi-Cheng Wang
Department of Computer Science and Engineering, National Sun Yat-sen University
Kaohsiung, Taiwan
m073040095@student.nsysu.edu.tw

Chun-I Fan
Department of Computer Science and Engineering, National Sun Yat-sen University
Kaohsiung, Taiwan
cifan@mail.cse.nsysu.edu.tw

Bo Sun
National Institute of Information and Communications Technology
Tokyo, Japan
sunshine@nsl.cs.waseda.ac.jp

Tao Ban
National Institute of Information and Communications Technology
Tokyo, Japan
bantao@nict.go.jp

Takeshi Takahashi
National Institute of Information and Communications Technology
Tokyo, Japan
takeshi_takahashi@nict.go.jp

Ting-Wei Wu
Department of Computer Science and Engineering, National Sun Yat-sen University
Kaohsiung, Taiwan
m083040057@student.nsysu.edu.tw

Shang-Wei Kao
Department of Computer Science and Engineering, National Sun Yat-sen University
Kaohsiung, Taiwan
m083140011@student.nsysu.edu.tw

***Abstract*—This paper presents a privacy-preserving federated learning (PPFL) system for the detection of android malware. The proposed PPFL allows mobile devices to collaborate together for training a classifier without exposing the sensitive information, such as the application programming interface (API) calls and permission configuration, and the learned local model by each mobile device. This work implements the privacy-preserving federated learning system based on support vector machine (SVM) and secure multi-party computation techniques. It also demonstrates the feasibility using the Android malware dataset by National Institute of Information and Communication Technology (NICT), Japan. The presented experiments evaluate the performance of the trained classifier by the proposed PPFL system. The evaluation also compares the performance of the classifier of PPFL and that of centralized training system for the use cases of i) different data set and ii) different features on distinct mobile device. The results show that the performance of the PPFL classifier outperforms that of centralized training system. Moreover, the privacy of app information (i.e., API and permission information) and trained local models is guaranteed. To the best of our knowledge, this work is the first Android malware detection system based on privacy-preserving federated learning system.**

***Acknowledgements*—This research was supported by the Ministry of Science and Technology, Taiwan (ROC), under Project Numbers MOST 108-2221-E-110-033, and by Taiwan Information Security Center at National Sun Yat-sen University (TWISC@NSYSU).**

***Keywords*—Android malware, Privacy-preserving federated learning, Support vector machine, Data privacy, Secure multi-party computation**

I. Introduction

In recent year, smartphones became essential technology products for people to use for work, online education, entertainment, etc. Statistics Portal [1] in the second quarter of 2018 showed that Android is the most widely used mobile operating system with 88% of the global mobile phone market. Because of Android open specification, Android application markets (e.g., Google Play, F-Droid) are easier to be attacked by cybercriminals. In the first half of 2019, experts in G DATA security discovered around 1.9 million new malicious apps [2]. The malware can make different types of attacks such as financial loss, steal personal information, encrypt data, etc., which have already seriously threatened users' security and privacy.

It is expected that the dramatic increase in the number of mobile malware will happen because Android devices become popular in the world. One reason is that users are allowed to install applications from third-party markets. Therefore, the attackers can easily apply drive-by download to mislead users to download malware from the attacker's server. Although the Android platform provides a security mechanism that uses the permission control to limit the functionality of malware, most

978-1-7281-9922-1/20/$31.00 ©2020 IEEE
DOI 10.1109/AsiaJCIS50894.2020.00031

users often blindly grant permissions to applications, which lead to the invalidation of the permission mechanism. As malware often use excessive permissions to conduct malicious activities, many studies use permissions as a feature to detect malicious applications [3], [5]–[8]. API calls are another important feature because many important system resources access could only be obtained by APIs [3], [5]–[7], [33].

Machine learning technology has been widely adopted to implement analysis automation for decision making to support various kinds of information services. The use of machine learning techniques, such as SVM and K nearest neighbors (KNN), to identify malicious software has become mainstream. Whether the features using static methods to extract such as APIs and permissions or using dynamic methods to extract monitored runtime behaviors of applications, these features are directly used in machine learning analysis without considering user/data privacy. Recently, people pay more attention to the security and privacy of sensitive information on smartphones, where contain much personal information that can be used to identify individuals, e.g., IMEI and location, such that users' behaviors can be observed. Even though the record of app usage does not contain explicit personal identity information, it may still provide essential knowledge to link to each specific individual after correlation analysis [45]. Also, several countries legislate laws regarding the protection of personal data privacy to the restricted analysis of unauthorized data in the future, e.g., EU GDPR [34].

A distributed machine learning system is also urgently desired since various kinds of app data may be owned by different mobile devices. Thus, it is upmost that how to utilize data more effectively, so that the training procedure of machine learning can exploit as much data as possible from distributed mobile devices to maximize the performance of the trained classifier. A federated learning framework is a practical solution for the training of distributed data cooperatively [17]–[21].

To fulfill the requirements of privacy protection and distributed learning, this work proposes a privacy-preserving federated learning system for android malware detection. The main idea of federated learning is that mobile devices collaboratively train a global model and the raw training data would not be known to the server. Because mobile devices just send model parameters to the server, but not the raw data while aggregating the model parameters from all users. Using federated learning has two benefits of protecting data privacy and saving network traffic. Furthermore, the federated model can have almost the same as performance compared to centralized learning. Federated learning and centralized learning overview are depicted in Fig. 1, respectively. Moreover, this work proposes a system architecture for exploiting the computing resources provided by edge computing (shown in Fig. 2) infrastructure for lower latency and decentralized data analytics in 5G. Our framework has the flexibility for the models of different features to combine as a global model.

In summary, this paper has the following contributions:

1) We proposed a novel Android malware detection approach that uses static analysis and meta-data analysis with privacy-preserving federated learning. To the best of our knowledge, this work is the first implementation of privacy-preserving federated learning for Android malware detection.
2) We design a system architecture based on edge computing to reduce latency and protect data privacy against honest-but-curious service operators.
3) We conduct experiments in different aspects to evaluate the performance of our proposed PPFL system. The results show that federated learning can perform almost the same accuracy with an identical amount of data compared to the traditional centralized machine learning systems.

The rest of this paper is organized as follows. Section 2 reviews related work on android malware detection. Section 3 presents a detailed description of the proposed approach for Android malware detection. Section 4 describes the dataset and our evaluation. The conclusion is presented in Section 5. Future work is presented in Section 6.

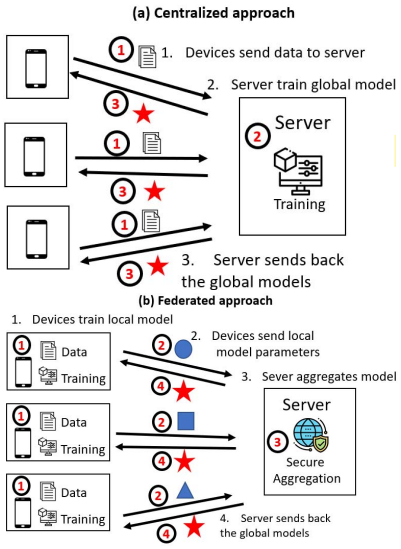

Fig. 1. Overview of machine learning approaches

II. Related Works

Over the past few years, there are many papers studying the detection of Android malware through static and dynamic analyses. This section summarizes relevant research works in two categories: static and dynamic analyses for Android malware detection, and federated learning.

A. Static and dynamic analysis

Peiravian et al. [5] proposed an Android malware detection framework using both permission and API to train a classifier to characterize malicious and benign apps. Huang et al. [8] used permissions as features and evaluated performance by four different machine learning methods and get the results that permission-based mechanisms can be used

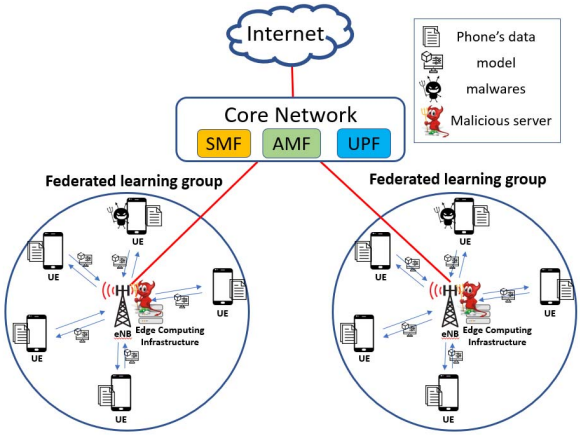

Fig. 2. System model

to quickly identify malicious applications. Arp et al. [3] proposed DREBIN that performed the broad static analysis with Android permissions and sensitive APIs. DREBIN is a lightweight method that enables to detect malware on the smartphone. Aafer et al. [4] provided a robust and lightweight classifier to mitigate Android malware and extract features by malware behavior analysis at the API level. Takahashi et al. [6] applied SVM to classify Android applications and evaluate the effectiveness of using various types of features including API calls, permissions, application category, and application descriptions. Sun et al. [7] proposed an approach that can process analysis with high accuracy and low computation time to solve large-scale detection problem. Zhang et al. [9] proposed a prototype system, DroidSIFT, that used a weighted contextual API dependency graph as features to do semantic-based malware classification. TaintDroid [10] performed the first dynamically taint analysis of applications by monitoring the leakage of sensitive data to detect suspicious behaviors during runtime. DroidScope [12] is a virtualization-based Android analysis platform that extracts native and Dalvik instruction traces to use in taint analysis. It reconstructs both the operating system level and Java level semantic information to enable their platform dynamically monitoring applications. Wang et al. [11] proposed an Android malware detection method through the text semantic features of network traffic to analyze the network flows as a document by using NLP string analysis.

Most of the papers work on extracting out features from malicious and benign apps and using machine learning for classification, but they do not consider users' privacy. The smartphone contains many users' private-sensitive data, such as bank accounts and personal contacts. Even if you only know what application to use, users may still reveal personally identifiable information [45]. Therefore, people may not want their applications to be analyzed, and the protection of the privacy of user app usage and permission information is essential when designing a detection system. The following is an introduction to federated learning that we use to analyze data distributed on mobile devices for Android malware detection, and protect the privacy of user app usage and permission information..

B. Federated Learning and Its Privacy Protection

Federated learning is a distributed machine learning approach that enables multiple decentralized devices to collaboratively train models without sharing raw training data. An early version of this concept was proposed in [15] that their privacy-preserving deep learning system enables multiple users to train local models and use selective stochastic gradient descent to share with a server to update the global model later. In [13] [14], they also have the similar motivation and method, but the difference compared to [15] is that their federated averaging algorithm can reduce the rounds of communication when using deep learning for decentralized data training. Moreover, the proposed method is robust to unbalanced data and none independent and identical data distributions. The first use case of federated learning is Google's keyboard [16] [17] [18] to make word prediction. Considering the impact of a mobile device's performance when training local models, so Google's federated learning mechanism performs for training when there is wireless connections available and the device is in idle mode. In recent years, academia and the medical profession have noticed federated learning because of the ability to resolve data privacy and data security. Nguyen et al. [19] proposed an autonomous self-learning distributed system for detecting anomalous deviations in IoT devices' communication behavior, potentially caused by malicious adversaries. In the NVIDIA blog [20], they introduce NVIDIA Clara Federated Learning that lets participating hospitals share their local training results across multiple hospitals to jointly develop robust AI models without leaking patient data.

Although the methods in [13] [15] allow local clients to update the results by model parameters using stochastic gradient descent (SGD), which is an optimization algorithm, these gradients may leak sensitive data information to attackers when exposed together with the data structure. Several researchers point to attacks on federated learning; e.g., In [21], researchers show that sensitive information can be obtained based on the shared gradients. In [22] [23] [24], they indicated that the vulnerability of the federated learning approach about the problem of poisoning the Federated model via malicious clients. Poisoning attacks on federated learning can be more powerful than centralized settings because federated learning gives the adversaries the opportunity to directly influence the properties of the joint model, not just be passive data providers. In [25], the authors consider user-level privacy leakage against a malicious server.

To prevent from the above attacks, several researchers proposed an additional mechanisms to make federated learning more secure. In [29] [30] [31], differential privacy is used to offer privacy guarantees. Bonawitz et al. [26] presented a practical and secure aggregation protocol for federated learning. By using the protocol based on secret sharing and key agreement, users can have low communication overhead, robustness to failures, and no leakage of user data. Trieu Phong et al. [27]

proposed a privacy-preserving deep learning system with the combination of asynchronous stochastic gradient descent and additively homomorphic encryption, which fix the problem of the previous work [15] that an honest-but-curious server may know the local data information. Guowen Xu et al. [28] proposed VerifyNet which utilizes a double-masking protocol to ensure the confidentiality of users' local gradients and require the server to provide the proof to let each user verify the server's aggregated results.

In this paper, we implement the privacy-preserving federated learning for Android malware detection. We use the open source OpenMined/PySyft [32] which is a python library for private and secure deep learning to evaluate our framework.

III. Proposed Method

In this section, we present the proposed method with three components. First, we introduce data collection and pre-processing that use static analysis to extract useful information as features from the APK files. Second, we introduce how to encode all the features consistently. Third, we implement privacy-preserving federated learning to Android malware detection and introduce the Secure Multi-Party Computation (SMPC) that protect data privacy.

A. Data Collection and Pre-processing

We collected APK files from the Opera Mobile Store by using crawler. The approaches of extracting useful information from APK file are basically classified into two types: static analysis and dynamic analysis. We focus on static analysis to extract features in this paper because of its lightweight property. Static analysis examines the APK file which consists of resources, certificates, and the program's code without executing the application. We use Apktool [35] to decompile the APK files and then convert AndroidManifest.xml into text. Moreover, we used Dedexer [36] to convert classes.dex to a Java source code. AndroidManifest.xml and classes.dex are our main sources of feature extraction that can extract permissions request and API calls, respectively. In addition, we also apply the metadata as a feature that can be obtained from the descriptive page on the markets. However, the descriptions of applications need to be converted to an informative feature. We use the approach in [37] to classify the APK files into clusters. The following are the three stages of this approach:

1) **Data preprocessing stage.** Usable words are produced for latent Dirichlet allocation (LDA) [38] from the descriptions. First, we discard non-English descriptions and non-textual items, i.e., numbers, HTML tags, web links, and email addresses, by checking the language and the format of the descriptions. Then, we use word stemming and stop work removal to preprocess the application description information. Finally, all descriptions which contain less than ten words are discarded after counting the number of words in the description.
2) **Topic model generation stage.** We use LDA to process the words in the remaining descriptions. A total of 300 topics, with a topic proportion threshold of 0.05 and a maximum of four topics per entry, were considered. Therefore, several (maximum four) topic number-proportion value pairs were outputted in this process.
3) **Cluster generation stage.** K-means was used to classify the APK files into clusters according to the topic number-proportion value pairs for each description. The identical number of categories is set to 12 as in the Opera Mobile Store.

B. Data Encoding

Feature format is different from source to source. Before using these features to implement machine learning approach, we must encode all the features as binary attributes to ensure consistency. If the permission/API is used, we set the attribute of permission requests and API calls to 1; otherwise, we set it to 0. To encode application categories and clusters, we use one-hot encoding which creates a binary column for each categorical feature so that the attribute is set to 0 or 1 according to whether the feature is used or not. After that, we can apply privacy-preserving federated learning by these features.

C. Privacy-Preserving Federated Learning

Following are the details of our model (Federated SVM) and Secure Multi-Party Computation (SMPC).

- **Federated SVM:** Federated learning can apply to any Stochastic Gradient Descent (SGD) based model. In SGD-based models, the clients can just exchange gradient updates with the server. After getting the local gradient from each client, the server uses the method called Federated Averaging in [13] to aggregate these gradients to get the global model by the following function:

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k, \tag{1}$$

where K is the number of clients,w_{t+1} is the new global model, η is a fixed learning rate, n_k is local example number, and g_k is the average gradient on its local data at the current model w_t. Because federated learning is used to train DNNs, we select SGD-based SVM to implement federated learning and compare the performance of federated approach with the centralized one in the previous work [42]. The different part of SGD-based SVM is how to solve the optimization problem, which uses SGD to compute the loss function. Otherwise, the SVM in sklearn uses Sequential minimal optimization (SMO). The goal of the two-class Support Vector Machine (SVM) is to find the decision hyperplane that can separate the data sample into two groups. The function of decision hyperplane,

$$f(x) = \langle w, x \rangle + b, \tag{2}$$

where w is the weight vector, x is the feature vector and b is the threshold, realizes the maximum margin from the hyperplane to the nearest data point of two classes. SVM uses hinge loss function,

$$l(w, x, y) = \max(0, 1 - y \cdot (w \cdot x)), \tag{3}$$

that extend the case in which the data are inseparable, to find the maximum margin. If $y \cdot w \cdot x > 1$, then $\nabla l(x, x, y) = 0$, and the data is on the correct side of the margin. Otherwise, the value is proportional to the distance of the margin, and minimizing the following objective function is our goal:

$$f(w, X, Y) = \sum_i l\left(w, x_i, y_i\right) + \lambda \|w\|^2. \tag{4}$$

Therefore, we can compute SGD update by using hinge loss and use Federated Averaging algorithm to get the new global model. According to [43], a linear SVM get a guarantee of fast convergence rate and generalization performance at high-dimensional data, so more than 30,000 unique APIs used in our data is also suitable for classifying android application.

- **Secure multi-party computation (SMPC):** Using SMPC can let multiple participants combine their private inputs to make computation without knowing each other inputs. OpenMined/PySyft [32] which this work implemented federated learning using SMPC by additive secret sharing [40] and relying on the crypto protocols SecureNN [39] and SPDZ [38]. The following will introduce additive secret sharing and SPDZ which we mainly use in our work.

  Additive secret sharing: Secret sharing is a method that we can distribute a secret with the participating devices. If there are N mobile devices that want to do federated learning, they can use secret sharing to divide the data to N pieces which is allocated to each device. Each device cannot know the information of individual share because the share of data is random value and can be used only when picking $N-1$ number of shares to reconstruct. The encryption use mathematical space called integer quotient ring $\mathbb{Z}_Q$ that the integers are between 0 and $Q-1$, where Q is a big enough prime. For example, there are two device want to do federated learning, so they share the secret x which is their models with all devices in the federated group. The value of SVM model contain *Weight:* tensor([[$-0.0014, -0.0965, ..., 0.1726, 0.0402$]], requires_grad=True), and *Bias:* tensor([-0.5555], requires_grad=True). $\langle x \rangle_0^Q$ and $\langle x \rangle_1^Q$ denote the two shares of x over $\mathbb{Z}_Q$. If the client number increased, we also can generates secret by the following equation:

$$\begin{cases} r \xleftarrow{\$} \mathbb{Z}_Q & \\ \langle x \rangle_i^Q = r, & \text{if } i \in 0, \ldots, N-2 \\ \langle x \rangle_i^Q = x - \sum_{i=0}^{N-2} \langle x \rangle_i^Q (\mathrm{mod} Q). & \text{if } i \in N-1 \end{cases}$$

After that, clients can send the model to the server to aggregate without leaking privacy because server do not have the value of big prime Q to reconstruct the secret. Moreover, clients also do not worry about leaking model information to other clients because each random value to hidden each client's model is unknown.

SPDZ: By implementing the secure protocols SPDZ, the server can make the computation of the secret. We first mention *addition* as an example that the method how to compute the global model z by x and y ,which is the model parameters from two clients. $\langle x \rangle_0^Q$ and $\langle x \rangle_1^Q$, $\langle y \rangle_0^Q$ and $\langle y \rangle_1^Q$ are their secrets for two secret x and y, respectively. $\langle z \rangle_0^Q = \left(\langle x \rangle_0^Q + \langle y \rangle_0^Q\right)/N$ and $\langle z \rangle_1^Q = \left(\langle x \rangle_1^Q + \langle y \rangle_1^Q\right)/N$, which N is the number of client in the federated group, can be generated while the variables are still encrypted, and then the server sends back to clients. After that, the clients can reconstruct the value to get z by $\langle z \rangle_0^Q + \langle z \rangle_1^Q (\mathrm{mod} Q)$. Second, we introduce *multiplication* that computes z by $x \cdot y$. As mentioned above, we generate the secret of x and y, and take a triplet $(\langle a \rangle_i^Q, \langle b \rangle_i^Q, \langle c \rangle_i^Q, i \in 0, \ldots, N-1)$ by Beaver multiplication triplets [44] , which $a \cdot b = c$. Because we need to generate these triplets and their shares, the computation is provided by the role called crypto provider who needs to be trusted and help clients to do computation in PySyft [32] .Then we compute $\langle x \rangle_i^Q - \langle a \rangle_i^Q$ to get δ, and $\langle y \rangle_i^Q - \langle b \rangle_i^Q$ to get ϵ. Finally, the value of z can be computed as $\langle z \rangle_i^Q = \epsilon\delta + \delta\langle b \rangle_i^Q + \epsilon\langle a \rangle_i^Q + \langle c \rangle_i^Q$.

- **System flow and security:** The system flow of federated learning designed for decentralized data on multiple devices is depicted in Fig. 3. We assume that the data transmitted between clients and a server is secure, so the data is protected with confidentiality and integrity. In the initialization phase, the N clients who participate in the federated learning group randomly choose one of them to generate the big prime Q, which is needed in SMPC and only the clients will know the value. The security of each client's model is guaranteed when each random value to hidden each client's model is unknown to the other clients and the server. First, each client trains local model based on local data. Second, because the model comes from the client application information, clients use additive secret sharing to protect their local model parameters from the server and then send it to the server. Third, the server makes secure aggregation with the encrypted model parameters by using SMPC to get the new global model. Fourth, the server sends back the updated global model to the clients. Finally, the clients decrypt the model and can use it to detect malware.

IV. Evaluation

A. Dataset

Between January and September 2014, we collected 87,182 APK files from the Opera Mobile Store. We excluded files that could not successfully extract permission requests because permission requests are a necessary feature for our mobile application detection method. Then, we use VirusTotal to determine if the APK files are benign or malicious. VirusTotal is a malware analysis website that uses multiple engines from different security vendors to evaluate the risk of APK files. If one or more security vendors conclude that the APK file was

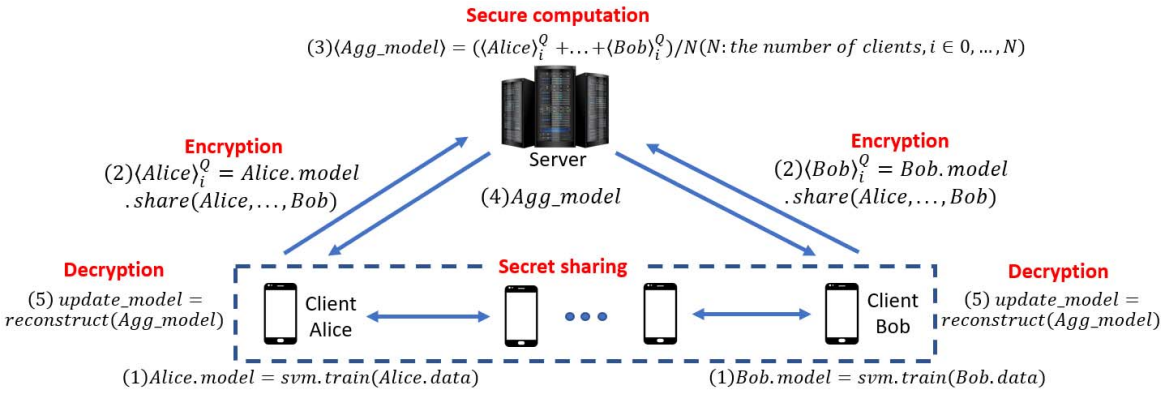

Fig. 3. System flow of privacy-preserving federated learning.

TABLE I
DATASET BY CATEGORY

Category	Benign	Malicious	Total
Business and finance	3779	268	4047
Communication	2114	323	2437
E-books	2784	479	3263
Entertainment	14138	2453	16591
Games	12090	2603	14693
Health	1536	228	1764
Languages and translators	734	41	775
Multimedia	2422	567	2989
Organizers	1300	87	1387
Ringtone	327	132	459
Theme skins	5276	5059	10335
Travel and maps	2545	445	2990
Total	49045	12685	61730

TABLE II
DATASET BY CLUSTER

Cluster	Benign	Malicious	Total
1	3574	934	4508
2	3883	889	4772
3	3945	976	4921
4	5247	1206	6453
5	4317	1174	5491
6	3820	1077	4897
7	3474	919	4393
8	5337	2091	7428
9	4104	811	4915
10	4346	832	5178
11	3496	818	4314
12	3502	958	4460
Total	49045	12685	61730

malware, we consider the APK file as a malicious application. Moreover, we previously excluded the APK files that could not be processed by VirusTotal from the dataset and omit the adware because most adware is designed to display ads. Following the above process, we obtained a dataset of 61,730 APK files, consisting of 49,045 benign and 12,685 malicious files. We also collected APK file metadata which included the application category, the description, and the number of downloads from the Opera Mobile Store. The statistical of the APK files for each category and cluster are shown in Table I and Table II, respectively.

B. Evaluation Method

To reduce problems such as overfitting or selection bias, we use 10-fold cross validation to evaluate our approach. The use of cross-validation can estimate the model's ability to predict unknown dataset because each round of cross-validation is performed using a different partitions of the dataset to be test dataset which was not used to train the model. In the beginning, the dataset is randomly partitioned into 10 folds. Then, the ith set is selected as a test set, and the complementary set of the ith set is used to be training sets. After 10 cross-validation iterations, the validation results are averaged to give an estimate of the model's predictive performance.

There are five widely used metrics to measure the performance of the classifier: accuracy, precision, recall, false positive rate, and F-measure. These metrics are calculated using following four intermediate measures.

- **True positive (TP):** the number of positive examples correctly classified as positives.
- **False positive (FP):** the number of negative examples incorrectly classified as positives.
- **True negative (TN):** the number of negative examples correctly classified as negatives.
- **False negative (FN):** the number of positive examples incorrectly classified as negatives.

Accuracy indicates what the probability of test records are classified correctly, i.e.,

$$Accuracy = \frac{TP + TN}{n}$$

The precision indicates what the probability of positive records are classified correctly, i.e.,

$$Precision = \frac{TP}{TP + FP}$$

The recall indicates what the probability of the positive record is classified correctly, i.e.,

$$Recall = \frac{TP}{TP + FN}$$

The FPR indicates what the probability of the negative record is classified incorrectly, i.e.,

$$FPR = \frac{FP}{FP + TN}$$

The F-measure is the harmonic mean of the precision and the recall, i.e.,

$$F - measure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

C. Model Evaluation

The codes of experiments are written in Python 3.8.1 and the programs are ran on a server with Intel i7-9700 3.0 GHz 8-core CPU and 40 GB memory.

- **Scenario 1: Centralized and Federated Models**

TABLE III
PERFORMANCE OF CENTRALIZED AND FEDERATED METHODS

Train on	Accuracy (%)	Precision (%)	Recall (%)	FPR (%)	F-measure
Local user 0	91.75	94.35	95.31	19.92	0.9478
Local user 1	91.99	95.99	94.14	15.71	0.9501
Local user 2	91.54	96.37	93.30	15.13	0.9476
Local user 3	91.61	95.86	93.86	16.22	0.9479
Local user 4	91.74	95.34	94.47	16.70	0.9481
Federated	93.45	96.96	94.94	12.41	0.9592
Centralized [42]	94.05	87.21	83.28	3.16	0.8520

In this experiment, we test the classification performance of federated SVM and centralized SVM with five users. We set local epochs to E=30, batch size to B=1, and all users use one-fifth of the training dataset to train the local model with API calls, permission, application category, and description feature. Table III shows the performance evaluation results, where the federated model performs slightly better than the local model on the accuracy and F-measure. We can also observe that the accuracy is very close to the previous work [42] trained by centralized way.

- **Scenario 2: Different Number of Clients**

TABLE IV
PERFORMANCE OF DIFFERENT NUMBER OF CLIENTS

Train on	Accuracy (%)	Precision (%)	Recall (%)	FPR (%)	F-measure
2 clients	92.88	96.65	94.55	13.67	0.9557
5 clients	93.36	98.87	95.81	12.49	0.9582
10 clients	93.65	97.05	95.07	12.18	0.9604
15 clients	93.87	96.53	95.79	13.78	0.9616
30 clients	93.87	96.81	95.53	12.97	0.9617

We consider that there will be many clients to use the service, so we test the classification performance of federated SVM with the different numbers of clients to evaluate our system with the increasing number of clients in this experiment. We set local epochs to E=30, batch size to B=1, and all users use 10,000 data which randomly chooses from the training dataset to train the local model with API feature. Table IV shows that the performance of 5 clients is better than 2 clients. Moreover, Fig. 4 shows that accuracy will converge after the number of clients reaches 15. Therefore, if there are more participating users, the model can perform better.

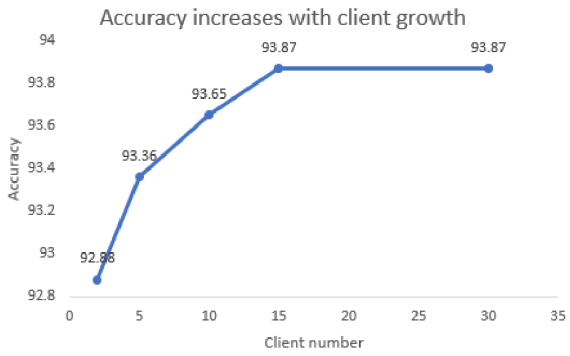

Fig. 4. Relation between accuracy and number of clients.

TABLE V
PERFORMANCE OF DIFFERENT TYPES OF FEATURES

Feature	Accuracy (%)	Precision (%)	Recall (%)	FPR (%)	F-measure
API calls	91.417	94.13	95.12	20.34	0.9455
Permission	81.43	92.92	85.90	31.14	0.8880
Federated	91.423	93.93	95.31	20.75	0.9454

- **Scenario 3: Use various type of features**
If two malware detection service providers have different features to detect android malware, they want to cooperate to detect android malware without leaking data. Therefore, in this experiment, we test the classification performance of federated SVM with a different types of features for each client to share its own local model securely. Thus, we set local epochs to E=30 and batch size to B=1. Two clients use 10,000 data to train a local model with permission and API calls, respectively. Table V shows that the classification performance of the aggregated model is almost the same as the local model using API calls. The case of using different types of features on individual clients for federated learning may lead to lower performance compared to that with the same features [13]. However, the result of the experiments shows acceptable performance using different features for malware detection.

D. Computation and Communication Evaluation

TABLE VI
COMPUTATION COSTS OF FEDERATED OPERATION

Federated operation	Mobile device	Edge device
train local model	30 $T_{SVM_{ldlf}}$	/
secret sharing	$R + N \cdot S + M$	/
model aggregation	/	$A + D$
decrypt model	M	/
$T_{SVM_{ldlf}}$: the computation of SVM training for the dataset ld with lf features R: random number selection M: module N: the number of clients A: addition S: subtraction D: division		

Table VI shows the computation of our framework. Table VII shows the time of centralized and federated operation. This experiment is tested with the same setting of scenario

TABLE VII
EXECUTION TIME OF CENTRALIZED AND FEDERATED LEARNING

Federated operation	time
train local model	187.03 (s)
secret sharing	0.006 (s)
model aggregation	0.052 (s)
Centralized operation	time (s)
train model	934.84 (s)

1, and the centralized method use SGD-based SVM to train the model. Compared with the centralized method, using the federated method to get the model took almost five times less execution time because five clients cooperate to train the model in parallel, and the model aggregation time can be almost omitted. Although we need to consider the different hardware condition because our experiment simulates the role with the same computation ability, we still think that federated learning can have less computation time to train the model. Moreover, the size of our model just has 145 KB, so it takes a very short time about 0.912 millisecond to send a model from client to server. Thus, the proposed framework only takes very low communication time.

In summary, we illustrated that our method can not only improve accuracy as the number of users rises but also can reduce the computation time. Moreover, we can aggregate multiple models that use different features. Therefore, android malware detection based on federated learning performs effectively when compared to the centralized approach in different situations.

V. CONCLUSION

In this paper, we propose a novel android malware detection framework against malware and honest-but-curious service providers. By employing privacy-preserving federated learning, the users' data are protected with SMPC so that mobile devices can utilize service providers resource, i.e. edge devices, to collaboratively train a global model without the leak of privacy. We extract features from the APK file by applying a static analysis and evaluate our method with different scenarios. Experiments show that the federated model can achieve higher accuracy than the local model and is comparable to the centralized model. Moreover, the experimental results show that the accuracy also increases with the growing number of clients.

VI. FUTURE WORKS

This section mentioned some discussions about the problems and limitations of our proposed framework in the real world, and we plan to implement the possible solutions in future work. First, we must consider that mobile phones in the real world have mobility issues, which means that there may not be a mobile phone participating in federated learning at any time. The mechanism of how to process mobile phones dropped out of training should be designed. Second, we plan to implement our proposed method and use dynamic analysis to extract features on smartphones. We perhaps use the open source for PPFL such as TensorFlow Federated [41] and PySyft [32], in which the plan that implements federated learning on mobile phones is on the road map. Third, we plan to deploy more machine learning method to the federated way in addition to SVM. Forth, the static analysis may not be enough to detect malware because some malware uses dynamic loading or obfuscation techniques, such as code encryption to evade detection. Therefore, we plan to use both static and dynamic methods to analyze the application. Fifth, the time that collects the feature and trains the model also needs to be considered since if we use dynamic analysis, we need to execute each application for nearly 2 minutes. Thus, we have to design how to decrease the training time or using a more lightweight method to analyze. Finally, we are also interested in the key agreement of the PPFL which mentioned in [26] because the encryption method in our framework maybe is not enough, and we want to design a stronger key agreement method to protect user data.

REFERENCES

[1] S. O'Dea, "The statistics portal," Feb. 2020. [Online]. Available: https://www.statista.com/statistics/266136/globalmarket-share-held-by-smartphone-operating-systems/

[2] K. Beckert-Plewka, H. Gierow, V. Haake, and S. Karpenstein, "G data mobile malware report," Jul. 2019. [Online]. Available: https://www.gdatasoftware.co.uk/news/2019/07/35228-mobile-malware-report-no-let-up-with-android-malware

[3] D. Arp, M. Spreitzenbarth, M. H¨ubner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," in *21st Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, Feb. 2014.

[4] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android," in *Security and Privacy in Communication Networks*, Sydney, NSW, Australia, 2013, pp. 86–103.

[5] N. Peiravian and X. Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls," in *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, 2013, pp. 300–305.

[6] T. Takahashi and T. Ban, "Android Application Analysis Using Machine Learning Techniques," in *AI in Cybersecurity*, Cham: Springer, 2019, pp. 181–205.

[7] B. Sun, T. Ban, S.-C. Chang, Y. S. Sun, T. Takahashi, and D. Inoue, "A scalable and accurate feature representation method for identifying malicious mobile applications," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, New York, NY, USA, Apr. 2019, pp. 1182–1189.

[8] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance Evaluation on Permission-Based Detection for Android Malware," in *Advances in Intelligent Systems and Applications*, vol. 2, Berlin, Heidelberg: Springer, 2013, pp. 111–120.

[9] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2014, pp. 1105–1116.

[10] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, vol. 57, no. 3, Mar. 2014., pp. 99–106.

[11] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting android malware leveraging text semantics of network flows," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1096–1109, May 2018.

[12] L. K. Yan and H. Yin, "DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis," in *Proceedings of the 21st USENIX Conference on Security Symposium*, 2012, pp. 29.
[13] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *arXiv preprint arXiv:1602.05629*, 2016.
[14] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," Apr. 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html
[15] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2015, pp. 1310–1321.
[16] M. Chen, R. Mathews, T. Ouyang, and F. Beaufays, "Federated Learning Of Out-Of-Vocabulary Words," *arXiv preprint arXiv:1903.10635*, 2019.
[17] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied Federated Learning: Improving Google Keyboard Query Suggestions," *arXiv preprint arXiv:1812.02903*, 2018.
[18] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated Learning for Mobile Keyboard Prediction," *arXiv preprint arXiv:1811.03604*, 2018.
[19] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A. Sadeghi, "DÏoT: A Federated Self-learning Anomaly Detection System for IoT," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, Jul. 2019, pp. 756–767.
[20] K. Powell, "Nvidia clara federated learning to deliver ai to hospitals while protecting patient data," Dec. 2019. [Online]. Available: https://blogs.nvidia.com/blog/2019/12/01/clara-federated-learning/
[21] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2017, pp. 603–618.
[22] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing Federated Learning through an Adversarial Lens," *arXiv preprint arXiv:1811.12470*, 2018.
[23] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Mitigating Sybils in Federated Learning Poisoning," *arXiv preprint arXiv:1808.04866*, 2018.
[24] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How To Backdoor Federated Learning," *arXiv preprint arXiv:1807.00459*, 2018.
[25] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond Inferring Class Representatives: User-Level Privacy Leakage From Federated Learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 2512–2520.
[26] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical Secure Aggregation for Privacy-Preserving Machine Learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2017, pp. 1175–1191.
[27] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-Preserving Deep Learning via Additively Homomorphic Encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.
[28] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and Verifiable Federated Learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2020.
[29] R. C. Geyer, T. Klein, and M. Nabi, "Differentially Private Federated Learning: A Client Level Perspective," *arXiv preprint arXiv:1712.07557*, 2017.
[30] A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers, "Protection Against Reconstruction and Its Applications in Private Federated Learning," *arXiv preprint arXiv:1812.00984*, 2018.
[31] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A Hybrid Approach to Privacy-Preserving Federated Learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, New York, NY, USA, 2019, pp. 1–11.
[32] "Openmined, pysyft." [Online]. Available: https://github.com/OpenMined/PySyft
[33] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining API calls," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, New York, NY, USA, 2010, pp. 1020–1025.
[34] "Eu general data protection regulation (gdpr)." [Online]. Available: https://gdpr.eu/
[35] "Apktool." [Online]. Available: https://ibotpeaches.github.io/Apktool/
[36] "Dedxer." [Online]. Available: http://dedexer.sourceforge.net/
[37] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA, 2014, pp. 1025–1035.
[38] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty Computation from Somewhat Homomorphic Encryption," in *Advances in Cryptology – CRYPTO 2012*, Berlin, Heidelberg, 2012, pp. 643–662.
[39] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," in *Proceedings on Privacy Enhancing Technologies*, vol. 2019, Jul. 2019, pp. 26–49.
[40] A. Shamir, "How to share a secret," in *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[41] "Tensorflow federated." [Online]. Available: https://www.tensorflow.org/federated.
[42] T. Ban, T. Takahashi, S. Guo, D. Inoue, and K. Nakao, "Integration of Multi-modal Features for Android Malware Detection Using Linear SVM," in *2016 11th Asia Joint Conference on Information Security (AsiaJCIS)*, 2016, pp. 141–146.
[43] V. N. Vapnik, "Statistical learning theory," Sep. 1998.
[44] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology — CRYPTO '91*, Berlin, Heidelberg: Springer, vol 576. 1992, pp. 420–432.
[45] Z. Tu, R. Li, Y. Li, G. Wang, D. Wu, P. Hui, L. Su, and D. Jin, "Your Apps Give You Away: Distinguishing Mobile Users by Their App Usage Fingerprints," in *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 3, Sep. 2018.