Nisarg Raval, Ashwin Machanavajjhala, and Jerry Pan

OLYMPUS: Sensor Privacy through Utility Aware Obfuscation

Abstract: Personal data garnered from various sensors are often offloaded by applications to the cloud for analytics. This leads to a potential risk of disclosing private user information. We observe that the analytics run on the cloud are often limited to a machine learning model such as predicting a user's activity using an activity classifier. We present OLYMPUS, a privacy framework that limits the risk of disclosing private user information by obfuscating sensor data while minimally affecting the functionality the data are intended for. OLYMPUS achieves privacy by designing a *utility aware obfuscation* mechanism, where privacy and utility requirements are modeled as adversarial networks. By rigorous and comprehensive evaluation on a real world app and on benchmark datasets, we show that OLYMPUS successfully limits the disclosure of private information without significantly affecting functionality of the application.

Keywords: machine learning, mobile, privacy

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

1 Introduction

A large number of smart devices such as smartphones and wearables are equipped with sophisticated sensors that enable fine-grained monitoring of a user's environment with high accuracy. In many cases, third-party applications running on such devices leverage this functionality to capture raw sensor data and upload it on the cloud for various analytics. Fine-grained collection of sensor data contains highly sensitive information about users: images and videos captured by cameras on smart phones could inadvertently include sensitive documents [42], and a user's health status could be inferred from motion sensor data [40, 51]. We observe that in many cases, sensor data that are shipped to the cloud are primarily used as input to a machine learning (ML) model (e.g. a classifier) hosted on the cloud. In this work, *we investigate whether we can avoid the disclosure of private sensor data to the cloud in such scenarios.*

Consider Alice who uses a *driver safety* app (e.g. CarSafe [50]) that helps her from distracted driving. The app uses a smartphone camera to record Alice, performs activity recognition on the cloud using a deep neural network (DNN) and alerts her when she is distracted. While she is comfortable allowing the app to monitor activities related to distracted driving (such as detecting whether she is drowsy or inattentive), she is not comfortable that the raw camera feed is uploaded to the cloud. She has no guarantee that the app is not monitoring private attributes about her such as her identity, race or gender. Our goal is to develop a mechanism that allows Alice to send as little information to the cloud as possible so as to (a) allow the *driver safety* app to work unmodified, (b) minimally affect the app's accuracy, while (c) providing her guarantee that app cannot monitor other attributes that are private to Alice.

We address the problem of *utility aware obfuscation*: design an obfuscation mechanism for sensor data to minimize the leakage of private information while preserving the functionality of the existing apps. Building upon prior work [18, 22], we formulate this problem as a game between an *obfuscator* and an *attacker*, and propose OLYMPUS, a privacy framework that uses generative adversarial networks (GAN) [20] to solve the problem. Unlike prior work, where utility is formulated using a closed form mathematical expression, OLYMPUS tunes the obfuscator to preserve the functionality of target apps. This ensures that the obfuscated data works well with the existing apps without any modifications.

In OLYMPUS, a user can specify utility using one or more apps whose functionality must be preserved. A user can specify privacy using a set of labeled examples on data collected from the sensors. Given this training data and access to the target app(s), OLYMPUS learns an obfuscation mechanism that jointly minimizes both privacy and utility losses. Moreover, if the private properties are correlated with the objective of the app's ML model, OLYMPUS allows the user to tradeoff between the privacy and utility losses (since they can no longer be simultaneously minimized). At runtime, OLYMPUS uses

Nisarg Raval: Duke University, E-mail: nisarg@cs.duke.edu
Ashwin Machanavajjhala: Duke University, E-mail: ashwin@cs.duke.edu
Jerry Pan: Duke University, E-mail: qiyuan.pan@duke.edu

the learned obfuscation mechanism to interact with the unmodified target apps.

Continuing our *driver safety* example, given access to the app and a few example images of people driving labeled by their identity, OLYMPUS learns to obfuscate the images such that it hides the identity of drivers while allowing the app to detect distracted driving.

We make the following contributions in this paper:

1. We design OLYMPUS, a privacy framework that solves the utility aware obfuscation problem wherein inputs to a machine learning model are obfuscated to minimize the private information disclosed to the model and the accuracy loss in the model's output.
2. To the best of our knowledge we present the first proof-of-concept implementation of a utility aware obfuscation mechanism, deploy it on a smartphone, and evaluate it against a real-world mobile app. Our evaluation on a real-world *handwriting recognition app* shows that OLYMPUS allows apps to run unmodified and limits exposure of private information by obfuscating the raw sensor inputs.
3. OLYMPUS allows users to tradeoff privacy and utility, when the two requirements are at odds with one another. We empirically demonstrate that OLYMPUS provides better privacy-utility tradeoffs than other competing techniques of image obfuscation.
4. OLYMPUS works across different data modalities. On image analysis tasks, we empirically show that OLYMPUS ensures strong privacy: the accuracy of an attacker (simulated as another ML model) trained to learn the private attribute in the obfuscated data was only 5% more than the accuracy of a random classifier (perfect privacy). For example, on a distracted driver detection dataset (StateFarm [6]) the attacker's accuracy of identifying drivers were 100%, 15.3% and 10% on unperturbed images, images perturbed by OLYMPUS and images perturbed by an ideal obfuscation mechanism, respectively. On the other hand, OLYMPUS suffers only a small loss in accuracy (<17%) across all image datasets. On motion sensor data, the accuracy of an attacker is slightly better, but no more than 13% higher than the accuracy of a random classifier, but with *no loss in accuracy* of the machine learning task.
5. We empirically show that OLYMPUS supports multiple target applications with a single obfuscation mechanism. We also demonstrate that OLYMPUS supports apps with different kinds of classifiers, namely, DNN and logistic regression.

The rest of the paper is organized as follows. Section 2 provides an overview of OLYMPUS along with its key design principles. Section 3 describes the methodology of learning the utility aware obfuscation. Section 4 provides the implementation details of training and deploying OLYMPUS on a smartphone to obfuscate data in real-time. We evaluate OLYMPUS on an Android app as well as on various real-world datasets, and present our results in Section 5. Finally, we summarize related work in Section 6 and conclude in Section 7.

2 OLYMPUS Overview

In this section, we begin with a description of the utility aware obfuscation problem. Then, we outline key design principles of OLYMPUS. Finally, we give an overview of our privacy framework.

2.1 Problem Setting

Our goal is to design a *utility aware obfuscation*: given a set of ML models U that take as input x, and a specification of private attributes in the input S, construct an obfuscation mechanism $\mathcal{M}$ such that the privacy loss and the accuracy loss are jointly minimized. We achieve this by developing a privacy framework – OLYMPUS, inspired by the idea of *adversarial games* [20]. In an adversarial game, two players compete against each other with conflicting goals, until they reach an equilibrium where no player can improve further towards their respective goals. Similarly, OLYMPUS constructs the obfuscation mechanism by competing against an attacker whose goal is to break the obfuscation.

Trust Model

We assume third party apps are *honest-but-curious* meaning they follow the protocol but may infer private information from the available data. We also assume that the device platform is trusted and properly isolated from the untrusted third party apps running on the devices. OLYMPUS runs as a part of the trusted platform on the device and intercepts all sensor data for obfuscation. The third party apps can not directly access the raw sensor data and have access to only obfuscated data.

Privacy Goals

Our privacy goal is to prevent *curious* third party apps from inferring private user information in the captured sensor data. To this end, we model the attacker as an ML adversary who has a complete access to the obfuscation mechanism. Thus, it can train on the obfuscated data (generated by OLYMPUS) to undo the obfuscation. On the other hand, OLYMPUS learns the obfuscation mechanism by competing against such ML attackers.

Rather than considering all powerful attackers, we limit the ML attacker to be one from a known hypothesis class (e.g. a DNN with a known structure). We believe this is a reasonable assumption since (a) state-of-the-art attacks on obfuscation methods are performed using DNN [34, 35], and (b) being a universal approximator [46], DNN is successfully used to learn many complex functions, making it an apt tool to model a strong adversary. In Section 3.1, we mathematically model attackers and formalize our privacy goals as an optimization problem. In Section 3.2, we describe how OLYMPUS achieves these privacy goals by solving the optimization problem. To verify how well OLYMPUS achieves the proposed privacy goals, we empirically evaluate OLYMPUS against a suit of attackers, namely DNN, logistic regression, random forest and SVM (Section 5.5).

2.2 Design Principles

A trivial solution to protect user data would be to run the classifier on a user's device which would not require apps to send raw sensor data to the cloud. There are several issues with this approach. First, the output of the classifier may itself leak information that a user may deem private. Second, it requires app developers to modify the app so that it can run the ML task on a user's device. This may be infeasible due to proprietary reasons or the limited availability of resources on a mobile device. To address above concerns we follow the following important principles in designing OLYMPUS.

2.2.1 Compatible with Existing Apps

Our primary objective is to develop an obfuscation mechanism that would let apps run unmodified. Most of the previous approaches of data obfuscation do not consider the target apps in designing the mechanism [18, 21, 41]. This could lead to the obfuscated data being incompatible with the app, or cause a significant loss in the app's utility. For instance, we show that the standard approaches of protecting visual secrets such as blurring, perform poorly against image classification.

To alleviate this issue, we allow users to specify their utility requirements in the form of apps. The onus is on the obfuscation mechanism to ensure that the specified apps work well with the obfuscated data. Specifying the utility requirements in terms of apps is not only easy and intuitive but also provides a natural way to quantify utility guarantees in terms of the accuracy of the specified apps on the obfuscated data. Moreover, we design OLYMPUS such that a single obfuscation mechanism is sufficient for multiple apps.

2.2.2 Privacy-Utility Tradeoff

In certain scenarios, the app-essential information is tightly coupled with users' private information. A trivial example is inferring the driver's activity reveals the fact that she is in the car. Thus, often it is impossible to achieve complete privacy without compromising the functionality of the app and vice-a-versa. We achieve this crucial balance between privacy and utility through modeling the obfuscation problem as a minimax optimization. OLYMPUS allows users to control the privacy-utility tradeoff by specifying a parameter (λ) that governs the relative importance of privacy and utility.

2.2.3 Holistic Obfuscation

In many cases, hiding only private information may not be enough to protect users' privacy. For instance, researchers have shown that blurring faces is not enough to conceal the identity of a person since it is correlated with seemingly innocuous attributes such as clothing or environment [34, 35]. Hence, instead of explicitly detecting and removing the private attribute (e.g. face), we take a holistic approach to obfuscation. By formulating the obfuscation problem as an adversarial game, we aim to minimize the information leakage caused by any such correlation in the data.

2.2.4 Data Agnostic Framework

Many methods of protecting secrets are specifically designed for certain types of private attribute [11, 26, 28] or data types [12, 23, 24]. We aim to provide a general privacy framework that is agnostic to the private

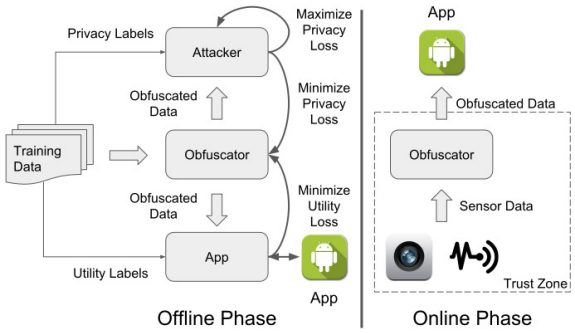


Fig. 1. OLYMPUS framework

attribute as well as the data types. OLYMPUS comprises of multiple DNNs that are responsible for manipulating sensor data. Thus, one can use OLYMPUS for any data type by plugging-in an appropriate DNN. By simply changing the underlying networks, we show that OLYMPUS can be used for protecting private information in images and in motion sensor data.

2.3 Privacy Framework

As outlined in Figure 1, OLYMPUS has two phases – an *offline phase* to learn the obfuscation mechanism, and an *online phase* where it uses the learned mechanism to obfuscate the sensor data. Before describing these phases, we describe the modules of OLYMPUS.

2.3.1 Modules

OLYMPUS consists of three modules – 1) an *App* that simulates the target app, 2) an *Attacker* that attempts to break the obfuscation, and 3) an *Obfuscator* that learns to obfuscate data to protect private information without affecting the functionality of the target app.

***App*:** The *App* module simulates the target app to verify that the obfuscated data preserve the functionality of the app, i.e., satisfy the utility requirements. The target app can be any classifier that takes sensor data as input and outputs a classification probability for each class. The inputs to the *App* module are the obfuscated data and the associated utility labels. The utility labels are the true class labels of the obfuscated data and are used in computing the utility loss. Informally, the utility loss is a classification loss that captures the performance of the target app on the obfuscated data.

The *App* module in the *driver safety* example uses obfuscated driver images and their activity labels as inputs. It uses the target app to get the classification probability for each driver image and computes the corresponding utility loss using the true activity labels.

***Attacker*:** The *Attacker* module simulates an adversary that attacks the obfuscation mechanism learned by the *Obfuscator*. In other words, it verifies that the obfuscated data hide private attributes, i.e., satisfy the privacy requirements. The inputs to the *Attacker* are a set of obfuscated images and their privacy labels, where a privacy label specifies the value (class) of the private attribute. Using these inputs, the *Attacker* trains a DNN to classify obfuscated images into correct privacy labels, and outputs the privacy loss. Intuitively, the privacy loss measures how well the *Attacker* classifies obfuscated images, i.e., infers the private attributes.

In our *driver safety* scenario, given obfuscated images together with their privacy labels (identity of drivers), the *Attacker* learns to identify drivers in the obfuscated images.

***Obfuscator*:** The *Obfuscator* module learns a transformation of the data such that the obfuscated data do not contain any private information but preserve useful information. The corresponding privacy and utility losses are estimated using the *Attacker* and the *App* modules, respectively. To satisfy the privacy and utility requirements, the *Obfuscator* learns a transformation function that minimizes both the privacy and utility losses.

The design of the *Obfuscator* follows the architecture of an autoencoder [49]. It consists of a sequence of layers, where both input and output layers are of the same size, while the middle layer is much smaller in size. The idea is to encode the input to this compact middle layer and then decode it back to the original size in the output layer. This encoding and decoding process is learned by minimizing the privacy and utility losses of the output (obfuscated data). The smaller size of the middle layer forces the *Obfuscator* to throw away unnecessary information while preserving the information required to reconstruct the output. Essentially, the privacy loss forces the *Obfuscator* to throw away private information, while the utility loss forces it to preserve the information required by the target app.

In the *driver safety* use case, the *Obfuscator* obfuscates the input image such that the features required to identify drivers are removed, but the features important for activity detection are preserved.

2.3.2 Offline Phase

In the offline phase, OLYMPUS learns an obfuscation mechanism that minimizes information leakage while maintaining the utility of the applications. The user specifies privacy and utility requirements by providing training data that includes examples with their respective privacy and utility labels. It is crucial to protect the training data as it contains private information. Note that the training data is only used during an offline phase and it never leaves the OLYMPUS system. Thus, we require the offline phase to be executed in a secure environment, either on the device or on the trusted server. Moreover, the target app always receives the obfuscated data, even during the training phase.

Given the training data, the offline phase learns an obfuscation function $\mathcal{M} : X \rightarrow X$ such that $\mathcal{M}(\cdot)$ satisfies given privacy and utility requirements. Here, X refers to the domain of the data. In the next section, we provide an algorithm to learn $\mathcal{M}$.

2.3.3 Online Phase

During the online phase, OLYMPUS simply uses the learned obfuscation $\mathcal{M}$ to obfuscate the data. In particular, when an app requests sensor data, OLYMPUS intercepts the request and obfuscates the data using $\mathcal{M}$ before sending it to the app.

3 Utility Aware Obfuscation

In this section, we mathematically formulate the problem of designing an obfuscation function $\mathcal{M}$ that satisfies given privacy and utility requirements, and describe how OLYMPUS learns $\mathcal{M}$ using adversarial games.

3.1 Problem Formulation

Let X be the domain of the data that need to be obfuscated (e.g., images, sensor readings, etc.). Let s denote a private attribute associated with each $x \in X$ (e.g., race, gender, identity, etc.). We use the notation $x.s = z$ to denote the fact that the private attribute associated with x takes the value z. Here, z is a *privacy label* of x and it takes a value from a set Z_s which we denote as *privacy classes*. For instance, if s is gender then Z_s = {male, female}, and $x.s$ = female implies that the gender of x is female.

Similarly, u denotes a utility attribute associated with each $x \in X$ (e.g., activity, expression, etc.). Again, we denote by $x.u = y$ the fact that the utility attribute associated with x takes the value y. Here, y is a *utility label* of x and it takes a value from a set Y_u which we denote as *utility classes*. For example, if u is activity then Y_u = {walking, running, ... }, and $x.u$ = walking implies that the activity associated with x is walking.

Let $D_{us} = \{(x_i, y_i, z_i)\}_{i=1}^{n}$ be a training dataset consisting of n examples drawn i.i.d. from a joint distribution P_{us} over random variables $(\mathcal{X}, \mathcal{Y}_u, \mathcal{Z}_s)$. Here, $y_i \in Y_u$ and $z_i \in Z_s$ are the corresponding utility and privacy labels of $x_i \in X$. Let $\mathcal{M} : X \rightarrow X$ denote a deterministic obfuscation function and $\mathcal{H}$ denote the hypothesis space of all obfuscation functions. Next, we formalize the privacy and utility requirements of $\mathcal{M}$.

Privacy Requirements

The privacy requirement is to protect the private attributes in the obfuscated data. As mentioned earlier in Section 2.1, the adversary is an ML model that learns to identify private attributes in the obfuscated data. The perfect privacy (i.e., the ideal obfuscation) is achieved when the attacker cannot perform better than randomly guessing the privacy labels, i.e., when the attacker's probability of predicting the correct privacy label is $1/|Z_s|$ for the private attribute s. Thus, we measure privacy loss in terms of how well the attacker performs over random guessing.

Let $\mathcal{C}_s : X \rightarrow [0,1]^{|Z_s|}$ be an attacker (classifier) that predicts the privacy label $z = x.s$ given $\mathcal{M}(x)$ as input. The output of $\mathcal{C}_s$ is a probability distribution p_s over privacy labels Z_s, where $p_s(\mathcal{M}(x))[z_i]$ is the predicted probability of $\mathcal{M}(x)$ having privacy label z_i, i.e., $Pr(z = z_i|\mathcal{M}(x))$. For a given s, the privacy loss of the mechanism $\mathcal{M}$ with respect to an attacker $\mathcal{C}_s$ can be measured using the cross entropy loss as follows.

$$LP(\mathcal{M}, \mathcal{C}_s) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{z \in Z_s}\frac{1}{|Z_s|}\log[p_s(\mathcal{M}(x_i))[z]] \quad (1)$$

The above privacy loss essentially measures the difference between two probability distributions, namely a uniform distribution (random guessing) and the distribution predicted by the attacker. The privacy loss increases as the probability of predicting the correct privacy label diverges from the uniform distribution. Thus,

minimizing the above privacy loss ensures that the adversary cannot perform better than random guessing.

Utility Requirements

The utility requirement is to preserve the utility attributes in the obfuscated data that are required by the target apps. Let $\mathcal{C}_u : X \rightarrow [0,1]^{|Y_u|}$ represent an app classifier that predicts the utility label $y = x.u$ given x as input. Given an obfuscated input $\mathcal{M}(x)$, the output of $\mathcal{C}_u$ is a probability distribution p_u over utility labels Y_u, where $p_u(\mathcal{M}(x))[y_i]$ is the predicted probability of $\mathcal{M}(x)$ having utility label y_i, i.e., $Pr(y = y_i|\mathcal{M}(x))$. To measure the impact of the obfuscation on the performance of the classifier, we measure the classification error using the following utility loss function.

$$LU(\mathcal{M}, \mathcal{C}_u) = -\frac{1}{n}\sum_{i=1}^{n} \log[p_u(\mathcal{M}(x_i))[y_i]] \qquad (2)$$

Here, y_i is the true utility label of x_i. The above utility loss is the cross entropy loss that is 0 when the classifier correctly predicts the utility labels, and increases as the classification error increases. Thus, minimizing the above utility loss ensures that the functionality of the target app is preserved.

Obfuscation Mechanism

Our aim is to design an obfuscation mechanism $\mathcal{M}$ that 1) hides the private attribute (s) of the data and 2) preserves the utility attribute (u) required by the target app. This can be achieved by designing $\mathcal{M}$ that minimizes both the privacy loss (LP) and the utility loss (LU) defined above. Minimizing the utility loss ensures that the obfuscated data works well with the target app's classifier $\mathcal{C}_u$. However, minimizing the privacy loss only protects the private attribute against a particular attacker $\mathcal{C}_s$. To ensure that the obfuscation mechanism protects against a class of attackers (e.g., DNN, SVM, etc.), we minimize the maximum privacy loss across all attackers belonging to a particular class. Thus, the optimal obfuscation mechanism is a solution to the below minimax optimization problem.

$$\arg\min_{\mathcal{M}\in\mathcal{H}} \lambda LU(\mathcal{M}, \mathcal{C}_u) + (1-\lambda)[\max_{\mathcal{C}_s\in\mathcal{H}_s} LP(\mathcal{M}, \mathcal{C}_s)] \qquad (3)$$

Here, $\mathcal{H}_s$ is a hypothesis space of attacker models. For example, in case of an SVM adversary, $\mathcal{H}_s$ is a set of all hyperplanes and $\mathcal{C}_s \in \mathcal{H}_s$ is a particular hyperplane learned by SVM. Even though the size of $\mathcal{H}_s$ is usually very large, we show that the optimal attacker (one with the maximum privacy loss) can be computed using various optimization techniques such as gradient descent.

We control the privacy-utility tradeoff using a hyper-parameter $\lambda \in [0,1]$. If we set $\lambda = 0$, then the utility loss is ignored, and the optimal mechanism achieves uniform conditional distribution, ensuring no information about the private attribute are leaked. An example of such an obfuscation mechanism is a constant function, i.e., $\mathcal{M}(x) = c, \forall x \in X$, where c is some constant. On the other hand, if we set $\lambda = 1$, then the privacy loss is completely ignored, and the optimal obfuscation mechanism is the identity function, i.e., $\mathcal{M}(x) = x, \forall x \in X$. In Section 5.6, we show that by setting an appropriate value of λ, we can achieve the desired trade-off between privacy and utility.

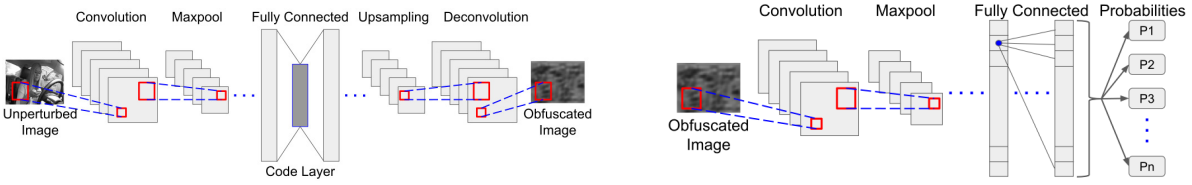
Next, we extend our formulation to design an obfuscation mechanism that protects multiple private attributes against multiple target apps. Let S denote a set of private attributes that user wants to protect and U denote a set of utility attributes required by the target apps. To design an obfuscation mechanism that protects all the private attributes in S while preserving all the utility attributes in U, we modify the above optimization function as follows.

$$\begin{aligned}\arg\min_{\mathcal{M}\in\mathcal{H}} \lambda[\max_{u\in U} LU(\mathcal{M}, \mathcal{C}_u)] + \\ (1-\lambda)[\max_{s\in S}\max_{\mathcal{C}_s\in\mathcal{H}_s} LP(\mathcal{M}, \mathcal{C}_s)]\end{aligned} \qquad (4)$$

Here, we measure the max privacy loss across all the private attributes ($\max_{s\in S}$) to minimize the maximum expected information leakage across all the private attributes. One can easily replace the max with a sum (average) to minimize the total (average) privacy loss across all the private attributes. Similarly, one can also consider the total/average utility loss instead of the max utility loss. Next, we present an iterative learning algorithm to solve the above optimization problem.

3.2 Learning to Obfuscate

So far, we have formulated the problem of designing an obfuscation mechanism as a minimax optimization. In order to solve this optimization, we use the concept of *adversarial nets* [20], originally proposed for learning an unknown data distribution using samples from that distribution. The problem of learning the distribution was formalized using a minimax optimization which was solved via training a pair of networks in an adversarial



(a) *Obfuscator* **(b)** *Attacker*

Fig. 2. OLYMPUS architecture for image data.

Algorithm 1 Obfuscation learning

Require: D, $P_u(\cdot, \theta_u)$, $P_s(\cdot, \theta_s)$, $M(\cdot, \theta_M)$, λ
Ensure: Optimal obfuscation mechanism $\mathcal{M}^*$
  Randomly initialize θ_s and θ_M
  for number of training iterations **do**
    for each minibatch $(X, Y, Z) \in D$ **do**
      ▷ Train *Attacker*
      Generate obfuscated data $M(X, \theta_M)$
      Update θ_s using equation (5)
      ▷ Train *Obfuscator*
      Update θ_M using equation (6)
    end for
  end for
  $\mathcal{M}^* = M(\cdot, \theta_M)$

fashion. We adapt this training approach to learn the optimal obfuscation mechanism $\mathcal{M}^*$.

The method for learning the obfuscation mechanism is outlined in Algorithm 1. For simplicity, we describe the learning algorithm for the case where a user wants to protect a single private attribute against one target app. It can be easily extended to support multiple private attributes with multiple applications.

Let M denote the *Obfuscator* network that takes as input $x \in X$ and outputs the obfuscated data $M(x, \theta_M)$, where θ_M is the parameter of the *Obfuscator* network. Let P_s denote the *Attacker* network that learns to predict the privacy labels from the obfuscated data. It takes as input the obfuscated data and outputs the probability distribution $p_s(M(x, \theta_M), \theta_s)$ over privacy labels Z_s, where θ_s is the parameter of the *Attacker* network. Similarly, P_u denotes the *App* network that takes as input the obfuscated data and outputs the probability distribution $p_u(M(x, \theta_M), \theta_u)$ over utility labels Y_u, where θ_u is the parameter of the *App* network.

OLYMPUS uses an iterative optimization algorithm to jointly train the *Obfuscator* and the *Attacker* networks as follows[1]. On each iteration, we alternate the training between the *Attacker* and the *Obfuscator*. First, we generate obfuscated data using the *Obfuscator* and train the *Attacker* to classify the obfuscated data into correct private labels. Formally, we train the *Attacker* network using the following objective function.

$$\arg\max_{\theta_s} -\frac{1}{n}\sum_{i=1}^{n} \log[p_s(M(x_i, \theta_M), \theta_s)[z_i]] \tag{5}$$

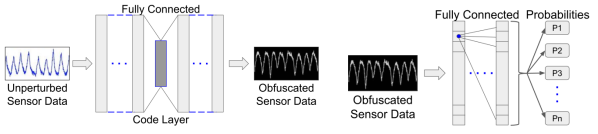
Here, z_i is the true privacy label of x_i. The above loss captures the attacker's ability to infer private attributes in the obfuscated data. It finds the attacker with a highest success rate in breaking the obfuscation mechanism. Note that maximizing the privacy loss (LP) defined in Equation (1) may have multiple worst case attackers. By maximizing Equation (5), we pick the worst case attacker that not only outputs a probability distribution that is far from the uniform distribution, but also correctly predicts the private attributes.

Next, we train the *Obfuscator* to minimize the privacy and utility losses computed using the *Attacker* and the *App* networks, respectively. Formally, the *Obfuscator* is trained via the following objective function.

$$\begin{aligned}\arg\min_{\theta_M} &-\lambda\frac{1}{n}\sum_{i=1}^{n} \log[p_u(M(x_i, \theta_M), \theta_u)[y_i]] \\ &-(1-\lambda)\frac{1}{n}\sum_{i=1}^{n}\sum_{z\in Z_s}\frac{1}{|Z_s|}\log[p_s(M(x_i, \theta_M), \theta_s)[z]]\end{aligned} \tag{6}$$

Here, the first term corresponds to the utility loss (LU) and the second term corresponds to the privacy loss (LP). We train both the networks until they reach an equilibrium, i.e., when the *Attacker* cannot perform better than random guessing. Since we are alternating the training between the *Attacker* and the *Obfuscator*, together equations (5) and (6) give a solution to the minimax optimization problem defined in Equation (3). Thus, at the equilibrium, the final *Obfuscator* $M(\cdot, \theta_M^*)$ gives the optimal obfuscation mechanism $\mathcal{M}^*$.

1 We do not train the *App* network (i.e., learn θ_u) as it simulates the pretrained classifier of the target app.



(a) *Obfuscator* **(b)** *Attacker*

Fig. 3. OLYMPUS architecture for motion sensor data.

4 Implementation

We developed a prototype implementation of OLYMPUS, deployed it on a Nexus 9 tablet, and evaluated on an Android app. The implementation of OLYMPUS consists of three steps – 1) instantiating its underlying modules, 2) training the *Obfuscator* (offline phase), and 3) deploying the *Obfuscator* on device (online phase).

4.1 Constructing OLYMPUS

Constructing OLYMPUS involves materializing its underlying modules – *App*, *Attacker* and *Obfuscator*. The *App* module uses the classifier from the target app. OLYMPUS supports any app as long as its classifier (a) outputs probability scores, (b) uses known loss function that is continuous and differentiable, and (c) uses raw sensor data as inputs. Any app that uses DNN for classification is supported by OLYMPUS since all of the above assumptions are true for DNN. The trend in ML is that most classifiers are gravitating towards DNNs because they achieve higher accuracy, and jointly perform feature computation and classification. In case of traditional classifiers, many of them satisfy assumptions (a) and (b) (e.g. Logistic Regression which we use in our experiments). Assumption (c) is not necessarily true as some classifiers involve arbitrarily complex feature computation steps that are hard to reason about. However, again, the trend is to use the last layers of standard DNNs to compute features, which can be readily integrated with OLYMPUS.

Both *Attacker* and *Obfuscator* are constructed using DNNs. The architecture of these networks depends on the type of the sensor data. For images, we use convolutional neural network (CNN) [30], while for motion sensor data we use DNN with fully connected layers. Figure 2 and 3 outline the architecture of the underlying networks of OLYMPUS for image and motion sensor data, respectively. Due to space constraints, we provide more details about the networks in Appendix C.

4.2 Training OLYMPUS

While training, OLYMPUS requires access to an app's classifier to ensure that the performance of the classifier is not hindered by the obfuscation. The exact process of accessing the classifier varies based on the target app. For instance, many apps use TensorFlow [10] for classification tasks such as activity and speech recognition [4, 5]. Using TensorFlow, an app developer can either train a custom classifier or use a pre-trained classifier from a public repository of models [8]. The trained classifier can either be embedded in the app for on-device classification or it can be hosted on the cloud and accessed via APIs. If the classifier is embedded in the app, it can be easily extracted and used by OLYMPUS during the training phase. On the other hand, for cloud-based classification, OLYMPUS can use the APIs to query the classifier to retrieve class probabilities.

As mentioned before (Section 2.1), our attacker model assumes that the third-party apps are honest-but-curious and they follow the protocol to ensure OLYMPUS learns the optimal obfuscation mechanism. More specifically, the target app must provide correct classification scores for the obfuscated data during the training phase. We argue that it is in the interest of the app to cooperate with OLYMPUS, otherwise it may impact the functionality of the app due to incorrect estimation of the utility loss. Note that the privacy is enforced via an independent attacker (not controlled by the app) making it harder for an uncooperative app to affect privacy guarantees. Learning an optimal obfuscation under adversarial setting (where the target app provides malicious labels) is an interesting future work.

OLYMPUS also requires appropriate training data to train the *Obfuscator*. The training data consists of several examples with their utility and privacy labels. For standard requirements (such as protecting identity), one can easily use publicly available benchmark dataset (e.g. celebA [31]) that fits the criteria. In cases when this is not feasible, the user needs to capture the dataset with appropriate labels. One can think of a data collection app that helps user collect appropriate data for training OLYMPUS. For instance, we developed a mobile app that allows users to collect training data for training OLYMPUS to hide identity of the writer while preserving the ability to recognize the handwritten text. More details about the app and the case study of the handwriting recognition app are given in Section 5.1.

Given the appropriate training data and an access to the app's classifier, we train the *Obfuscator* using Algorithm 1 as described in Section 3.2. In our *driver*

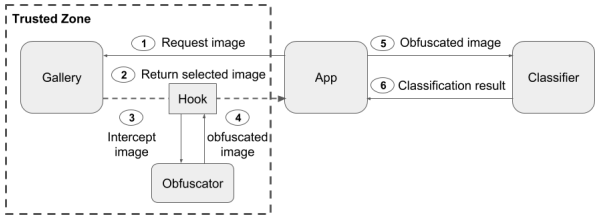


Fig. 4. An illustration of how OLYMPUS intercepts and obfuscates images requested by the target app *Classify*.

safety use case, the training data consists of several images of drivers with their activities (utility labels) and identities (privacy labels). Using these training data and the app, OLYMPUS learns the obfuscation that hides driver's identity by minimizing the attacker's ability to identify drivers in the obfuscated images. At the same time, it ensures that the obfuscated images preserve the activity by maximizing the app's accuracy of identifying the activity in the obfuscated images. To improve usability, we envision a repository of pretrained obfuscators that users can simply use to protect their privacy against various third party apps.

4.3 Deploying OLYMPUS

OLYMPUS intercepts and obfuscates sensor data before they are received by the target app. This is achieved by instrumenting Android OS using *Xposed* [43], an open source framework that provides means to alter the functionality of an Android app without modifying the app. Xposed provides APIs to intercept method calls, modify method arguments, change the return values, or replace the method with custom code. Using these APIs, we built an *Xposed module* to intercept an app's request to sensor data and apply obfuscation on-the-fly before the requested data reach the app.

Since Android provides standard APIs to access the sensor data, one can easily hook those API calls to apply the obfuscation. Consider an Android app – *Classify* that allows a user to select an image from the Gallery and finds objects in the selected image using a classifier. It launches the Gallery app by invoking the *startActivityForResult* method for the user to select an image. Upon selection, the Gallery app wraps the selected image in a special class called *Intent* and returns it to the Classify app via the *onActivityResult* method. The *Classify* app then extracts the image from the received *Intent* and sends it to the classifier to detect objects.

Figure 4 demonstrates how OLYMPUS obfuscates images requested by the *Classify* app. Since the app gets image data through the *onActivityResult* method, OLYMPUS hooks this method using the APIs provided by Xposed. The installed hook intercepts every call to the *onActivityResult* method, runs the *Obfuscator* on the image selected by the user and returns the resulting obfuscated image to the *Classify* app. Thus, the *Classify* app always receives an obfuscated image.

Similarly, one can apply appropriate hooks to obfuscate the sensor data based on the target app. Note that this approach does not require any modifications to the app. However, we need to install appropriate hooks for each target app. In the future, we envision the OLYMPUS framework to be a part of the operating system that seamlessly performs obfuscation on sensor requests based on the target apps.

5 Experiments

In this section, we evaluate OLYMPUS and compare it with the existing approaches of protecting secrets. Through rigorous empirical evaluation, we seek answers to the following questions on an Android app (Qns 1-3) as well as on benchmark datasets (Qns 1-6).

1. What is the impact of obfuscation on the functionality of the target app?
2. How well does the obfuscation protect private information?
3. What is the overhead of obfuscating sensor data?
4. How well does the obfuscation mechanism tradeoff privacy for utility compared to existing approaches?
5. How well does the obfuscation mechanism scale with multiple applications?
6. How well does the obfuscation mechanism perform against different kinds of app classifiers?

5.1 Experimental Setup

In this section, we describe the target app, benchmark datasets, and metrics we used to evaluate OLYMPUS.

Android App Case Study: Handwriting Recognition

Consider a following use case: Alice wants to use a mobile app that transcribes text as she writes on the device. However, she does not want to reveal her handwriting style to the app, as that could reveal private at-

Dataset	Data Type	Target App	Private Information	#Utility Classes	#Privacy Classes	Training time (s)	Obfuscation time (ms)
KTH	Image	Action recognition	Identity of people	6	6	862	0.15
StateFarm	Image	Distracted driving detection	Identity of people	10	10	607	0.11
CIFAR10	Image	Object recognition	Face	10	2	3384	0.04
HAR	Inertial sensors	Action recognition	Identity of people	6	30	173	0.05
OPPORTUNITY	Motion sensors	Action recognition	Identity of people	5	4	7222	0.03

Table 1. Summary of benchmark datasets used for evaluating OLYMPUS. The training time shows the mean time (in seconds) to learn the obfuscation mechanism while the obfuscation time shows the mean time (in ms) to obfuscate one sample on a GPU.

tributes like her identity [52] or personality [48]. Hence, she wants to use OLYMPUS to obfuscate the app's input.

Motivated by this scenario, we evaluate OLYMPUS on a *handwritten digit recognition app – DL4Mobile* [2] downloaded from the Google Play Store. *DL4Mobile* allows users to draw a digit between 0 to 9 and recognizes it using a DNN. Our goal is to learn an obfuscation mechanism that protects the writer's identity while allowing *DL4Mobile* to correctly classify the written digits. To obtain training data with utility labels (digit) and privacy labels (user identity), we developed an Android app to collect images of handwritten digits. Using this app, we collected data from two users with 30 images per person per digit.

We consider two variants of *DL4Mobile*, the first performs *on-device classification* and the other performs classification *on the cloud*. The original app uses an embedded DNN to perform on-device classification using the TensorFlow library [7]. We modified the original app to create a variant that uses the same DNN to perform classification on the cloud using the Google Cloud API [3][2]. Since in both the variants the target classifier is the same, we train a single obfuscation mechanism for both. We use the DNN architectures described in Section 4.1 for the *Attacker* and the *Obfuscator* networks.

We evaluate OLYMPUS's obfuscator on a 10% sample of the examples (held out as a test set) through the *DL4Mobile* app instrumented using Xposed. As explained in Section 4.3, we install hooks in both variants of the *DL4Mobile* app to intercept the original digit image and replace it by the corresponding obfuscated image via the *Obfuscator*.

Benchmark Datasets

We also evaluate OLYMPUS on three image datasets and two motion sensor datasets. Table 1 gives the summary of all the datasets along with their privacy and utility requirements. Due to space constraints, the detailed description of the datasets is provided in Appendix A.

For each dataset, we use OLYMPUS to learn an obfuscation mechanism that protects respective private information (e.g. faces, identity of people, etc.) while preserving the ability to perform a specified classification task (e.g. object recognition, action recognition, etc.). We construct OLYMPUS based on the type of the dataset as explained in Section 4.1. We use the same architecture across datasets except minor modifications required for handling the data (e.g. input size). The complete specification of all the networks is available in Appendix C.

For each dataset, we simulate the target app by training a DNN to classify unperturbed data into corresponding utility labels. We randomly split each dataset into train, validation and test set, with 80%, 10% and 10% splits, respectively. We train OLYMPUS for 100 iterations (epochs) using the training set, use the validation set to choose the best obfuscation model, and evaluate it using the test set.

All the networks are implemented in Keras [16] and are trained on a NVIDIA Tesla K80 GPU. For training, we used the Adam optimizer [27] with a learning rate of 0.001 and fixed the value of λ to 0.5. All reported results are averaged over 10 independent runs.

Evaluation Metrics

We used the following metrics to evaluate OLYMPUS.

5.1.1 Privacy

The privacy is measured in terms of the accuracy of an attacker who attempts to infer private information in the obfuscated data. It is a common practice to provide privacy guarantees relative to perfect privacy. In our case, the perfect privacy is achieved when the attacker cannot perform better than random guessing. The accuracy of random guessing is $\frac{1}{\#\text{ privacy classes}}$, which differs for different applications. So we decided to measure privacy loss in terms of how much better an adversary is

2 The source code of *DL4Mobile* is available here – https://github.com/nalsil/TensorflowSimApp.

User	Digits										Utility	Privacy	Execution Time (ms)	
	0	1	2	3	4	5	6	7	8	9			On-Device	On-Cloud
User A											0.94 (±0.009)	0.32 (±0.04)	36.2 (±0.9)	338.3 (±62.14)
User B														
User A											0.93 (±0.04)	0.01 (±0.02)	44.48 (±0.9)	346.6 (±60.1)
User B														

Table 2. Evaluation results on *DL4Mobile* without OLYMPUS (first two rows) and with OLYMPUS (last two rows). Utility is the digit classification accuracy of the app classifier. Privacy is the attacker's accuracy to identify users measured in terms of the improvement over random guessing.

compared to random guessing. Given a set of privacy classes Z, we define the attacker's score as follows.

$$\left| \frac{\#\text{ samples correctly classified by the attacker}}{\text{total number of samples}} - \frac{1}{|Z|} \right| \tag{7}$$

The range of the attacker's score is from 0 (perfect privacy) to $(1 - \frac{1}{|Z|})$ (no privacy).

5.1.2 Utility

We measure the utility of our obfuscation mechanism in terms of the classification accuracy of the target app. The utility score is defined as follows.

$$\text{utility score} = \frac{\#\text{ samples correctly classified by the app}}{\text{total number of samples}} \tag{8}$$

Hence, the utility score ranges from 0 (no utility) to 1 (highest utility). Unlike privacy, relative utility score (app's accuracy on unperturbed data - accuracy on perturbed data) is not a suitable metric in our case as it may lead to a negative utility score, making it harder to explain the results. Sometimes OLYMPUS learns better features which results in higher accuracy on the perturbed data than on the unperturbed data. In particular, we observed this phenomenon when the app's classifier is a weaker model (for example see Table 4).

5.1.3 Overhead

We evaluate the efficiency of our mechanism by reporting the time to obfuscate the sensor data.

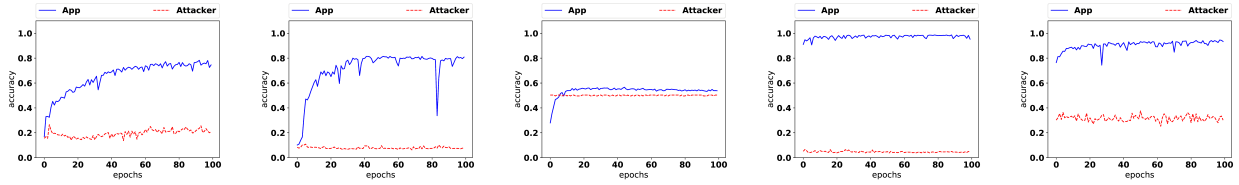
5.2 Evaluation on Android App

We evaluate OLYMPUS on *DL4Mobile* using a Nexus 9 tablet running Android 7.1 (Nougat) and Xposed version 88. We compare our results by running *DL4Mobile* on an unmodified Android OS. Table 2 summarizes the results of evaluating *DL4Mobile* without OLYMPUS (first two rows) and with OLYMPUS (last two rows).

Without OLYMPUS, the adversary's accuracy of correctly predicting user's identity from the handwritten digits is 82% (64% improvement over random guessing) which shows that handwriting contains unique patterns that can be exploited to identify users. With OLYMPUS, the adversary's accuracy of identifying user's identity drops to 51% which is close to random guessing (50%) as there are only two users in the dataset. On the other hand, OLYMPUS incurs a minor drop (1%) in digit classification accuracy compared to the accuracy on the unperturbed images. These results show that OLYMPUS successfully protects users' identity without affecting the functionality of *DL4Mobile*.

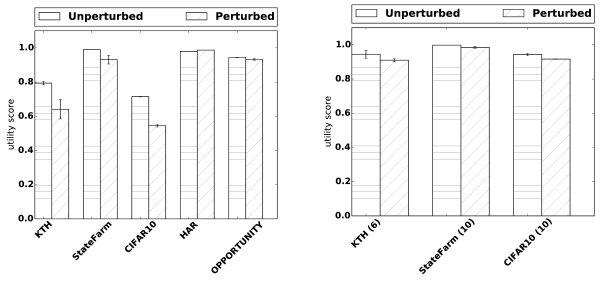
The high privacy and high utility come at a small cost of obfuscating the image. The average time to train the *Obfuscator* was 718 seconds on a GPU. The mean time to obfuscate an image on Nexus 9 was 8.28 ms resulting in 22.9% and 2.4% overhead when classifying images on-device and on the cloud, respectively.

Table 2 also shows a randomly selected digit sequence from each user with the corresponding obfuscation. In many cases, the same digit is drawn differently by each user which makes it easy for the adversary to identify a user based on her digits. However, the obfuscated images of the same digit across different users look very similar, making it harder for the adversary to identify the user. On the other hand, the obfuscated images across different digits are different, allowing the classifier to easily distinguish different digits. Even though the obfuscated images are visually very different from the unperturbed images, the classifier (trained only on unperturbed images) can still classify them with high accuracy. Note that we do not re-train the *DL4Mobile* classifier to recognize digits in the obfuscated images. Instead, the *Obfuscator* learns to obfuscate images such



(a) KTH **(b)** StateFarm **(c)** CIFAR10 **(d)** HAR **(e)** OPPORTUNITY

Fig. 5. Accuracy of *App* (in blue) and *Attacker* (in red) networks on obfuscated data while training the *Obfuscator*. OLYMPUS quickly learns to hide private information while preserving useful information.



(a) Single target app **(b)** Multiple target apps

Fig. 6. Classification accuracy of the target apps on unperturbed and perturbed data. In Figure 6b, we report the accuracy averaged over all target apps and the values in the bracket indicate the number of target apps.

that it preserves features required by the *DL4Mobile* classifier for recognizing digits. Thus, the target app is able to complete the classification task with high accuracy on the obfuscated image without retraining.

5.3 Evaluation on Benchmark Datasets

Figure 5 shows the accuracy of the *App* and the *Attacker* networks while training the *Obfuscator*. In the initial stage of learning, the output of the *Obfuscator* is somewhat random due to random initialization of network parameters. Hence, the accuracy of both the networks are low. As the training proceeds, the *Obfuscator* learns to preserve information that is useful for the target app while hiding the private information. Over time, the accuracy of the *App* network increases and saturates at some point across all the datasets. On the other hand, the accuracy of the *Attacker* remains low due to the obfuscation of the private attributes. This shows that the output of the *Obfuscator* is somewhat private to begin with due to the random initialization. All that *Obfuscator* needs to learn is to produce the output with the utility attributes that are required by the target application. Of course, the privacy loss ensures that the *Obfuscator* should not leak any private information in the output while learning to preserve the utility attributes.

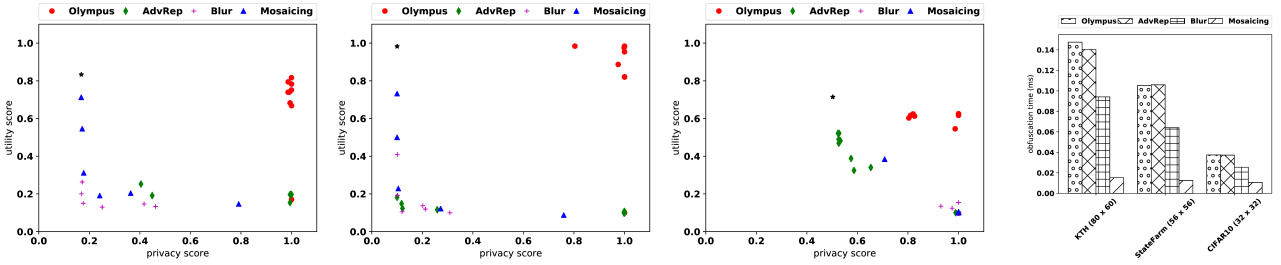
In Table 1, we report the training time (in seconds) and the obfuscation time (per sample in ms) averaged over 10 independent runs on a GPU. The training time varies from 2 minutes to a little over 2 hours while the obfuscation time across all the datasets is always under 1 ms. The high variation in the training time across different datasets is due to the varying size of the training data and the complexity of the underlying networks. We argue that the training time in the order of hours is acceptable since it is a one-time process that happens in the offline phase.

5.4 Utility Evaluation

We evaluate the utility of OLYMPUS by comparing the classification accuracy of the target app on the unperturbed data and the obfuscated data. The results are summarized in Figure 6a. In the case of the motion sensor data, the classification accuracies on unperturbed data and the obfuscated data are comparable. Thus, the functionality of the target app is preserved even after obfuscation. For image data, we see a slight drop in the accuracy. The maximum drop occurs in the case of the CIFAR10 dataset which is about 17%. Note that in CIFAR10, we artificially added faces on to the images that may cover parts of the object. Due to this occlusion, it is hard to classify the objects correctly. This is also evident from the low classification accuracy on the unperturbed images of CIFAR10. In summary, OLYMPUS preserves the functionality of the target app in the case of motion data, while achieving comparable accuracy in the case of image data.

5.5 Privacy Evaluation

Traditional methods of protecting secrets (especially visual secrets) have focused on protecting secrets from



(a) Results on KTH (left), StateFarm (center) and CIFAR10 (right) datasets **(b)** Obfuscation time

Fig. 7. Comparison with existing approaches. OLYMPUS achieves higher privacy and utility compared to the existing approaches with a minor overhead of time to obfuscate data. In Figure 7a the top-right quadrant represents high utility and high privacy.

Dataset	Unperturbed	Perturbed				
	(A1)	A1	A2	LR	RF	SVM
KTH	0.83	0.01	0.005	0.07	0.06	0.07
StateFarm	0.9	0.05	0.01	0.1	0.1	0.12
CIFAR10	0.5	0.04	0.03	0.07	0.03	0.06
HAR	0.7	0.008	0.008	0.09	0.09	0.02
OPPORTUNITY	0.75	0.11	0.13	0.16	0.43	0.17

Table 3. Accuracy of attackers on obfuscated data measured in terms of the improvement over random guessing. The standard error is within ($\pm$0.09) across all the reported results.

human adversaries. In our case, the obfuscated data is human imperceptible and thus is as good as random noise to any human adversary as evident from the examples given in Table 2. However, that does not mean it is secure against machines. Recently, researchers have shown that DNN can be trained as an adversary to recover hidden information in the obfuscated images [34]. Thus, we employ similar attacks by training five ML models to evaluate privacy of our mechanism.

- The first attacker (A1) is a DNN similar to the *Attacker* network that is used in training OLYMPUS.
- The second attacker (A2) is also a DNN similar to A1, but with an additional layer. Using A2 we attempt to simulate more complex (and possibly stronger) adversary than the one used in training OLYMPUS.
- The remaining three attackers are logistic regression (LR), random forest (RF), and support vector machine (SVM).

All attackers are trained on obfuscated data with the corresponding privacy labels, to classify a given obfuscated image into the correct class. Both A1 and A2 are trained using the same parameters (e.g. epochs, optimizer, etc.) that we used to train the *Obfuscator*. For the remaining three attackers, we set the regularizer parameter of LR to 1, the number of estimators of RF to 10, and used RBF kernel with c=1 for SVM. In the case of image datasets, we train these three attackers on the HOG (Histogram of Oriented Gradients) feature representation computed as described in [17].

Table 3 summarizes the results across various attackers and datasets. On unperturbed data, A1 achieves significant improvement over random guessing. In fact, A1 is able to infer almost all the private information (100% accuracy in classifying private attributes), across all the datasets. Compared to this, OLYMPUS offers a significant improvement in protecting private information against all the attackers across all the datasets. Both A1 and A2 perform poorly (<5% improvement in accuracy over random guessing) in inferring private information from the obfuscated data except in the case of the OPPORTUNITY dataset. Thus, the obfuscation mechanism successfully protects private information against such adversaries, namely DNNs.

On the other hand, the traditional attackers (LR, RF and SVM) perform better than DNN attackers. This is because the underlying attacker model of OLYMPUS is DNN. Moreover, in case of images, the inputs to these attackers are sophisticated HOG features that are specially design to capture various patterns pertaining to human detection. Even though we do not explicitly train against traditional attackers, OLYMPUS protects against those adversaries to some extent. For all the attackers, the accuracy of inferring private information is <17%, except in the case of RF on OPPORTUNITY dataset. This is because the OPPORTUNITY dataset contains many different sensors which makes it harder to obfuscate. In the future, we plan to investigate this further and aim to improve our mechanism for multimodal data.

5.6 Privacy-Utility Tradeoff

We use image datasets to analyze how OLYMPUS balances privacy-utility tradeoff and compare our results with the following existing approaches.

- **Blur:** Blurring is a popular mechanism to hide private information in images and videos [9]. It is a convolution operation that smoothen the image by applying a Gaussian kernel. The size of the kernel determines the amount of blurring. Hence, increasing the kernel size gives more privacy.
- **Mosaic:** Mosaicing is also an averaging operation that applies a grid over an image and replace the values of all the pixels overlapping a cell with the average value of all the pixels that fall in that cell. Increasing the cell size results in averaging over a larger region and thus provides more privacy.
- ***AdvRep*:** *AdvRep* refers to a recent work on protecting visual secrets using GAN [41]. The main idea is to hide the private object by minimizing the reconstruction error and privacy loss given by an attacker network. It is not trained towards preserving the utility of any particular app. Like OLYMPUS, *AdvRep* also has a parameter λ that controls privacy-utility tradeoff. Similar technique have also been used to hide sensitive text in images [18].

We evaluate above mentioned methods as well as OLYMPUS on three image datasets with the goals of protecting respective private attributes and preserving the functionality of the corresponding target app, i.e., the classifier. We obfuscate images using each method and then evaluate the obfuscation in terms of privacy and utility metrics defined previously. For the utility measure, we use the classification accuracy of the target app, i.e., the utility score. We measure privacy using the attacker A1, and report the privacy score as (1 - attacker's score) for ease of representation. Thus, higher values mean higher privacy which makes it easy to compare against the utility score.

To generate different degrees of obfuscation, we vary the kernel (cell) size from 3 to 50 in the case of Blur (Mosaic) method, and vary λ from 0 to 1 in the case of *AdvRep* and OLYMPUS. The resulting privacy-utility tradeoff graph is shown in Figure 7, where * indicates the privacy and utility of the unperturbed data. OLYMPUS outperforms other methods in terms of providing the best privacy-utility balance across all the datasets. In particular, the Blur and Mosaic methods provide either high privacy or high utility, but not both. The *AdvRep* mechanism provides good tradeoff in the case of CIFAR10 but fails in the other two datasets. This is because it learns to reconstruct the given image while removing the private information from it. This strategy works well when the private information is separate from the useful information such as faces vs. objects in the case of CIFAR10. But, it does not perform well when the private and useful information are blended as in the case of other two datasets (identity vs. activity).

An interesting observation is that OLYMPUS strives to achieve high privacy even when $\lambda = 1$, i.e., privacy loss removed from the optimization (Equation 3). For instance, in case of KTH dataset, OLYMPUS provides high privacy irrespective of the value of λ. This is because the features learned by the target app for activity recognition are not very useful for person recognition. Since OLYMPUS only attempts to learn the features used by the target app, the output does not contain enough information to identify people.

We also investigated the effect of correlation on the privacy-utility tradeoff achieved by OLYMPUS. The results of this analysis is available in Appendix B.

5.7 Obfuscation Time

We showed that OLYMPUS outperforms simple obfuscation methods like blurring and mosaicing. However, the gain in accuracy comes at the cost of the overhead of learning the mechanism as well as applying the obfuscation at run time. As mentioned before, the overhead of learning is reasonable as it is a one time operation. Here, we measure the overhead of applying the obfuscation mechanism at run time.

Figure 7b shows the time (ms) it takes to obfuscate an image using above mentioned methods. For Blur (Mosaic), we fix the kernel (cell) size to 25, and fix λ to 0.5 for *AdvRep* and OLYMPUS. Mosaicing is the fastest among all since it only involves a single averaging operation per cell. Blurring is the second fastest method taking slightly more time due to the application of Gaussian kernel. Both *AdvRep* and OLYMPUS takes similar amount of time across all the datasets. Overall, the obfuscation time is proportional to the size of the image across all the methods. OLYMPUS takes about 0.15 ms to obfuscate an image of size 80 x 60 (KTH dataset) which is sufficiently good for real-time processing.

5.8 Scaling to Multiple Applications

To understand how well OLYMPUS scales with multiple apps, we perform the following experiment. For each image dataset, we train a classifier per utility class in one-vs-rest fashion. Each of these classifiers is considered as a target app that is interested in the corresponding utility class. Thus, we have in total 6, 10 and 10

Dataset	Utility (LR)		Privacy (DNN)	
	Unperturbed	Perturbed	Unperturbed	Perturbed
KTH	0.51	0.49	0.20	0.08
StateFarm	0.43	0.74	0.9	0.05
CIFAR10	0.46	0.45	0.17	0.06
HAR	0.93	0.95	0.5	0.02
OPPORTUNITY	0.75	0.86	0.75	0.18

Table 4. Evaluating OLYMPUS using LR as an app classifier. The standard error is within (± 0.03) across all the reported results.

classifiers for KTH, StateFarm and CIFAR10 datasets, respectively. For each dataset, we train OLYMPUS with the appropriate set of classifiers as target apps. The *App* module queries each classifier to compute the respective utility loss and averages it over all the classifiers.

From Figure 6b, we can see that the classification accuracy on obfuscated data is comparable to the accuracy on unperturbed data across all image datasets. When comparing these results with the results from Figure 6a (single target app), we see that the accuracy increases significantly in the case of perturbed as well as unperturbed images. This is not surprising, since each classifier is responsible to classify only a single utility class and hence it is a much simpler task.

To evaluate privacy, we measure the accuracy of attacker (A1) on the obfuscated data learned by OLYMPUS when trained with multiple apps and compare it with the results we got in the single app setting. We found a moderate increase in the attacker's accuracy in the multiple apps case. The attacker's accuracy increased from 0.01, 0.05 and 0.04 to 0.04, 0.06 and 0.05, for KTH, StateFarm and CIFAR10 datasets, respectively.

Note that having multiple target apps only increases the overhead linearly in terms of querying the target apps to compute utility loss. It does not affect the online-phase since we learn a single obfuscation mechanism that works with all the target apps.

5.9 App Classifiers

So far, we have evaluated OLYMPUS only using a DNN as an app classifier. To see how well OLYMPUS performs on different kinds of app classifiers, we change the app classifier to a logistic regression (LR) model. Unlike DNN, LR takes features as inputs instead of raw sensor data. Thus, for image datasets, we use HOG features as inputs to the app classifier. In other words, OLYMPUS learns to obfuscate features instead of raw images. Since motion sensor datasets already comprise of features, we directly use them as inputs to OLYMPUS.

We evaluated OLYMPUS on all the benchmark datasets with LR as the app classifier. As shown in Table 4, the utility on the unperturbed data is low compared to DNN classifiers across all datasets. This is not surprising given that DNN is a more powerful model than LR. However, what is surprising is that on some datasets the utility improves significantly on the obfuscated data. We believe this is due to the *Obfuscator* learning better features in an attempt to minimize utility loss. Since we are using DNN as an adversary, the privacy achieved by OLYMPUS is comparable to previous results of privacy evaluation.

6 Related Work

In this section, we summarize existing approaches of protecting private information in the sensor data. Many prior methods of protecting visual secrets primarily rely on computer vision algorithms to classify images or objects within the images as sensitive (non-sensitive) and hide (reveal) their presence [11, 23, 38, 42, 47]. Unlike OLYMPUS, these methods do not provide an efficient way to balance privacy-utility tradeoff and are prone to leak private information against correlation attacks [34, 35].

Erdogdu [19] proposed privacy preserving mapping for time-series data using information theoretic approach to optimize for statistical privacy-utility tradeoff. Although this approach provides stronger guarantees, in practice, solving such an optimization problem is hard in many scenarios. Shokri *et al.* [45] proposed an optimal user-centric data obfuscation mechanism based on a Stackelberg game. The obfuscation mechanism is a solution of a carefully constructed linear program that minimizes utility loss under specified privacy constraints. This approach works well on certain types of data such as location trajectories that are easy to discretize. However, it is not clear how we can use such mechanism on continuous data such as images. In [36], the authors proposed an approach of learning adversarial perturbations based on a *user-recognizer game*. Although the idea of adversarial game is common to our work, the user-recognizer game model restricts players to play from a fixed set of strategies and hence is limited. On the contrary, OLYMPUS does not fix the strategy of obfuscation/attack, allowing the obfuscator/attacker to automatically learn the optimal strategy based on the specified privacy-utility requirements.

SenseGen [12] uses GAN to generate synthetic sensor data to protect user's privacy. However, it does not provide any privacy guarantees on the generated synthetic data. The approach proposed in [22] is not appli-

cable in our setting as it uses generative models that are tailored to the statistics of the datasets, and applying it to real-life signals (such as images) is an open problem. AttriGuard [25] uses adversarial perturbations to defend against an attacker that attempts to infer private attributes from the data. By defeating a particular attacker, AttriGuard protects against other similar attackers based on the principle of *transferability*. In our work, we take a different approach of adversarial learning to ensure that the private data is protected against all ML attackers belonging to a specific class.

Replacement AutoEncoder (RAE) [33] learns a transformation that replaces discriminative features that correspond to sensitive inferences by features that are more observed in the non-sensitive inferences. This approach only works when the sensitive and non-sensitive data is clearly separated. On the contrary, OLYMPUS also handles cases when there is a high overlap among sensitive and non-sensitive data. Moreover, RAE does not protect against an adversary who has the knowledge of the original gray-listed data, i.e., non-sensitive inferences. On the other hand, OLYMPUS protects against an adversary who has a complete access to the training data as well as the obfuscation mechanism.

Our work is closely related to the recently proposed obfuscation techniques that formulate the obfuscation problem as a minimax optimization [15, 18, 32, 37, 39, 41]. Malekzadeh *et al.* [32] introduces Guardian-Estimator-Neutralizer (GEN) framework. The Guardian learns to obfuscate the sensor data and the Estimator guides the Guardian by inferring sensitive and non-sensitive information from the obfuscated data. Unlike the *Attacker* module in OLYMPUS, the Estimator is pretrained and fixed during the optimization process. Thus, the learned obfuscation defends against a specific attacker, namely the pretrained Estimator. On the other hand, the *Attacker* in OLYMPUS continuously evolve with the *Obfuscator* resulting in an obfuscation mechanism that defends against a class of attackers.

PPRL-VGAN [15] learns a privacy preserving transformation of faces that hides the identity of a person while preserving the facial expression. Given a face image of a person it synthesizes an output face with a randomly chosen identity from a fixed dataset while preserving the facial expression of the input image. SGAP [37] uses Siamese networks to identify discriminative features related to identity and perturbs them using adversarial networks. A similar idea is proposed in [39] where Siamese networks are used to learn embeddings that are non-discriminatory for sensitive information, making it harder for the adversary to learn sensitive information from the embeddings. Adversarial networks are also used to learn an obfuscation mechanism to hide text [18] and QR-code [41] in images.

All of these approaches focus on learning a transformation that preserves some property of the data without considering any target apps. On the contrary, OLYMPUS learns an obfuscation mechanism that seamlessly works with the existing apps without modifying them. We also demonstrated the feasibility of OLYMPUS by developing a prototype implementation that runs on a smartphone, and evaluated against a real world app.

7 Conclusions

We proposed a privacy framework OLYMPUS to learn a *utility aware obfuscation* that protects private user information in image and motion sensor data. We showed that such a mechanism can be constructed given the training data, and the obfuscated data works well with the target third-party apps without modifying the apps.

We implemented OLYMPUS by instrumenting Android OS and evaluated the obfuscation mechanism on a handwriting recognition app. We showed that OLYMPUS successfully protects the identity of users without compromising the digit classification accuracy of the app. We also evaluated OLYMPUS on three image datasets and two sensor datasets containing readings from various motion sensors. For each dataset, we showed that OLYMPUS significantly reduced the risk of disclosing private user information. At the same time, it preserved the useful information that enabled the target app to function smoothly even on the obfuscated data. We verified the privacy guarantees using a number of ML adversaries that are trained to defeat the obfuscation. We also compared our approach with existing approaches of protecting visual secrets and demonstrated that OLYMPUS provides better control over privacy-utility tradeoff.

Acknowledgement

This work was supported by the National Science Foundation under grants 1253327, and by DARPA and SPAWAR under contract N66001-15-C-4067.

References

[1] AT&T Database of Faces. http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.

[2] Deep learning for mobile:dl4mobile. https://play.google.com/store/apps/details?id=com.nalsil.tensorflowsimapp&hl=en.

[3] Google Cloud AI. https://cloud.google.com/products/machine-learning/.

[4] Human activity recognition using cnn. https://github.com/aqibsaeed/Human-Activity-Recognition-using-CNN.

[5] Speech recognition tensorflow machine learning. https://play.google.com/store/apps/details?id=machinelearning.tensorflow.speech&hl=en.

[6] State Farm Distracted Driver Detection. https://www.kaggle.com/c/state-farm-distracted-driver-detection/data.

[7] TensorFlow Inference API. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/android.

[8] TensorFlow Models. https://github.com/tensorflow/models.

[9] Youtube official blog. Blur moving objects in your video with the new custom blurring tool on youtube. https://youtube-creators.googleblog.com/2016/02/blur-moving-objects-in-your-video-with.html, 2016.

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. OSDI, 2016.

[11] P. Aditya, R. Sen, P. Druschel, S. Joon Oh, R. Benenson, M. Fritz, B. Schiele, B. Bhattacharjee, and T. T. Wu. I-pic: A platform for privacy-compliant image capture. MobiSys, 2016.

[12] M. Alzantot, S. Chakraborty, and M. B. Srivastava. Sensegen: A deep learning architecture for synthetic sensor data generation. *BICA*, 2017.

[13] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. ESANN, 2013.

[14] R. Chavarriaga, H. Sagha, A. Calatroni, S. T. Digumarti, G. Tröster, J. D. R. Millán, and D. Roggen. The opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recogn. Lett.*, 2013.

[15] J. Chen, J. Konrad, and P. Ishwar. Vgan-based image representation learning for privacy-preserving facial expression recognition. CVPR Workshops, 2018.

[16] F. Chollet. keras. https://github.com/fchollet/keras, 2015.

[17] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. CVPR, 2005.

[18] H. Edwards and A. J. Storkey. Censoring representations with an adversary. *ICLR*, 2016.

[19] M. Erdogdu, N. Fawaz, and A. Montanari. Privacy-utility trade-off for time-series with application to smart-meter data. AAAI Workshops, 2015.

[20] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. NIPS, 2014.

[21] J. Hamm. Minimax filter: Learning to preserve privacy from inference attacks. *JMLR*, 2017.

[22] C. Huang, P. Kairouz, X. Chen, L. Sankar, and R. Rajagopal. Context-aware generative adversarial privacy. *Entropy*, 2017.

[23] S. Jana, D. Molnar, A. Moshchuk, A. Dunn, B. Livshits, H. J. Wang, and E. Ofek. Enabling Fine-Grained Permissions for Augmented Reality Applications With Recognizers. USENIX Security, 2013.

[24] S. Jana, A. Narayanan, and V. Shmatikov. A Scanner Darkly: Protecting User Privacy from Perceptual Applications. S & P, 2013.

[25] J. Jia and N. Z. Gong. Attriguard: A practical defense against attribute inference attacks via adversarial machine learning. USENIX Security, 2018.

[26] J. Jung and M. Philipose. Courteous glass. UbiComp '14 Adjunct, 2014.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[28] M. Korayem, R. Templeman, D. Chen, D. J. Crandall, and A. Kapadia. Screenavoider: Protecting computer screens from ubiquitous cameras. *CoRR*, 2014.

[29] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, 2009.

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[31] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. ICCV, 2015.

[32] M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi. Protecting sensory data against sensitive inferences. EuroSys Workshop, 2018.

[33] M. Malekzadeh, R. G. Clegg, and H. Haddadi. Replacement autoencoder: A privacy-preserving algorithm for sensory data analysis. IoTDI, 2018.

[34] R. McPherson, R. Shokri, and V. Shmatikov. Defeating image obfuscation with deep learning. *CoRR*, 2016.

[35] S. J. Oh, R. Benenson, M. Fritz, and B. Schiele. Faceless person recognition; privacy implications in social media. *ECCV*, 2016.

[36] S. J. Oh, M. Fritz, and B. Schiele. Adversarial image perturbation for privacy protection - A game theory perspective. *ICCV*, 2017.

[37] W. Oleszkiewicz, T. Wlodarczyk, K. J. Piczak, T. Trzcinski, P. Kairouz, and R. Rajagopal. Siamese generative adversarial privatizer for biometric data. CVPR Workshops, 2018.

[38] T. Orekondy, M. Fritz, and B. Schiele. Connecting pixels to privacy and utility: Automatic redaction of private information in images. CVPR, 2018.

[39] S. A. Ossia, A. S. Shamsabadi, A. Taheri, H. R. Rabiee, N. D. Lane, and H. Haddadi. A hybrid deep learning architecture for privacy-preserving mobile analytics. *CoRR*, 2017.

[40] K. Plarre, A. Raij, S. M. Hossain, A. A. Ali, M. Nakajima, M. Al'absi, E. Ertin, T. Kamarck, S. Kumar, M. Scott, D. Siewiorek, A. Smailagic, and L. E. Wittmers. Continuous inference of psychological stress from sensory measurements collected in the natural environment. IPSN, 2011.

[41] N. Raval, A. Machanavajjhala, and L. P. Cox. Protecting visual secrets using adversarial nets. CVPR Workshops, 2017.

[42] N. Raval, A. Srivastava, A. Razeen, K. Lebeck, A. Machanavajjhala, and L. P. Cox. What you mark is what apps see. MobiSys, 2016.
[43] rovo89. Xposed framework. https://forum.xda-developers.com/showthread.php?t=3034811.
[44] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: A local svm approach. ICPR, 2004.
[45] R. Shokri. Privacy Games: Optimal User-Centric Data Obfuscation. *PETS*, 2015.
[46] S. Sonoda and N. Murata. Neural network with unbounded activations is universal approximator. *ACHA*, 2017.
[47] R. Templeman, M. Korayem, D. Crandall, and A. Kapadia. PlaceAvoider: Steering first-person cameras away from sensitive spaces. NDSS, 2014.
[48] A. Varshney and S. Puri. A survey on human personality identification on the basis of handwriting using ann. ICISC, 2017.
[49] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. ICML, 2008.
[50] C.-W. You, N. D. Lane, F. Chen, R. Wang, Z. Chen, T. J. Bao, M. Montes-de Oca, Y. Cheng, M. Lin, L. Torresani, and A. T. Campbell. Carsafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones. MobiSys, 2013.
[51] A. Zhan, M. Chang, Y. Chen, and A. Terzis. Accurate caloric expenditure of bicyclists using cellphones. SenSys, 2012.
[52] X. Y. Zhang, G. S. Xie, C. L. Liu, and Y. Bengio. End-to-end online writer identification with recurrent neural network. *THMS*, 2017.

A Datasets

Below, we describe the datasets we used to evaluate OLYMPUS.

KTH: KTH [44] is a video database for action recognition. It contains six different actions performed by 25 subjects under four different scenarios. The actions are walking, jogging, running, boxing, hand-waving and hand-clapping. Our goal is to protect the identity of the subjects performing the actions while allowing the target app to recognize the actions correctly. The videos were recorded at 25fps with 160x120 resolution. We uniformly sampled 50 frames from each video of six randomly selected subjects for the evaluation. As a preprocessing step, we scaled all the extracted frames to 80x60 and converted them to grayscale.

StateFarm: StateFarm [6] is an image dataset used in a Kaggle competition for detecting distracted drivers. It has images of drivers performing various activities in the car. In total, there are 10 different activities performed by the driver – safe driving, texting (right), texting (left), talking on phone (right), talking on phone (left), operating radio, drinking, reaching behind, hair/makeup and talking to passenger. Motivated by our *driver safety* example, our goal is to protect the identity of drivers while allowing the target app to infer driver activities. For our experiments, we use images of 10 randomly selected drivers. Each image was of size 224 x 224 which was scaled down to 56x56 and converted to grayscale as a preprocessing step.

CIFAR10: CIFAR10 [29] is a popular object detection dataset. It consists of 32×32 color images from 10 categories, each having 6000 images. For the private attribute, we added faces from the ATT face dataset [1]. A random face from the ATT face dataset was added to about half of the randomly selected images from the dataset. Each face is added at a random location in the image such that the entire face is visible. The original face images were of size 92x112 pixels which we scaled down to 10x10 before adding. Our goal is to obfuscate the image such that the target app can classify obfuscated images into one of the 10 object categories correctly, while the adversary cannot infer whether there exist a face in the image. We preprocess all the images by converting them to grayscale.

HAR: HAR [13] is a human activity recognition dataset containing readings from the accelerometer and gyroscope embedded in a smartphone. The readings involved 30 users performing six activities – walking, walking-upstairs, walking-downstairs, sitting, standing and lying. A 561 dimensional feature vector with time and frequency domain variables is computed using a sliding window method. Each feature vector has an associated subject id and the activity label performed by the subject at that time. Our goal is to obfuscate the feature vector to protect the identity of the subject while allowing the target app to infer activities.

OPPORTUNITY: OPPORTUNITY [14] is also a human activity recognition dataset. It contains 242 attributes from wearable, object, and ambient sensors. The sensor readings were recorded using four subjects performing various activities. In our experiments, we use data pertaining to following locomotion activities – stand, walk, sit, lie, and null, where null represents a transition between activities or other activities. The dataset contains six independent runs of daily activities per user. We ran our experiment on randomly sampled two runs across all users. As in the case of HAR, our goal is to protect the identity of individuals, while allowing the target app to infer the locomotion activities from the obfuscated sensor data.

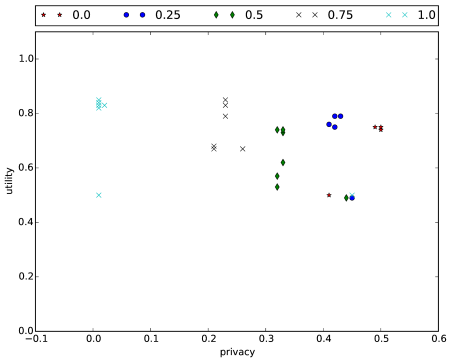


Fig. 8. OLYMPUS gracefully handles correlation among sensitive and useful properties.

B Effect of Correlation

Handling correlation between private and app-essential (utility) information is a major challenge in designing any obfuscation mechanism. To understand the effect of such correlation, we evaluate OLYMPUS on a synthetically generated data where we carefully control the correlation between private and utility attributes.

B.1 Synthetic Data Generation

We generate a two dimensional synthetic dataset sampled from a normal distribution with a specific mean and a covariance matrix. We fix the mean to (0,0) and vary the covariance matrix to control the degree of correlation among two dimensions. One can think of these two dimensions as features f_s and f_u representing private and utility attributes, respectively. Both these attributes are binary (positive and negative) and their values with respect to a data point is computed based on the values of their respective features as follows. For each data point, we say it belongs to the positive privacy (utility) class, if the value of f_s (f_u) is positive, and belongs to the negative class otherwise. We vary the value of the correlation factor among two features from 0 (no correlation) to 1 (highest correlation). For each correlation factor, we generate 10000 samples with their privacy and utility labels to evaluate OLYMPUS.

B.2 Network Architecture

Both the *App* and the *Attacker* networks comprise of a fully connected (FC) layer with 64 nodes followed by another FC layer with 32 nodes. The *Obfuscator* network consists of three FC layers with 64, 32 and 64 nodes, respectively. We use ReLU activation and dropout (p=0.25) at each layer in all the networks.

B.3 Results

We split the data into train, validation and test set based on 80%, 10% and 10% splits, respectively. We train OLYMPUS using the Adam optimizer with the learning rate of 0.0001 using 100 epochs and report the results averaged over 10 runs. The privacy-utility tradeoff achieved by OLYMPUS with different degree of correlation is given in Figure 8. We can see that when there is little or no correlation among private and utility attributes, OLYMPUS achieves high privacy and high utility. As the correlation increases, OLYMPUS degrades one at the expense of the other depending on the specified tradeoff parameter λ. In summary, the results show that OLYMPUS gracefully handles correlation among private and app-essential information in the data, and allows users to control the privacy-utility tradeoff.

C Neural Network Architectures

In this section, we provide details of the underlying networks of OLYMPUS.

C.1 OLYMPUS for Images

For the image datasets, we use CNN as shown in Figure 2. As a preprocessing step, we normalize images to [0,1] which is a standard practice in computer vision. Below, we provide complete specification of the networks used in evaluating OLYMPUS on the image datasets.

C.1.1 KTH

App Net

1. Input: 60 x 80 x 1 obfuscated image
2. Conv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
3. MaxPool (size=(2,2))
4. Conv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
5. MaxPool (size=(2,2))
6. Dense (n=32,activation=ReLU)
7. Dense (n=6,activation=softmax)

Attacker Net

1. Input: 60 x 80 x 1 obfuscated image
2. Conv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
3. MaxPool (size=(2,2))
4. Conv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
5. MaxPool (size=(2,2))
6. Dense (n=32,activation=ReLU)
7. Dense (n=6,activation=softmax)

Obfuscator Net

1. Input: 60 x 80 x 1 unperturbed image
2. Conv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
3. MaxPool (size=(2,2))
4. Conv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
5. MaxPool (size=(2,2))
6. Dense (n=8,activation=ReLU)
7. Dense (n=300,activation=ReLU)
8. Reshape (size=(15,20))
9. Deconv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
10. UpSample2D (size=(2,2))
11. Deconv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
12. UpSample2D (size=(2,2))
13. Deconv2D (filters=1,size=(3,3),stride=1,activation=sigmoid)

C.1.2 StateFarm

App Net

1. Input: 56 x 56 x 1 obfuscated image
2. Conv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
3. MaxPool (size=(2,2))
4. Conv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
5. MaxPool (size=(2,2))
6. Dense (n=32,activation=ReLU)
7. Dense (n=10,activation=softmax)

Attacker Net

1. Input: 56 x 56 x 1 obfuscated image
2. Conv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
3. MaxPool (size=(2,2))
4. Conv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
5. MaxPool (size=(2,2))
6. Dense (n=32,activation=ReLU)
7. Dense (n=10,activation=softmax)

Obfuscator Net

1. Input: 56 x 56 x 1 unperturbed image
2. Conv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
3. MaxPool (size=(2,2))
4. Conv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
5. MaxPool (size=(2,2))
6. Dense (n=8,activation=ReLU)
7. Dense (n=196,activation=ReLU)
8. Reshape (size=(14,14))
9. Deconv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
10. UpSample2D (size=(2,2))
11. Deconv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
12. UpSample2D (size=(2,2))
13. Deconv2D (filters=1,size=(3,3),stride=1,activation=sigmoid)

C.1.3 CIFAR10

App Net

1. Input: 32 x 32 x 1 obfuscated image
2. Conv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
3. Conv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
4. MaxPool (size=(2,2))
5. Dropout (p=0.25)
6. Conv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
7. Conv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
8. MaxPool (size=(2,2))
9. Dropout (p=0.25)
10. Dense (n=512,activation=ReLU)
11. Dropout (p=0.5)
12. Dense (n=10,activation=softmax)

Attacker Net

1. Input: 32 x 32 x 1 obfuscated image
2. Conv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
3. MaxPool (size=(2,2))
4. Conv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
5. MaxPool (size=(2,2))
6. Dense (n=32,activation=ReLU)
7. Dense (n=2,activation=softmax)

Obfuscator Net

1. Input: 32 x 32 x 1 unperturbed image
2. Conv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
3. MaxPool (size=(2,2))
4. Conv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
5. MaxPool (size=(2,2))
6. Dense (n=8,activation=ReLU)
7. Dense (n=64,activation=ReLU)
8. Reshape (size=(8,8))
9. Deconv2D (filters=64,size=(3,3),stride=1,activation=ReLU)
10. UpSample2D (size=(2,2))
11. Deconv2D (filters=32,size=(3,3),stride=1,activation=ReLU)
12. UpSample2D (size=(2,2))
13. Deconv2D (filters=1,size=(3,3),stride=1,activation=sigmoid)

C.2 OLYMPUS for Motion Sensors

For the motion sensor data, we use DNN with fully connected layers as shown in Figure 3. We normalize the sensor data to [-1,1] as a preprocessing step. Below, we provide complete specification of the networks used in evaluating OLYMPUS on the motion sensor datasets.

C.2.1 HAR

App Net

1. Input: 561 x 1 obfuscated feature vector
2. Dense (n=128,activation=ReLU)
3. Dense (n=6,activation=softmax)

Attacker Net

1. Input: 561 x 1 obfuscated feature vector
2. Dense (n=128,activation=ReLU)
3. Dense (n=30,activation=softmax)

Obfuscator Net

1. Input: 561 x 1 unperturbed feature vector
2. Dense (n=64,activation=ReLU)
3. Dense (n=8,activation=ReLU)
4. Dense (n=64,activation=ReLU)
5. Dense (n=561,activation=tanh)

C.2.2 OPPORTUNITY

App Net

1. Input: 242 x 1 obfuscated feature vector
2. Dense (n=128,activation=ReLU)
3. Dense (n=5,activation=softmax)

Attacker Net

1. Input: 242 x 1 obfuscated feature vector
2. Dense (n=128,activation=ReLU)
3. Dense (n=4,activation=softmax)

Obfuscator Net

1. Input: 242 x 1 unperturbed feature vector
2. Dense (n=64,activation=ReLU)
3. Dense (n=8,activation=ReLU)
4. Dense (n=64,activation=ReLU)
5. Dense (n=242,activation=tanh)