An Attack on InstaHide: Is Private Learning Possible with Instance Encoding?

Nicholas Carlini  
ncarlini@google.com

Samuel Deng  
sd3013@columbia.edu

Sanjam Garg  
sanjam@berkeley.edu

Somesh Jha  
jha@cs.wisc.edu

Saeed Mahloujifar  
sfar@princeton.edu

Mohammad Mahmoody  
mohammad@virginia.edu

Shuang Song  
shuangsong@google.com

Abhradeep Thakurta  
athakurta@google.com

Florian Tramèr  
tramer@cs.stanford.edu

Figure 1: Our solution to the InstaHide Challenge. Given 5,000 InstaHide encoded images released by the authors, under the strongest settings of InstaHide, we recover a visually recognizable version of the original (private) images in under an hour on a single machine.

Abstract

A learning algorithm is private if the produced model does not reveal (too much) about its training set. InstaHide [Huang, Song, Li, Arora, ICML’20] is a recent proposal that claims to preserve privacy by an encoding mechanism that modifies the inputs before being processed by the normal learner.

We present a reconstruction attack on InstaHide that is able to use the encoded images to recover visually recognizable versions of the original images. Our attack is effective and efficient, and empirically breaks InstaHide on CIFAR-10, CIFAR-100, and the recently released InstaHide Challenge.

We further formalize various privacy notions of learning through instance encoding and investigate the possibility of achieving these notions. We prove barriers against achieving (indistinguishability based notions of) privacy through any learning protocol that uses instance encoding.

1 Introduction

Neural networks are increasingly trained on sensitive user data, for example building classifiers to diagnose from medical images [HPO+18, WYB+10] or help users compose emails or text messages by training on actual user data [DLB+19]. When these classifiers are made publicly available, they must not reveal the sensitive details of their training data, but still must be accurate and solve the underlying learning task. It is thus important that these models preserve the privacy of their sensitive training data.

Most well-known techniques to train with guaranteed differential privacy [DR+14] come at the cost of accuracy (or equivalently, the same accuracy level requires more data [BSTT3][BBKNT]). For example, training deep neural networks with differentially private stochastic gradient descent [SCST3][ACG+16] can more than double the error rate of classifiers. This raises the question: Are there other ways to perform private learning without sacrificing accuracy?
We consider one leading alternative that works as follows. Given an arbitrary dataset, convert it to an encoded (privacy-preserving) version, and then train a non-private neural network on this encoded training dataset. Because the training data has been privately encoded, the model gets privacy “for free.” We formalize this setup, and place fundamental limits on how well such an approach can work in theory.

InstaHide [HSLA20] is the most well known concrete instantiation of this encoding framework. InstaHide claims to offer strong (empirical) privacy, by showing that it is robust to a variety of attacks considered by the authors.

We introduce a reconstruction attack on InstaHide. Given an encoded dataset generated with the strongest version of InstaHide, we are able to perform a complete reconstruction attack and recover visually similar copies of the original private data. In contrast, models trained with differential privacy provably prevent even distinguishing attacks [Vad17] (i.e., determining whether a given data point was used to train the model or not).

Figure 1 shows our attack on the InstaHide Challenge, a collection of 5,000 encoded images released by the authors that are the “encryption” of 100 private images. Presented are ten of the reconstructed images contained in the private dataset; the full list of 100 reconstructed images are given in Appendix A.

InstaHide does not have any provable privacy guarantee; as a result, our attacks do not violate any theorems or theoretical results in the paper. One of the main reasons why training techniques that can prove they are differentially private [DMNS06, DR14] are so appealing is that it offers provable semantic privacy guarantees [KS08], albeit at the cost of loss in accuracy in some settings. Nonetheless, design of security and privacy preserving techniques without precise security/privacy definitions is often error prone [GKS08] and should be avoided. Historically, privacy techniques like k-anonymity, ℓ-diversity, m-invariance, and t-closeness were attacked [GKS08], primarily because they failed at providing any formal semantic guarantees [KS08].

We notified the authors of our reconstruction attack, and in Section 6 give their response to our attack.

2 InstaHide Definition & Notation

Let \((x, y) \in X\) be a sensitive, labeled dataset with training examples \(x_i \in [−1, 1]^d\). Because this dataset is sensitive, it is undesirable to directly train a (non-private) machine learning model on these examples for fear that the model may reveal particular aspects of the underlying training data.

InstaHide proposes a technique to construct an encoded dataset, upon which any (non-private) algorithm can be trained. InstaHide proceeds as follows. First, gather a large public dataset \(p \in P\), e.g., of arbitrary images (e.g., from the internet). Then, generate the encoded dataset \((e, z) \in E\) (representing encoded images \(e\) with encoded labels \(z\)) by assigning

\[ E \leftarrow \{ (XMix(\{x_i, y_j\}, p, \lambda), YMix(y_i, y_j, \lambda) : ((x_i, y_i), (x_j, y_j)) \in X, p \subset P, |p| = k - 2 \}. \]

The core algorithms to InstaHide are thus \(XMix\) and \(YMix\), defined as follows.

\[ XMix(x, p, \lambda) = \sigma \left( \sum_{i=1}^{2} x_i \lambda_i + \sum_{i=3}^{k} p_i \lambda_i \right) \]

with \(\lambda\) chosen uniformly at random such that \(\sum_{i=1}^{d} \lambda_i = 1\); \(\sigma\) chosen uniformly at random from \(\{−1, 1\}^d\), and where \(a \circ b\) denotes element-wise multiplication. The function \(YMix\) is much simpler and given by

\[ YMix(y_i, y_j, \lambda) = y_i \lambda_1 + y_j \lambda_2 \]

with addition here taken component-wise across one-hot labels. The size of the encoded dataset is determined by the encoding multiple \(N = |E|/|X|\), with each instance being encoded \(N\) times. In practice this multiple is equal to the number of epochs of training (e.g., 50 or 100). The authors argue InstaHide is secure for \(k \geq 4\), with the strongest version at \(k = 6\) (the InstaHide Challenge released by the authors uses \(k = 6\), for example).

As claimed by InstaHide, this encoded dataset preserves the privacy of the original training dataset. One can now use arbitrary non-private training techniques to train on this encoded dataset and
Figure 2: **Our attack process.** Given the encoded dataset, we cluster together images generated from the same original source image and then from these sets “decrypt” them to the original sources.

get privacy “for free” without having to resort to differentially private training, which can reduce accuracy.

We make use of some additional notation. Let \( \phi : E \rightarrow (|X| \times |X|) \) represent the mapping from the encoded images to original private images. By \( \phi(e_i) = (j, k) \) we mean that encoded image \( e_i \) is built out of the original images \( x_j \) and \( x_k \). Similarly, let \( \phi^{-1} \) be the inverse so that \( \phi^{-1} : X \rightarrow 2^{|E|} \), for example \( i \in \phi^{-1}(x_j) \) and \( i \in \phi^{-1}(x_k) \). Note that while \( \phi \) maps one \( x \in X \) to exactly two \( e_1, e_2 \in E \), the inverse \( \phi^{-1} \) maps one \( x \in X \) to approximately \( 2^N \) encoded images \( e \in E \).

**Why InstaHide does not give “instance hiding” security.** InstaHide claims to be similar to an encryption scheme that hides almost all the information of the private images by appealing to instance hiding and assuming that \( \sigma \) operates as a “one time pad”. However, as the InstaHide authors observe in their paper, given an arbitrary input \( x \), the multiplication \( x \circ \sigma \) where \( \sigma \) is sampled uniformly at random from \( \sigma \in \{-1, 1\}^d \) is no more secure than simply releasing \( \text{abs}(x) \). An adversary can, trivially, convert \( \text{abs}(x \circ \sigma) = \text{abs}(x) \). If releasing \( \text{abs}(x) \) would not be considered secure, then releasing \( x \circ \sigma \) should not be considered secure. Therefore, the “instance hiding” intuition is not well founded: a trivial operation on the encoded inputs recovers a piecewise linear function of the original data.

3 **Our Attack on InstaHide’s Privacy**

We break InstaHide’s privacy through an attack that consists of three stages:

1. **Remove instance hiding:** Replace the encoded dataset by
   \[
   E \leftarrow \{ \text{abs}(e) : e \in E \}
   \]
   which in effect removes all impact of the sign flipping step of instance hiding.

2. **Cluster encoded dataset:** Given these absolute-value images, we recover the mapping \( \phi \) that determines which original source images were used to generate each encoded image. We achieve this through training a neural network to detect when two encoded images were generated from the same original image. This allows us to build a graph representing pairwise similarity between encoded images, from which we can extract one clique per original image with the vertices in this clique corresponding to the encoded images generated from that original image.

3. **Recover original images:** Then, given the encoded images and the mapping \( \phi \), we recover (an approximation of) the original labeled images \( X \).

   This step happens all at once, solving an under-determined (nonlinear) system of equations via gradient descent. Because the system is under-determined, it is provably impossible to recover the original images pixel-perfect, however this does not prevent reconstructions that have high similarity to the original images both qualitatively and quantitatively.

We release the source code of our attack as a utility that can be used to break the privacy of arbitrary InstaHide encoded images. As we will show, our attack is hyperparameter free (except for the sizes
of the images) and the one configuration we release breaks the privacy of InstaHide on CIFAR-10, CIFAR-100, and the InstaHide challenge.

3.1 Clustering

The purpose of the clustering process is to recover \( \phi \), the function that maps original source images onto encoded images. Because each encoded image has two original images that were used to generate it, our goal is to recover \(|X|\) sets \( S_i \) of encoded images so that each set has size about \(|S_i| \approx 2N\). That is, at the end of this step, even though we will not have a reconstruction of any particular image \( x_i \), we will know which encoded images were generated using \( x_i \).

This procedure follows a five step process.

1. Create a pairwise similarity function \( sim(e_i, e_j) \rightarrow [0, 1] \) so that \( sim \) is high if \( e_i \) and \( e_j \) share at least one source image and low otherwise.
2. Construct the complete weighted similarity graph \( G \) that represents the all-pairs similarity.
3. Identify sets \( \{S_j\}_{j=1}^{\log_2|X|} \) by finding the densely connected cliques.
4. Construct a new bipartite graph that maps the similarity between each encoded \( e_i \) and the nearest set \( S_j \).
5. Assign each encoded image \( e_i \) to two sets \( S_j \), and assign each set \(|N|\) encoded images, minimizing total cost.

3.1.1 Create a similarity function

Our first step of the attack constructs a similarity function \( sim \) that determines if two images \( e_i \) and \( e_j \) were generated using at least one shared original image.

**Inputs:** The public dataset \( P \).

**Outputs:** The function \( sim \), so that \( sim(e_i, e_j) \) is (usually) 1 if \( \phi(e_i) \cap \phi(e_j) \neq \emptyset \) and 0 otherwise.

**Method** We train a neural network to approximate this similarity function \( sim \). We create a large training dataset with examples of pairs of images encoded together and not. This neural network receives the two inputs \( e_i \) and \( e_j \) stacked on the channel dimension (so, concretely, for \( 32 \times 32 \times 3 \) color images the input to the neural network is \( 32 \times 32 \times 6 \)). The neural network outputs a single scalar \( y \in \mathbb{R} \) and we assign a standard sigmoid loss so that \( y > 0 \) when the two images share an original image and \( y < 0 \) otherwise.

We train a single neural network to be used for all attacks in this paper. We use a Wide ResNet-28 trained with Adam with a learning rate of 0.1 and a weight decay factor of \( 5 \cdot 10^{-4} \) for \( 10^6 \) steps. We use a 32x32 downsampling of ImageNet as the public dataset following the process described in the InstaHide paper, and the CIFAR-10, CIFAR-100, and STL-10 training images as the private images. We augment the training process with standard flips and shifts. The final trained model reaches 91% accuracy on a held-out validation set.

3.1.2 Construct the similarity graph

**Inputs:** The encoded images \( E \), and the similarity function \( sim \) from the prior subsection.

**Outputs:** A complete weighted similarity graph \( G \) that has an edge between each encoded image \( e_i \) and \( e_j \) with weight equal to \( sim(e_i, e_j) \).

**Method** This step is trivial. We evaluate the neural network on all \(|E|^2\) pairs of images. For modestly sized encoded datasets this process is efficient, for example on the 5,000 image contest dataset this step finishes in 10 minutes.

---

[1] https://github.com/Hazelsuko07/InstaHide_Challenge

[2] Because each image is sampled randomly, the sets are only each of size approximately \( 2N \).
3.1.3 Identify densely connected cliques

**Inputs:** The complete weighted graph $G$ from the prior subsection.

**Outputs:** A coloring of the vertices into one of $|X|$ non-overlapping subsets $S = \{S^{(i)}\}_{i=1}^{|X|}$ that approximately maximizes

$$\sum_{S \in S} \sum_{e_i \in S, e_j \in S} \text{weight}(e_i, e_j).$$

In an ideal reconstruction, we would have that

$$\bigcap_{e \in S^{(i)}} \phi(e) = 1 \quad \forall i \neq j : \left( \bigcap_{e \in S^{(i)} \cup S^{(j)}} \phi(e) \right) = \emptyset$$

That is, each subset contains encodings that share exactly one source image (the representative of this subset). Moreover, no two subsets have the same representative.

**Method** The purpose of this algorithm is to create $|X|$ clusters, one for each original image in the dataset. In reality, each encoded image is mapped onto 2 different original images; however, for now, we will simply assign each encoded image to just one original image. That allows this step to be a simpler problem of “coloring” this graph with $|X|$ different colors minimizing cost.

We develop a greedy algorithm to approximately solve this problem. The core of our algorithm is a recursive loop that iteratively selects the next best encoded image to add to an existing set using the update rule

$$\text{insert}(S) = S \cup \left\{ \arg \max_{e \in E} \sum_{u \in S^{(i)}} \text{weight}(e, u) \right\}.$$  

That is, we greedily add the closest example that has the highest weight when considering those examples that are already in the set. Then we define

$$\text{create}(S, M) = \text{insert}(\ldots (\text{insert}(S) \ldots)$$

This lets us compute the sets $T^{(i)} = \text{create}(\{e_i\}, M)$ for each encoding $e_i \in E$. To choose the integer $M$ we select a constant $M < N/2$ (we found that setting $M = N/4$ works in practice). At this point, we should expect that there are $|X|$ distinct sets among the collection of sets $\{T^{(i)}\}_{i=1}^{\lvert X \rvert}$.

**Justification.** If each step up until this point was perfect (i.e., if the similarity neural network returned 1 if and only if two encoded images were generated from the same source image) then with probability almost 1 we would expect exactly $|X|$ distinct sets: one for each original image. That is, formally, we can inductively prove that

$$\bigcap_{s \in T^{(i)}} \phi(s) > 0$$

(and with overwhelming probability this intersection contains exactly one element). To see that this is the case, when we start with a set containing a single element $\{e_i\}$ and call $\{e_i, e_j\} \leftarrow \text{insert}(\{e_i\})$ we are guaranteed to have that $e_i$ and $e_j$ share at least one original image $x$ (formally, $|\phi(e_i) \cap \phi(e_j)| > 0$). With probability $\frac{1}{|X|}$, we should expect $|\phi(e_i) \cap \phi(e_j)| = 1$ because each encoded image is constructed by pairing together two original images at random, and so the probability that two encoded images share both original images given that at least one is identical is $\frac{1}{|X|}$. The inductive case is identical.

Importantly, if $\phi(e_i) = (x_a, x_b)$, then both of the original images $x_a$ and $x_b$ have equal probability of also being a part of some other encoding $e_j$. Thus, consider each encoded image $e$ that is generated using the original image $x_b$. The probability that

$$x_b \notin \bigcup_{e \in \phi^{-1}(x_b)} \left( \bigcap_{x \in \phi(e)} \phi(x) \right)$$
is exactly $1/2^N$, as this happens only if each call to \textit{create}(e) creates a set based around the other private image used to generate that encoding \(e\). Thus, with \(N = 100\) as we have in our experiments, we can discount this ever happening. This allows us to conclude that we will have \(|X|\) sets.

Unfortunately the prior steps are not perfect. As a result, it is possible to have \(\epsilon < |T^{(i)} \cap T^{(j)}| < N\) for \(\epsilon\) an integer greater than zero. We can still solve this problem approximately, however. Given the \(|E|\) sets, we are looking to cluster them into \(|X|\) clusters-of-sets where we maximize the similarity of the sets contained in one cluster. To do this, we perform k-means clustering on these sets (with \(k = |X|\)), where the distance between sets is defined as \(d(s, t) = \frac{|s \cap t|}{|s \cup t|}\). We run this to cluster the sets into \(|X|\) different clusters and then choose one representative (arbitrarily) from each cluster to form the sets \(S^{(i)}\).

### 3.1.4 Compute similarity between encodings and the cliques

**Inputs:** The encoded images \(E\), and the \(|X|\) (near-)cliques \(S\).

**Outputs:** A new graph \(G'\) that computes the distance from any encoded image \(e \in E\) to each of the other sets \(S\).

**Method** The simplest strategy is trivial to define. Just compute the average \(\sum_{v \in S} \text{weight}(e, v)\) for each \(S \in S\), and picks whichever is closest.

We can do better, though. This similarity graph was constructed with a single neural network that receives two encoded images and returns 1 if they share an original image. Our problem is now easier. Here, we have \(|S^{(i)}|\) encoded images, all of which (probably) belong to the same original image \(x\), and would like to test if a new encoded image \(e\) also corresponds to the same original image \(x\). We have found experimentally that we reach diminishing returns once we provide the neural network with 4 examples.

Concretely, we train this new similarity neural network to return 1 if an encoded image \(e\) shares the same original image as all four exemplars \(\{e_1, e_2, e_3, e_4\}\). This makes the task easier for the network to solve. By having 4 examples of what the original image looks like, it is easier for the model to learn to predict if a 5th image uses a similar base image. In practice, this new neural network increases the prediction accuracy from 91% to 96% (reducing the error rate by a factor of 2).

To construct the similarity graph \(G'\) we choose four “canonical” representatives of each set \(S\) at random. Then, we compute the distance from each \(e \in E\) to the four canonical representatives from each set, giving us a dense bipartite graph connecting the \(|X|\) sets to the \(|E|\) examples.

### 3.2 Assign each encoded image to an original image

**Inputs:** The new similarity graph \(G'\).

**Outputs:** A mapping \(\phi'\) that maps encoded images to original images. Ideally, we will have that \(\phi' = \phi\).

**Method** It turns out that we can solve the final assignment problem with a single invocation of min cost max flow. We construct a source node with a supply of \(2|X|\), and a sink node with a supply of \(-2|X|\). Then, we connect the source to each set \(S\) with capacity \(|N|\), each set \(S\) to each example \(e_i\) with capacity 1, and each example \(e\) to the sink with capacity 2.

The min cost max flow assignment will therefore assign each example \(e_1\) to exactly two sets \(S\), and assign each set to exactly \(|N|\) distinct examples \(e_i\), exactly satisfying the constraints specified for \(\phi\). This gives us the mapping function \(\phi'\).

The fact that each encoded image correspond to exactly two original images, and each set contains exactly \(N\) encoded images, is built into the design of the InstaHide algorithm: instead of randomly choosing two original images to pair together to form each encoded image \(x\), InstaHide generates two random permutations of the original images \(p^{(1)}, p^{(2)}\) and then pairs together the elements in this sequence, so \(e_1\) is generated from \(p^{(1)}_1\) and \(p^{(2)}_1\), through to \(e_N\) generated from \(p^{(1)}_N\) and \(p^{(2)}_N\). A new permutation is then generated, and the process repeats.
If InstaHide instead randomly selected sets of size approximately (but not exactly) \( |N| \) our attacks would remain approximately as effective; it would require a slightly modified scheme but preliminary experiments suggest that attack success rate remains unchanged.

### 3.3 Recovery

Given the resulting images pairings \( \phi' \), we must now reconstruct the actual values of the original images.

#### 3.3.1 Simple proof of concept

At this step, we can collect together all encoded images \( \{e_{x_i}\} \) that include the same original image \( x \) by inverting the approximated \( \phi' \) to obtain \( \phi'^{-1} \). Then, by computing the pixel-wise mean after taking the absolute value \( \tilde{x}_i = \text{mean}_{e \in S(i)} \text{abs}(e) \) we can obtain an approximation of the absolute value of the original images.

Why should this work? By taking the absolute value, we have removed any pixel-flipping information-hiding induced by multiplication with \( \sigma \). Then, by taking the mean (along each pixel independently) we can “average out” the noise from all of the other images that are mixed up with this one image, which gives us just the signal.

This recovers visually recognizable images, but (a) we have lost the sign information, and more importantly (b) we introduce a large amount of visual noise to the resulting images.

#### 3.3.2 Recovering \( \lambda \)

In order to do better, we will first need to recover not only \( \sigma \) but also the mix-up values of \( \lambda \) used. Fortunately, this step is (almost) trivial. The unordered values of \( \lambda \) are provided to the adversary by the InstaHide algorithm in the form of the labels \( z \)—each label in InstaHide is also mixed up directly.

As a result, we can (almost directly) read off the coefficients of \( \lambda \) with one exception: if InstaHide mixes up two images of the same class, then their contributions will add and we will be left with only one label with value \( l = \lambda_i + \lambda_j \). Because it is impossible to disentangle these values, we simply guess \( \lambda_i = \lambda_j = l/2 \).

#### 3.3.3 Recovering complete images, assuming no sign missing

Given this additional information of \( \lambda \) we show how it can help recover the original images, assuming that InstaHide did not perform any pixel-flipping by multiplying images with \( \{-1, 1\}^d \).

**Input.** The encoded images \( E \) (without pixel flipping), the mapping \( \phi' \), and the values of \( \lambda \).

**Output** The original images \( X \).

**Method** This attack is straightforward least squares. Let \( A \) be a \( |X| \times d \) unknown matrix (if solved for correctly, with rows corresponding to images \( x \)). Let \( B \) be a \( |E| \times d \) known matrix with rows corresponding to images \( e \).

\[
B = \begin{bmatrix}
e_1 \\
e_2 \\
\vdots \\
e_{|E|}
\end{bmatrix}
\]

Then finally let \( M \) be a sparse \( |E| \times |X| \) dimensional matrix that is zero almost everywhere except when \( \phi(i) = (j, k) \) where

\[
M_{i,j} = \lambda_{e_{i,1}} \\
M_{i,k} = \lambda_{e_{i,2}}.
\]

Therefore if \( A \) was correct then we would have that

\[
M \cdot A = B + \sigma.
\]
where $\sigma$ is the noise component corresponding to the public images (factored out). Therefore we can “just” solve for the equation

$$A = M^{-1}(B + \sigma) = M^{-1} \cdot B + M^{-1} \sigma \approx M^{-1} \cdot B$$

assuming that $\sigma$ is distributed normally. The reason this holds true is that if $\sigma$ is symmetric about zero, then the expected mean value of $M^{-1} \sigma \approx 0$.

Put differently, what we’re effectively doing is minimizing the “unexplained variance” by minimizing

$$\arg \min_{A' \in [-1,1]^{X \times d}} \|B - M \cdot A'\|_2^2$$

because the true solution to this equation would give

$$\|B - M \cdot A\|_2^2 = \|(M \cdot A + \sigma) - M \cdot A\|_2^2 = \|\sigma\|_2^2.$$

and so this approach is well justified assuming that minimizing $\sigma^2$ is the correct objective—which it is for adding isotropic Gaussian noise.

### 3.3.4 Recovering complete images on full InstaHide

It is more difficult to solve the above equation if we mask the images by multiplying with a random $\{−1,1\}^d$ vector. However, we can still rely on the same intuition as before.

To begin, observe that solving Equation (1) is the same as solving the formulation

$$\arg \min_{A' \in [-1,1]^{X \times d}} \|\sigma\|_2^2$$

such that $M \cdot A' + \sigma = B$.

This modified formulation is identical, but while Equation (1) will not generalize to the full InstaHide Equation (2) will. To do this, we modify the minimization to instead solve

$$\arg \min_{A' \in [-1,1]^{X \times d}} \|\sigma\|_2^2$$

such that $M \cdot \text{abs}(A') + \sigma = \text{abs}(B)$

where $\text{abs}$ is taken component-wise on the matrix.

Note that this formulation is (if we replace the $\ell_2$ norm objective with an $\ell_1$ norm), a mixed integer linear program. It is thus conceivable that we could search for an exact solution; however, when we try to implement this even after ten hours our straightforward implementation has not yet found an integer-feasible solution. We expect that an optimized formulation should be solvable; however, we find that an alternate (and simpler approach) performs well.

We search for $A'$ via gradient descent. Given an attempted solution $A'$ we can use the constraint $M \cdot \text{abs}(A') + \sigma = \text{abs}(B)$ to solve for $\sigma$, which then lets us compute the objective $\|\sigma\|_2^2$. Note that there is one complication here: given a single matrix $A'$, there are multiple values $\sigma$ which satisfy the above constraint. Fortunately, because we know that it is our objective to minimize $\|\sigma\|_2^2$ we can greedily choose each entry $\sigma_{ij}$ as the smaller of the two candidates. Along with being much more computationally efficient, this approach has the benefit that we can solve the $\ell_2$ norm minimization as well.

### 3.4 Results

We evaluate our attack on the two datasets considered in the original paper: CIFAR-10 and CIFAR-100. Additionally, we evaluate our attack on an unknown dataset challenge released by the authors consisting of 5,000 encoded images from an unknown distribution generated from 100 original source images.

Because our attack is hyperparameter free and independent of any particular dataset (as long as the images are the same size—fortunately, all datasets considered are $32 \times 32$) we do not need to change any details to perform the attack below.

We implement our attacks in JAX [BFH+18], a numerically accelerated version of NumPy with built-in automatic differentiation. We train our neural networks using Objax [3], a neural network framework built on JAX.

[https://github.com/google/objax](https://github.com/google/objax)
3.4.1 CIFAR-10 and CIFAR-100 Results

**Constructing the encoded dataset.** To construct our own dataset we follow the process as described in the InstaHide paper [HSLA20] and use their existing open source code. In developing our attacks, we discovered two flaws in the InstaHide implementation that were fixed by the InstaHide authors.

We take the first 100 images in the test set, and then encode this to a dataset of 5,000 total encoded images using the \( k = 6 \) InstaHide scheme described above.

Our attack is extremely effective across both of these datasets. Figure 3 shows the first 10 images of the 100 total images in the dataset. The full 100 examples are given in Appendix A.

Our attack is computationally efficient. Computing the initial all-pairs distance takes 21 minutes on a single GPU, finding the \(|X|\) approximate-cliques takes 1.5 minutes, computing the next \(|X| \times |E|\) all-pairs graph takes 18.5 minutes, and the final recovery step 1 minute. In total, this attack takes under an hour on a single GPU.

3.4.2 InstaHide Challenge Results

The InstaHide Challenge was released by the InstaHide authors as a public challenge to break InstaHide. The authors use the strongest possible version of InstaHide and release 5,000 encoded images corresponding to 100 private images. Because only the images are released it is not possible for us to accidentally cheat—however, because it is still a public challenge we do not have ground truth available and so can not visually compare our results with the actual images. As before, our attack takes under an hour to complete.

Figure 3 shows ten images we were able to recover from the original images. The complete 100 extracted images are given in Appendix A.

4 Special-purpose attacks on InstaHide

The above reconstruction attack is fully general and breaks InstaHide with a reconstruction attack under the defense settings the authors describe. We now consider what happens under two alternate scenarios.
Figure 4: Reconstruction attack on InstaHide evaluated on CIFAR-10 with a single encoding per private image. Our attack first trains a GAN to invert (i.e., “re-color”) the absolute value of the mixed image (top). When the re-coloring succeeds, the private image is extracted near-perfectly by recovering the public mixing images with highest similarity to the mixture (bottom).

4.1 Attacking InstaHide with a single encoding

Two core components of our attack on InstaHide, the clustering step and final image recovery step, exploit the fact that we have access to multiple random encodings of every private image. We now propose an alternative attack strategy that recovers private data given a single encoding of each image. We demonstrate the efficacy of this attack on CIFAR-10.

To achieve this stronger form of attack, we consider a stronger adversary (which still lies within InstaHide’s informal threat model). First, we assume that the adversary has knowledge of the distributions from which the private data $X$ and public data $P$ come from. With this knowledge alone, our attack succeeds in recovering the mask $\sigma$, thereby leaking visually-identifiable content of mixed images. Second, to recover mixed images from a single encoding, we further assume that the adversary has full knowledge of the public dataset $P$. While this latter assumption is strong, the success of our attack illustrates that if InstaHide is to provide any security even when releasing a single encoding, then this security must partially rely on the secrecy of the “public” mixing data.

Our attack proceeds in two steps (with details deferred to below). First, we train a Generative Adversarial Network [GPAM+14] to learn to “re-colorize” encoded images. That is, we learn the mapping $\text{abs}(x) \mapsto x$ where $x$ is a mixture of $k$ images. Learning this mapping necessarily requires some prior on the distribution of private data $X$ and mixing images $P$. Then, we simply compare the image similarity of the mixed image with all public images. With high probability, we recover the mixed public images via this simple process (the complexity of this step is linear in $|P|$).

We evaluate this attack on CIFAR-10 for an InstaHide scheme with $k = 4$. Since we release a single encoding per private image, we mix each private image (from the first 100 examples in the CIFAR-10 test set), with 3 images from a public set $P$ containing the remaining 9,900 CIFAR-10 test samples. The outputs of our two-stage attack are shown in Figure 4. In a majority of cases, the GAN re-coloring succeeds in recovering most of the random mask $\sigma$. The success of the second stage is contingent on the first. Given a close enough re-colorization, enumerating all public images and subtracting the ones with highest similarity to the mixture recovers a near-perfect copy of the private image. For the 100 encodings we generated, our attack recovers the 3 public mixing images in 69% of cases (and at least $2/3$ in 85% of cases).

4.1.1 Additional details

Here we provide additional details on our implementation of this attack on CIFAR-10.

Re-coloring mixed images with a GAN. We first train a GAN to learn the mapping $\text{abs}(x) \mapsto x$ where $x$ is a mixture of $k = 4$ images from the CIFAR-10 training set. Our approach directly borrows

https://github.com/Hazelsuko07/InstaHide
https://github.com/Hazelsuko07/InstaHide_Challenge
from the use of GANs to colorize grayscale images on CIFAR-10. Given the absolute value of a mixed image $\text{abs}(x)$, the generator is trained to output a mask $\hat{\sigma} \in [-1, 1]$ so that $\text{abs}(x) \circ \hat{\sigma}$ is indistinguishable (to the discriminator) from unmasked mixed images.

**Unmixing the public images.** Given a re-colored mixed image $x$, we simply iterate over the entire public dataset $P$, and compute, for each public image $p$, the **Structural Similarity Index**, $\text{SSIM}(x, p)$. We select the public image with highest similarity, subtract it from the mixture (we simply “guess” that the mixing weight is $\lambda = \frac{1}{k}$), and recurse. That is, we recompute the structural similarity with the remaining public images and repeat until we have subtracted 3 public images.

### 4.2 Pixel-perfect reconstruction by exploiting implementation flaws

All above attacks break the algorithmic foundation of InstaHide, and any implementation of InstaHide would be vulnerable to these attacks. We additionally discovered several weaknesses in the implementation of InstaHide that allow us to achieve a **pixel perfect** reconstruction of the original dataset. These implementation weaknesses are not fundamental to InstaHide, and can be easily be corrected; nevertheless, we describe this attack for completeness.

To ensure that this attacks does not taint the results of the attacks we developed in prior sections, we developed this attack only after completing all other aspects of this paper.

At a high level, this attack exploits two weaknesses in the implementation of InstaHide (and of the InstaHide Challenge):

- InstaHide masks each encoded image with a random mask $\sigma$. However, instead of using a cryptographically secure random number generator the implementation calls `torch.random`, and `numpy.random`, which uses a Mersenne Twister [MN98].
- The InstaHide Challenge releases the encoded dataset where each pixel is represented as a 32-bit floating point number, 4× more precision than typical 8-bit integers used to represent images.

#### 4.2.1 PRNG state extraction

Pseudo random number generators (PRNG), work by maintaining a state vector $v$. When calling the generator, a deterministic function is applied to the current state to yield a new number to output, and an updated state. Critically, if initialized with the same state, a PRNG will generate the same output sequence.

The InstaHide implementation uses a Mersenne Twister [MN98] PRNG, the default random number generator in NumPy, in most of its computations. This includes the randomness in the encoding, including selecting which original images will be used to generate each encoded image, generating the $\lambda$ values, choosing which public images to mix into the private images, and generating the random masks $\sigma$. This PRNG is not intended for security-sensitive purposes.

We extract the PRNG state via brute force search of the $2^{32}$ possible initial seeds. To do this we implement an efficient test that, given a potential PRNG seed, allows us to determine if the seed was correct. This allows us to check if any particular seed is correct in roughly 0.1 milliseconds. We then repeat this check for each of the $2^{32}$ possible seeds. This takes 120 CPU hours, which we parallelize across 100 cores to obtain the solution in a little over an hour.

Once we extract the PRNG seed, we can use it to compute the exact mapping $\phi$, the exact values of $\lambda$, and, most importantly, allows us to **undo the encryption operation** of multiplication by $\sigma$. Note that if InstaHide only released $\text{abs}(e)$ for each encoded image $e$, this attack would not be possible because the information would be truly destroyed.

However, because the authors insist on making an analogy to encryption (and instance hiding) by multiplying by a random $\{-1, 1\}^d$ vector, it is possible to “decrypt” the original images and recover the encoded images without sign information missing. This demonstrates that even two mathematically identical techniques can have very different failure modes in practical implementations.

---

[4]: https://github.com/karoly-hars/GAN_image_colorizing

[7]: If this was computationally intractable then stronger mathematical analysis would allow us to recover the complete state.
4.2.2 High-fidelity image reconstruction

Given all of this information ($\phi$, $\lambda$, and $E$ without sign flipping), the reconstruction attack from Section 3.3.3 applies directly. Figure 5 shows the result of this attack on the InstaHide challenge compared to the images we extract using the prior attack. All 100 reconstructed images are given in Appendix A.

4.2.3 Pixel-perfect refinement

We are able to make one final improvement that allows us to recover a pixel perfect reconstruction when given access to the public dataset. Because we have reverse engineered the PRNG seed, it turns out that not only do we get access to the function $\phi$ but we can even determine which public images were used in each encoded image—because these values are determined using the same PRNG.

As a result of this, we now have an over-determined system of equations. By replacing the noise value $\sigma$ from Equation TODO with the actual public images, this reduces the number of free variables to just $M \cdot d$ when there are $M$ original images of dimension $d$. Because the number of encoded images is greater than the number of original images (and in practice $50 \times$ as many for the Challenge) we can perfectly solve for the reconstruction.

Unfortunately we are unable to mount this attack on the actual InstaHide Challenge: the authors do not release the public dataset used to generate the challenge dataset. However, we have confirmed this attack on our CIFAR-10 case study and it works as expected.

5 Barriers against Learning through Instance Encoding

Broadening from the context of InstaHide, we consider the general approach of instance encoding and their usefulness in realizing private machine learning. We start with some basic notation and our formalization of an encoding mechanism.

5.1 Formal Definitions

Notation. Let $X$ be an instance space and $Y$ be a label space. We specify a learning problem with a tuple $(D, C, H)$ where $C \subseteq Y^X$ is a class of concept functions (from $X$ to $Y$), $D$ is a distribution of instances and $H \subseteq Y^X$ is a class of hypothesis functions. For a concept function $c \in C$, we use $D_c$ to specify the joint distribution of labels and instances $D_c \equiv (D, c \circ D)$. For a hypothesis $h$, a concept class $c$ and a distribution $D$ the risk of $h$ with respect to $c$ and $D$ is defined as $\text{Risk}_D(h, c) = \Pr_{x \sim D}[h(x) \neq c(x)]$. We use $D_1 \equiv D_2$ to specify that two distributions/random variables $D_1$ and $D_2$ are identically distributed.

The following definition formalizes a general notion of encoding that allows instance encodings to depend on a whole dataset. This e.g., can capture encoding through data augmentation.

Definition 5.1 (Dataset encoding mechanism). A dataset encoding mechanism for a learning problem $(D, C, H)$ is a potentially randomized algorithm $E: (X \times Y)^* \rightarrow (\tilde{X} \times \tilde{Y})^*$ that takes a dataset $S$ as input and outputs an encoded dataset $\tilde{S}$.

Remark 5.2. The above definition captures a very broad range of techniques to achieve privacy. For example, it captures local and central differential privacy mechanisms. Importantly, this encoding mechanism also captures InstaHide as it is allowed to be randomized. Note that, the InstaHide scheme is assumed to have access to a public dataset. To incorporate this setting, the encoding algorithm could have the full public dataset hard coded in its description and then use randomness to sample points from that dataset.

In order to better understand this encoding mechanism we consider three examples: (i) Identity mechanism: Let $E$ be the identity function. Note that using this encoding method, we can continue...
to train a learning task on the encoded data just as we could have on the original data, without an increase in the accuracy of any training done using this data. However, note that such an encoding does not offer any privacy gains. (ii) **Null mechanism:** Let $E$ be both the $\perp$ function. In this case, the encoding should hide everything about the original data, but the generated encodings are useless for any kind of training. (iii) **Local DP mechanism:** Finally, consider a DP mechanism, where $E(S)$ generates a deferentially private noisy version of $S$. In this case, we can train using the encoded dataset with some possible degradation in accuracy.

**Private learning through encoding.** We formalize an (encoding-based) privacy preserving learning protocol for a learning problem $(D, H, C)$ by a tuple $(L, E)$ where $E$ is a dataset encoding scheme and $L$ is a learning algorithm that works on encoded datasets. In the following, we formalize the accuracy and privacy requirements of such protocols.

**Accuracy.** One can define accuracy on both encoded and original examples. Here we focus on the case where the accuracy is defined on the actual examples. The protocol $(L, E)$ is $(\epsilon, \delta)$-accurate, if for all $c \in C$, $n \in \mathbb{N}$ we have

$$ \Pr_{S \sim D^n, \hat{S} \leftarrow E(S), h \leftarrow L(\hat{S})} \left[ \text{Risk}_D(h, c) \geq \epsilon(n) \right] \leq \delta(n). $$

For privacy, we define two attack models both of which are privacy notions for the encoding itself—meaning that the privacy requires the encoding to hide the sensitive information. If the encoding can hide the input so that it is hard to distinguish from other inputs, or at least hard to recover, then the model trained on encoded instances would also be private by standard post-processing arguments.

- **Dataset distinguishing attacks:** For a fixed $c \in C$, the adversary selects two datasets $S_0 = \{(x_1, c(x_1)), \ldots, (x_n, c(x_n))\}$ and $S_1 = \{(x'_1, c(x'_1)), \ldots, (x'_n, c(x'_n))\}$ such that the label frequency across $S$ and $S'$ are exactly equal. Then the encoder selects $b$ at random and encodes $S_b$ to get $\hat{S} \leftarrow E(S_b)$. Given $\hat{S}$ the adversary must decide whether $b = 0$ or $b = 1$. The advantage of the adversary against $c$ is defined by

$$ \text{Adv}(A, c) = \left| \Pr[A(E(S_1)) = 1] - \Pr[A(E(S_0)) = 1] \right|. $$

Note that this definition is incomparable to differential privacy. It is a weaker notion of privacy in the sense that the adversary is allowed to only use datasets that share the same label statistics. At the same time, it is stronger in that the adversary can use non-neighboring datasets. We find this to be a very natural definition of privacy (closely inspired by the InstaHide paper).

- **Instance recovering attacks:** A dataset $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ is encoded to $(X, Y) \equiv E(S)$ and given to the adversary. The goal of the adversary is to find a $x'$ such that $d(x^*, x_i) \leq \gamma$, for some $i \in [n]$ under some (context-dependent) meaningful metric $d(\cdot, \cdot)$.

The distinguishing attacks are harder to defend against. In the following section, we give a barrier against achieving privacy against distinguishing attacks. Note that our theoretical barrier does not rule out the possibility of privacy against instance recovering attacks. Indeed, to rule out such attacks one has to first decide on a natural metric of choice (e.g., based on some $\ell_p$ norm), which is context dependent, while our results of this section are general.

### 5.2 General Attacks

In this section, we present attacks on privacy of learning protocols equipped with an instance encoder against distinguishing attacks. The theorem bellow shows a impossibility result for achieving both accuracy and distinguishing privacy.

**Theorem 5.3.** Let $c_1$ and $c_2$ be two distinct and non-constant concept functions for inputs $X$ and labels $\{0, 1\}$. Let $S = \{(x_1, c_1(x_1)), \ldots, (x_n, c_1(x_n))\}$ and $S' = \{(x_1, 1 - c_1(x_1)), \ldots, (x_n, 1 - c_1(x_n))\}$ be two datasets sampled in a way that $\forall i \in [n], c_2(x_i) = 1$ and also $\sum_i c_1(x_i) = n/2$. If a learning protocol $(L, E)$ can operate on $S$ and $S'$ and respectively

---

*By distinct we mean $c_2$ is not identical to $c_1$ or $1 - c_1$.\footnote{By distinct we mean $c_2$ is not identical to $c_1$ or $1 - c_1$.}
learn $c_1$ and $1 - c_1$ from them with training accuracy more than 0.5, then there is a dataset distinguishing adversary for $(L, E)$ against either $c_1$, $1 - c_1$ or $c_2$ with advantage at least $1/3$. Moreover, the running time of this adversary is essentially the running time of $L$.

**Proof.** We first show that the encodings of $S$ and $S'$ are distinguishable with advantage 1, by a distinguishing algorithm $q$. The algorithm $q$ gets $S$ which is either the encoding of $S$ or the encoding of $S'$. Then it trains a model $h$ by applying $L$ on $S$. Then it queries the model $h$ on the training examples $\{x_1, \ldots, x_n\}$. Since the training accuracy of $h$ should be better than 0.5 with respect to either $c_1$ or $1 - c_1$, the algorithm can distinguish the two cases by looking at the predictions of the trained model on the training examples. In particular, if the predictions were mostly agreeing with $c_1$ the adversary outputs 1 otherwise it outputs 0. Therefore, this algorithms has distinguishing advantage 1. Namely

$$\left| \Pr[q(E(S)) = 1] - \Pr[q(E(S')) = 1] \right| \geq 1. \quad (4)$$

So far, we have showed that an algorithm can distinguish between the encoding of $S$ and $S'$. But note that we still do not have a real attack as $S$ and $S'$ are not labeled according to the same concept function. In the rest of the proof we see how we can use inequality [4] to prove that there are at least two datasets that are labeled according to the same concept function and their encodings are distinguishable.

To prove the theorem, we use three hybrid arguments. We construct two datasets $S_a$ and $S_b$ as follows. Let $S_a$ be a dataset consisting of two parts $S_a = S_{a_1} \cup S_{a_2}$ where

$$S_{a_1} = \{(x_1^{a_1}, 0), \ldots, (x_{n/2}^{a_1}, 0)\}$$

is a set of $n/2$ examples such that $c_1(x_i^{a_1}) = c_2(x_i^{a_1}) = 0$ and

$$S_{a_2} = \{(x_1^{a_2}, 1), \ldots, (x_{n/2}^{a_2}, 1)\}$$

is a set of $n/2$ examples such that $c_1(x_i^{a_2}) = c_2(x_i^{a_2}) = 1$. We also construct $S_b = S_{b_1} \cup S_{b_2}$ such that

$$S_{b_1} = \{(x_1^{b_1}, 0), \ldots, (x_{n/2}^{b_1}, 0)\}$$

and

$$S_{b_2} = \{(x_1^{b_2}, 1), \ldots, (x_{n/2}^{b_2}, 1)\}$$

such that $c_1(x_i^{b_1}) = 1$, $c_2(x_i^{b_1}) = 0$, $c_1(x_i^{b_2}) = 0$ and $c_2(x_i^{b_2}) = 1$. Note that $S_a$ is constructed in a way that its labels are consistent with both $c_1$ and $c_2$, while $S_b$ is constructed in a way that its labels are consistent with both $c_2$ and $1 - c_1$.

Now observe that $S$ and $S_a$ are both correctly labeled according to $c_1$ and share the same label statistics. Now imagine an adversary $A_{c_1}$ that outputs $S$ and $S_a$ as two challenge datasets against $c$. Then, when receiving the encoded dataset, it uses the algorithm $q$ described above to distinguish them. We have

$$\text{Adv}(A_{c_1}, c_1) = \left| \Pr[q(E(S)) = 1] - \Pr[q(E(S_a)) = 1] \right| \quad (5)$$

Similarly, we can observer that $S_a$ and $S_b$ are both labeled according to $c_2$ and share the same label statistics, therefore the encoding of $S_a$ and $S_b$ could be used as challenge datasets against $c_2$. Imagine an adversary $A_{c_2}$ that uses these two datasets as challenge datasets and then uses the algorithm $q$ to distinguish their encodings. We have

$$\text{Adv}(A_{c_2}, c_2) = \left| \Pr[q(E(S_a)) = 1] - \Pr[q(E(S_b)) = 1] \right| \quad (6)$$

Finally, we can observe that $S_b$ and $S'$ are both labeled according to $1 - c_1$ and share the same label statistics which means they could be used as the challenge datasets against $1 - c_1$. Imagine an adversary $A_{1-c_1}$ that uses these challenge datasets and then uses $q$ to distinguish their encodings. We have
Adv\((A_{1-c_1}, 1-c_1) = |\Pr[q(E(S))] = 1 - \Pr[q(E(S')) = 1]| \) \tag{7}

Putting these together, applying triangle inequality on Equations (5), (6) and (7) we have
\[
\text{Adv}(A_1, c_1) + \text{Adv}(A_2, c_2) + \text{Adv}(A_{1-c_1}, 1-c_1) \geq |\Pr[q(E(S))] = 1 - \Pr[q(E(S')) = 1]|
\]
(By Inequality (4)) \geq 1.

This finishes the proof as by an averaging argument at least one of the advantages must be at least \(1/3\).

\textbf{Remark 5.4.} Note that the impossibility theorem above does not consider the scenario where the encoding mechanism could get some auxiliary information about the concept function that it tries to learn.

6 \hspace{1cm} \textbf{Response from InstaHide Authors}

We contacted the InstaHide authors about this attack and provided them with a preliminary draft of this paper. Below their (verbatim) response.

We are grateful to the authors for sending us the manuscript for comments and congratulate them on a fairly strong and interesting attack. (Also shame on us for not using a secure random generator when creating the challenge dataset!)

We present the following caveats:

(a) InstaHide is not intended to be a mission-critical encryption like RSA. It lets users and the internet of things to use machine learning without giving eavesdroppers or servers an easy way to gather their raw data. Now we know that Google’s powerful GPU took an hour to break our challenge set of 100 images. But this may still not be be a cost-effective route for companies when millions or billions of clients are involved. There is no other cost-effective encryption method similar to InstaHide for this application.

(b) The challenge dataset corresponded to an ambitious form of security, where the encrypted images themselves are released to the world. The more typical application is a Federated Learning scenario where one obtains encrypted inputs passed through a neural net. The attacks in this paper do not currently apply to that scenario.

(c) We have suggested to the authors that their Introduction and Conclusions sections show a misunderstanding of the security guarantees of existing methods like differential privacy (DP). In particular, we described to them ways to use InstaHide that resist their attacks while providing much better utility than DP.

(d) Finally, unlike the authors, we very much hope to see the “cat and mouse” game between defenders and attackers in this space! Users sorely need novel ways to benefit from ML without having to hand over all their data. Further discussion appears on www.offconvex.org

7 \hspace{1cm} \textbf{Conclusion}

Training neural networks while preserving training data privacy is of clear importance across many settings. For this reason, training schemes that offer provable privacy, most often training with a differentially private mechanism, are currently the preferred strategy when privacy is required.

InstaHide takes a different approach. First, InstaHide aims for stronger cryptographic privacy, instead of statistical privacy as offered by differentially private stochastic gradient descent. Second, by aiming to offer a stronger version of privacy that does not come at a cost to utility or accuracy, InstaHide relies on ad hoc security arguments rather than a proof.

With the goal of gathering evidence for/against this ad hoc security argument, InstaHide released a challenge. Due to lack of formal security proofs, our work is the next logical step of the “cat-and-mouse” game of ad-hoc schemes and attacks against them. Specifically, we introduce powerful reconstruction attacks on InstaHide.
We optimized our exact attack to defeat the particular version of InstaHide released by the authors; while keeping our techniques general to capture other modifications. In particular, we expect that our attack methodology will be resilient (or will find natural extensions) to fortified extensions and modifications of InstaHide.

Furthermore, very importantly, we also aimed for developing the strongest possible attacks in this paper: complete image reconstruction. A privacy-preserving technique should aim for much stronger privacy properties—indeed, InstaHide does aim for stronger privacy properties. Even if reconstruction attacks could provably be prevented, they are not sufficient for meaningful privacy. Notably, for instance, there is no known attack that can perform training image reconstruction attacks on a neural network trained with standard non-private training—yet standard gradient descent is not considered private. In our opinion, the strength of our attacks offers further evidence of the limitations of InstaHide.

Finally, with the goal of understanding the power of instance encodings in realizing privacy preserving machine learning, we formalize a security model that captures InstaHide (and generalizations). In this model, we provide theoretical results highlighting the challenges of realizing such an encoding scheme that lead to private protocols with stronger guarantees than mere hardness of reconstruction.

We hope that the ultimate goals of InstaHide—training neural networks without sacrificing accuracy or training time—will be achievable in the future. This is an important area of research, and one that deserves significant study. In this paper, we provide evidence of limitations in InstaHide (and generalizations) in achieving this goal. As a result, we make the case for schemes with provable security properties. In our opinion, design of security and privacy preserving techniques without precise security definitions is often error prone and should be avoided.

References

[ACG+16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 308–318, 2016.

[BBKN14] Amos Beimel, Hai Brenner, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. Bounds on the sample complexity for private learning and private data release. Machine learning, 94(3):401–437, 2014.

[BFH+18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Warrender-Milne. JAX: composable transformations of Python+NumPy programs, 2018.

[BST14] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In 2014 IEEE 55th Annual Symposium on Foundations of Computer Science, pages 464–473. IEEE, 2014.

[DLB+19] Andrew Dai, Benjamin Lee, Gagan Bansal, Jackie Tsay, Justin Lu, Mia Chen, Shuyuan Zhang, Tim Sohn, Yinan Wang, Yonghui Wu, et al. Gmail smart compose: Real-time assisted writing. 2019.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Proc. of the Third Conf. on Theory of Cryptography (TCC), pages 265–284, 2006.

[DR+14] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. Foundations and Trends in Theoretical Computer Science, 9(3-4):211–407, 2014.

[GKS08] Srivatsava Ranjit Ganta, Shiva Prasad Kasiviswanathan, and Adam Smith. Composition attacks and auxiliary information in data privacy. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 265–273, 2008.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680, 2014.
[HPQ+18] Ahmed Hosny, Chintan Parmar, John Quackenbush, Lawrence H Schwartz, and Hugo JWL Aerts. Artificial intelligence in radiology. *Nature Reviews Cancer*, 18(8):500–510, 2018.

[HSLA20] Yangsibo Huang, Zhao Song, Kai Li, and Sanjeev Arora. Instahide: Instance-hiding schemes for private distributed learning. *ICML*, 2020.

[KS08] Shiva Prasad Kasiviswanathan and Adam Smith. On the semantics’ of differential privacy: A bayesian formulation. *arXiv preprint arXiv:0803.3946*, 2008.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[SCS13] Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 245–248. IEEE, 2013.

[Vad17] Salil Vadhan. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*, pages 347–450. Springer, 2017.

[WYB+10] Miles N Wernick, Yongyi Yang, Jovan G Brankov, Grigori Yourganov, and Stephen C Strother. Machine learning in medical imaging. *IEEE signal processing magazine*, 27(4):25–38, 2010.

A Additional Figures
Figure 6: Reconstruction of each of the 100 images in the CIFAR-10 encoded dataset. In each pair of columns, the upper image is the original image and the lower image is the reconstructed version of that image.
Figure 7: Reconstruction of each of the 100 images in the CIFAR-100 encoded dataset. In each pair of columns, the upper image is the original image and the lower image is the reconstructed version of that image.
Figure 8: Reconstruction of each of the 100 images in the InstaHide Challenge.

Figure 9: Reconstruction of each of the 100 images in the InstaHide Challenge, using the improved PRNG cryptanalytic attack.