Is Private Learning Possible with Instance Encoding?

Nicholas Carlini*
Google

Samuel Deng
Columbia University

Sanjam Garg
UC Berkeley and NTT Research

Somesh Jha
University of Wisconsin

Saeed Mahloujifar
Princeton University

Mohammad Mahmoody
University of Virginia

Abhradeep Thakurta
Google

Florian Tramèr
Stanford University

***Abstract*—A private machine learning algorithm hides as much as possible about its training data while still preserving accuracy. In this work, we study whether a non-private learning algorithm can be made private by relying on an instance-encoding mechanism that modifies the training inputs before feeding them to a normal learner. We formalize both the notion of instance encoding and its privacy by providing two attack models. We first prove impossibility results for achieving a (stronger) model. Next, we demonstrate practical attacks in the second (weaker) attack model on InstaHide, a recent proposal by Huang, Song, Li and Arora [ICML'20] that aims to use instance encoding for privacy.**

I. Introduction

Neural networks are increasingly trained on sensitive user data, for example building classifiers to diagnose diseases from medical images [1], [2] or help users compose emails or text messages by training on actual user data [3].

Protecting the privacy of users' data while training such models currently requires either a trusted central party with all users' data, or applying cryptographic techniques such as multiparty computation [4], [5], [6] that introduce large computation and communication overheads. In turn, preventing the trained model itself from leaking private information, e.g., with differential privacy [7], [8], [9], typically comes at a high cost in accuracy. This raises the question: *Are there other ways to perform private learning without sacrificing performance or accuracy?*

An alternate method for privately training a neural network is to first convert users' data to an encoded (private) version, and then train a non-private model on this encoded dataset [10], [11]. Since the training data has been privately encoded, the model training gets privacy "for free." We formalize this *private instance encoding* setup, and investigate fundamental limits on how well such an approach can work in theory.

We show that training a model on encoded data cannot offer privacy guarantees as strong as cryptographic techniques. Specifically, we prove that no useful encoding can resist **distinguishing** attacks of two forms. Our first attack distinguishes with non-negligible probability whether dataset S_1 or dataset S_2 was used to generate an encoded dataset. Our second attack distinguishes encodings of *instances* alone with a higher probability by relying on further assumptions about the encoding function and its utility. We formalize these definitions in Section II, and theorems in Section III.

We next study practical private instance encoding schemes. While our distinguishing attacks apply to any instantiation of instance encoding, we now attempt the stronger goal of **reconstruction** for specific instance encoding schemes. Given the encoded dataset, a reconstruction attack recovers (nearly identical) copies of individual training examples used. This privacy goal is weaker than indistinguishability, and arguably the weakest form of privacy that could be expected.

We design a reconstruction attack that breaks InstaHide [10], the state-of-the-art privacy-preserving encoding-based technique which was awarded a Bell Labs Prize [12]. InstaHide applies to image classification. Its encoding function *mixes* together multiple images [13] (with a linear pixel blend), and then it randomly flips the signs of the pixels. Our reconstruction attack (Section IV) recovers high-quality reconstructions—for example we solve the challenge released by the authors [14] and recover a nearly visually identical reconstruction of all the private encoded images. Our attack demonstrates that InstaHide fails to satisfy meaningful privacy notions.

Our attack leverages the fact that InstaHide encodings *are* distinguishable, as our theoretical results predict. Given multiple encoded images (produced by training for multiple epochs), we cluster encodings that correspond to the same source image. We then merge these encodings to recover the original image by solving a noisy linear system. Our attack sidesteps the encoding's sign flipping (which provides no privacy in itself) by simply taking the absolute value before all operations.

We further show (Section IV-H) that extensions of InstaHide offer no more privacy. Mixing *more* images into each encoding *strengthens* our attack. Even given a *single* encoding of an image, an attacker with precise knowledge of InstaHide's *public* parameters can reconstruct the encoded image near-perfectly.

*Authors ordered alphabetically.

A. The Instance Encoding Problem

In the *instance encoding* problem setup, the defender encodes a (sensitive) training dataset $S = \{(x_1, y_1), \ldots (x_n, y_n)\}$ by processing it with an encoding function E. The encoded version $\mathcal{S} \leftarrow E(S)$ is released publicly.

Any learning algorithm L can then train on the encoded set $\mathcal{S}$ to learn the *concept function* c that was used to construct the training dataset (i.e., it was used to construct the labels $c(x_i) = y_i$). The encoding is useful if get both of the following.

> **Utility-preserving.** A model trained on the encoded datset $\mathcal{S}$ should be (approximately) as accurate as a model trained on the original dataset S.
>
> **Privacy-preserving.** Given access to the encoded dataset $\mathcal{S}$, it should be difficult to learn sensitive properties about the original training dataset S.

Forms of encoding. An encoding function is an arbitrary function operating over a training dataset S, allowing for a wide range of techniques. At one extreme, a valid encoding function could take the entire training dataset S, run the learning algorithm L on all of it, and output the trained model $h \leftarrow L(S)$ as the output of the encoding. This "encoding" may (or may not) be useful or private. At the other extreme, an encoding scheme might be completely "local" and operate *independently* on each training example to produce $\mathcal{S} = \{e(x_i, y_i) : (x_i, y_i) \in S\}$.

Another natural class of encoding schemes are those that are based on "mix-up"-type operations [13] that apply a simple linear operation on a small number of instances to produce an encoded instance. Such encoding schemes usually have a nice property: they can be "decomposed" into two encoding algorithms that operate separately on instances and on labels. This class of encoding schemes includes the mix-up encoding function used in the recent InstaHide protocol [10]. Since decomposable encodings apply to instances and labels separately, in such cases we indeed deal with an *instance* encoding together with a *label* encoding that work in tandem.

Adversary capabilities. We assume the adversary is given access to the encoded dataset $\mathcal{S}$. The adversary does not have *any* access to the original dataset S In some algorithms, the encoding scheme might receive as input some public data P; for these schemes we assume the adversary has access to P. (The InstaHide algorithm, for example, takes a "private" and "public" dataset as input.)

Adversary objective. The adversary aims to learn as much information as possible about S given all available information. The most powerful attacks we consider are complete *reconstruction* attacks that recover training examples x'_i where $x'_i \approx_m x_i$ according to some similarity metric m (e.g., Euclidean distance).

We also consider more restrictive *distinguishing* (inference) attacks where the adversary aims only to determine if a particular x_i was used as training data or not.

Main question. This paper studies the following question

> *For a dataset S with instances labeled by a nontrivial concept function, is it possible to design an encoding function $E(S) = \mathcal{S}$ so that given $\mathcal{S}$ a learning algorithm L can produce an accurate model but so that the original data S remains hidden from an adversary?*

Note that we assume that the adversary has direct access to $\mathcal{S} \leftarrow E(S)$, before the learning algorithm L is run on $\mathcal{S}$.

It is easy to achieve privacy alone if E hides everything about S (e.g., define $E(x) = 0$ as a constant function), but then no meaningful learning is possible. Alternatively, if the concept function is trivial (e.g., all examples have the same label) then trivial encoding functions exist. We are interested only in encoding functions that operate over nontrivial concept functions. Our goal is to understand the barriers and trade-offs that arise between the privacy provided by the encoding function vs. the *utility/accuracy* of the learning algorithm.

B. Results

We provide negative results in the form of theoretical barriers that prevent any encoding function from protecting some forms of privacy. Moreover, we demonstrate practical attacks on specific encoding functions from the literature [10].

1) Theoretical Impossibility Results.

We prove that for the case of distinguishing attacks, it is not possible to construct nontrivial encoding functions that preserve both utility and a weak form of distinguishing privacy.

Limits of privacy with dataset encoding. We first study *distinguishing* attacks whose goal is to find out with probability (non-negligibly) more than $1/2$ which dataset out of two (different) sets S_1, S_2 has been encoded. In fact, we will show how to achieve this even if the datasets share many similarities and only differ in one example pair (with the *same* label).

Theorem 1 (Informally stated – limits of privacy based on dataset encoding)**.** *Let (E, L) be an arbitrary encoding and learning scheme with at least 51% accuracy on the original data (not the encoded data). Then for any "nontrivial" concept class C (e.g., sufficient to be closed under complement and to contain at least two distinct concepts), there is an adversary who can pick a concept $c \in C$ and two datasets S_1, S_2 and distinguish their encodings with probability $1/2 + (1/n)$, while the sets satisfy the following restrictions:*

1) *$S_1 = \{e_1\} \cup S, S_2 = \{e_2\} \cup S$ differ in one sample only.*
2) *All instances in S_1, S_2 are cleanly labeled (by c). This includes also the differing examples e_1, e_2 (i.e., $e_1 = (x_1, y), e_2 = (x_2, y)$ for $y = c(x_1) = c(x_2)$).*

Limits of privacy with instance encoding. *We now describe our next result which states the limits of what* instance *encoding (as a special form of general dataset encoding) can offer for (our minimal and natural indistinguishability-based notion of) data privacy. In this setting, we deal with a* decomposable

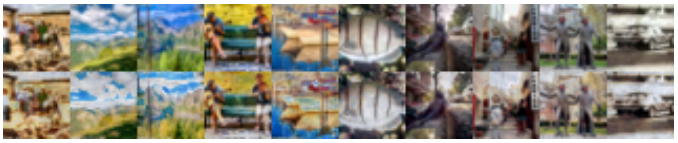


Fig. 1: Our reconstruction attack on the InstaHide Challenge, for 10 randomly selected images [14]. *Upper row:* ground truth obtained from a cryptanalytic attack [15] on the PRNG in InstaHide's implementation (Appendix B). *Lower row:* our reconstruction attack yields high fidelity image reconstructions. A complete set of the 100 recovered images are in Appendix C.

encoding, which encodes instances and their labels separately. Such decomposable encodings can cover, e.g., the mix-up operation [13] used in InstaHide [10]. For encoding $\tilde{x}$ and instance x, we write $x \in E^{-1}(\tilde{x})$ if x is one of the instances that are used for generating the encoding $\tilde{x}$.

In our next result we show barriers for achieving privacy based on instance encoding, when two conditions hold: (1) The goal of the adversary is to distinguish encodings $\tilde{x}$ where $x \in E^{-1}(\tilde{x})$ from those where $x' \in E^{-1}(\tilde{x})$ for $x \neq x'$. (2) The learning algorithm (E, L) allows some nontrivial accuracy on *encoded* strings as defined above.

Theorem 2 (Informally stated – limits of privacy based on instance encoding)**.** *Suppose the goal is to learn instances that are distributed according to distribution D and the concept class is rich enough to contain m concept functions $c_1, \ldots, c_m$ that are each balanced under D (i.e., $\Pr[c_i(D) = 1] = 1/2$) and are also independent from each other. (For example, this would be the case when the concepts contain m orthogonal half spaces and D is the isotropic Gaussian, all in dimension m). Also, suppose the protocol (E, L) has* encoded *accuracy $1/2 + \delta$ for a constant $\delta > 0$ (that can depend on the locality of E, e.g., $\delta \geq 2^{-r}$). Then, the adversary can distinguish the encodings of two randomly selected instances $x, x' \in D$ with advantage $\Omega(1)$ (over the trivial bound of $1/2$).*

We also prove a variant of Theorem 2 that does *not* rely on the richness of the concept class. This result states that if instance encoding works on a single concept function c, then one of the following happens: *either* (1) we obtain a distinguishing attack on the instance encoding, or (2) the learning error on c can be arbitrarily close to 0. This barrier applies to any setting where classifiers on c achieve accuracy bounded away (by some constant) from 1 (e.g., image classification).

2) Concrete Attack Results.

We further demonstrate that InstaHide [10], a practical instance encoding scheme, is not private. Figure 1 shows the result of our attack on the InstaHide Challenge. This challenge contains $|\tilde{S}| = 5,000$ encoded images from $|S| = 100$ original encoded images—thus, each original image has been encoded 50 different times. We are able to completely reconstruct a nearly-identical version S given access to $\tilde{S}$.

Our attack directly leverages the fact that InstaHide encodings *are* distinguishable. Given the encoded dataset, we construct a similarity function that allows us to detect when two examples $x, y \in \tilde{S}$ are derived from the same original image in S. Theorem 2 explains why such similarity function should exist as we can use a rich concept class to map encoding to a embedding space and use clustering to identify encoding that encode the same image. However our actual attack takes a different approach and leaves the computation of this similarity metric to a neural network. Specifically, we train a neural network that distinguishes whether a pair of encodings share the same input image which generalizes to unseen examples with high accuracy. This construction already consists of a privacy leak according to the definition in the prior section. However, we are able to extend the attack to complete reconstruction. Given our similarity function, we can group together multiple encoded images $T \subseteq \tilde{S}$ so that all images in the encoded subset T correspond to the same original image. and then develop a recovery function r so that $r(T) \approx x \in S$.

We further introduce a second attack that works in *linear* time and that succeeds even when given a *single* encoding of an image x_i. This attack assumes knowledge of the public images used in the InstaHide algorithm. (While we assume an adversary would have access to the "public" images, we can not use this attack on the InstaHide Challenge as it *does not* release these images.) This attack similarly produces nearly perfect image reconstructions when it succeeds, but does fail with a small constant probability in our experiments.

C. Related Work

Theorem 1 can be seen as a (dimension-independent) lower bound on the sample complexity of private PAC learning. Prior work has studied similar lower bounds on sample complexity of learning algorithms in various contexts. For example, the work of [16], [17], [18] use packing arguments to give sharp bounds for parameter/probability estimation goals, and [19] proves lower bounds on the sample complexity of differentially private algorithms that accurately answer large sets of counting queries. In addition, lower bounds on the sample complexity of differentially private (general) PAC learning were proved in [20], [21]. It might be possible to improve Theorem 1 by incorporating the data dimension, however not depending on the dimension is a postive, and we emphasize that our result comes with specific guarantees that are important: the two sets are consistent with a concept function and have the same set of labels. This makes our lower bound more amenable to real world setting, where we want to distinguish two data sets with say, the same number of cat and dog images in them.

Our attacks on data privacy of ML models are related to "membership inference" attacks [22], [23], [24], [25] as well as *model inversion* attacks [26], [27], [28], and our attack on InstaHide is a form of reconstruction attack [29], [30], [31], [32], [33], [34].

II. Privacy with Instance Encoding: Definitions

A. Formal Definitions For Learning with Instance Encoding

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 $C \subseteq Y^X$ (resp. $H \subseteq Y^X$) is a class of concept (resp. hypothesis) functions from X to Y and D is a distributions over X.[1] For a concept function $c \in C$, we use D_c to specify the joint distribution of labels and instances $(x, c(x))_{x \sim D}$ where we sample $x \sim D$ first, and then label x according to c. For a hypothesis $h \in H$, a concept class $c \in C$ and h with respect to c under the distribution D is defined as $\mathsf{Risk}(h, c) = \Pr_{x \sim D}[h(x) \neq c(x)]$.

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

Definition 1 (Dataset encoding mechanism)**.** *A dataset encoding mechanism for a learning problem (D, C, H) is a potentially randomized algorithm $E : (X \times Y)^* \to (\tilde{X} \times \tilde{Y})^*$ that takes a dataset S as input and outputs an encoded dataset $\tilde{S}$. We define two properties for such encodings:*

1) *Decomposablity: The encoding is decomposable if it performs on instances and labels separately; namely, it could be expressed using a pair of potentially randomized algorithms $E_X : X^* \to \tilde{X}^*$ and $E_Y : Y^* \to \tilde{Y}^*$ that share randomness. To encode a labeled dataset using such mechanism, one would apply E_X to instances to get $x_1, \ldots, x_m$ and E_Y to labels to get $y_1, \ldots, y_m$ and then output $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. Since such dataset encoding mechanism works on instances and labels separately, we refer to it as* instance encoding *as well.*
2) *Locality: And encoding scheme is r-local if all $x \in E(S)$ would depend only on the randomness of E and at most r examples in S. If z is an encoding that might depend on example z, we denote it by $z \in E^{-1}(z)$. Additionally, for $i \in [m]$, by $E^i(z_1, \ldots, z_n)$ we denote the process of encoding S using E and then outputting one of the encoded examples z where $z_i \in E^{-1}(z)$ uniformly at random. For decomposable encodings, we define notations E^i_X, E^i_Y for $i \in [m]$ similarly.*

Examples. We recall three natural examples: (i) *Identity mechanism:* In this case, we let E be the identity function. This trivial encoding mechanism fully preserves the utility of learning on the original data set, but it does not offer any privacy gains. (ii) *Null mechanism:* Here, we let E be the constant $?$ function. In this case, the encoding hides everything about the original data, but the generated encodings are useless for nontrivial training. (iii) *Local DP mechanism:* Here, $E(S)$ generates a deferentially private noisy version of S. In this case, we can train using the encoded dataset with some possible degradation in accuracy. Note that in all these examples, encodings can be made decomposable and 1-local

Discussion. Definition 1 captures a broad range of techniques to achieve privacy. For example, it captures local (by choosing $r = 1$) and central (by choosing $r = n$) encodings that might offer respectively local or central notions of differential (or another form of) privacy.[2] Importantly, this encoding mechanism also captures InstaHide as it is allowed to be randomized and we put no limitation on the complexity of the encoding mechanism. Indeed, the InstaHide scheme is allowed to use randomness and also have access to a public dataset. To incorporate InstaHide into our 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 fact, InstaHide comes with decomposaibility and locality properties and is a special case of our definition.

We now formalize several accuracy and privacy notions of encoding-based learning protocols. One can define accuracy on both encoded and original examples. Here we first define the accuracy on the original examples.

Definition 2 (Accuracy on plain (non-encoded) data)**.** *The protocol (E, L) is (ε, δ)-accurate, if for all $c \in C, n \in \mathbb{N}$,*

$$\Pr_{S \sim D_c^n; \tilde{S} \sim E(S); h \sim L(\tilde{S})}[\mathsf{Risk}_D(h, c) \geq \varepsilon(n)] \leq \delta(n).$$

One natural property that an instance encoding mechanism can provide is to enable the trained model to have some (perhaps weak) form of accuracy for predicting labels on the *encoded examples*. For example, suppose we use an r-local instance encoding mechanism, and that x is an encoded instance that depends on r distinct training samples, one of which is x. Then we could ask the trained model h to predict the *true* concept $c(x)$ of x when it is given the encoded sample x as input. Indeed, we define (see Definition 3) the notion of *encoded accuracy* for the model h to be the probability of satisfying $h(x) = c(x))$ when x is a random instance and $x \in E_X^{-1}(x)$. Of course this notion of accuracy may only be satisfiable in a *weak* sense, as each locally encoded instance encoding x depends on r different instances which may have different labels. However, we argue that natural instance encoding schemes could still allow the error to be bounded away from (and smaller than) $1/2$. For example, using a 2-local encoding on all pairs (x, x') of instances in a set S potentially allows getting (weak) accuracy on encoded instances of 0.75, because when the labels of (x, x') are the same, the prediction of the model h on the encoded string x could be close to 1, and in other cases it could be close to 0.5.

Definition 3 (Accuracy on encoded instances)**.** *We say the*

[1] Since we aim to prove impossibility results, focusing on the *distribution-specific* learning setting only makes our results stronger.

[2] A more general notion of locality refers to the setting where the data S is partitioned into r subsets, and then each of these subsets are independently encoded. Our 1-locality definition covers this case when each of the sets includes one example only, but the definition could be generalized easily.

protocol (E, L) *is* (ε, δ)*-accurate on encoded instances if:*

$$\Pr_{\substack{S \leftarrow D_c^n \\ \tilde{S} \leftarrow E(S) \\ h \leftarrow L(\tilde{S})}} \left[\Pr_{\substack{x \leftarrow D^{n-1} \\ \tilde{x} \leftarrow E_x^1(x;\mathbf{x})}} [h(\tilde{x}) \neq c(x)] \geq \varepsilon(n) \right] \leq \delta(n).$$

Additionally we say the protocol has balanced (ε, δ)*-accuracy if for all possible labels* y *we have*

$$\Pr_{\substack{S \leftarrow D_c^n \\ \tilde{S} \leftarrow E(S) \\ h \leftarrow L(\tilde{S})}} \left[\Pr_{\substack{x \leftarrow D^{n-1} \\ \tilde{x} \leftarrow E_x^1(x;\mathbf{x})}} [h(\tilde{x}) \neq c(x)] \geq \varepsilon(n) \right] \leq \delta(n).$$

Note that if the decomposable encoding E combines inputs with different labels, we might not expect the labeling error ε on encoded instances to be too close to 0. Indeed, if an encoded instance $\tilde{x}$ combines two samples of different labels, the learned model necessarily assigns an "incorrect" label with respect to one of the instances. Nevertheless, if the encoder samples the r inputs to combine uniformly at random, these r inputs will have consistent labels with probability 2^{-r+1} and thus non-trivial accuracy is possible whenever r is constant.

B. Threat Model Formalization

We now formalize our threat model introduced in Section I-A.

Attacking in polynomial time. There is an asymmetry between the "efficiency" requirements for algorithms that are used frequently by *hon* parties in a system, versus for algorithms that might rarely be used by malicious parties. When designing a learning scheme, one goal is to minimize its running time as much as possible. Even shaving a logarithmic factor might be important when the algorithm is run frequently and on large inputs. Attacks, on the other hand, are run rarely and in extreme cases (possibly only once). Thus, the system designer's goal is to achieve security against adversaries who might spend an *unspecified*, yet feasible, amount of resources. The reason is that we do not want to base its security on the hope that an adversary's running time cannot be improved further in the future. Indeed, modeling adversaries a polynomial-time entities is commonplace in cryptography. Here we employ the same approach for adversaries and the threat threat. Hence, we consider an attack efficient if it runs in polynomial time. Yet, we emphasize that our attacks do have *small* (absolute) running times, even though we do not optimize them.

Distinguishing vs. reconstruction attacks. Just like in encryption, our ultimate goal in private learning is to hide examples from the parties who are not supposed to know them. In both contexts, one can imagine weaker forms of attackers who can only *distinguish* the target piece of data (e.g., plaintext in cryptography or private data in the context of learning) from irrelevant (e.g., random) pieces of information. This types of attacks, e.g., are the standard attacks against *pseudorandom generators* in cryptography as well as attacks on differential privacy (e.g., membership inference attacks) in learning. A stronger, and more devastating form of attack consists of adversaries who *completely recover* the sensitive information. E.g., one-way functions are design with respect to such attackers (and not surprisingly inverting functions breaks their pseudo-randomness as well). Such attacks also exist in the context of learning and, more generally, releasing public information about private. In this work we use *both* types of distinguishing and reconstruction attacks. We prove *general* barriers against distinguishing adversaries in the context of private-learning using instance encodings, and for the concrete case of InstaHide scheme, we present the (stronger) form of adversaries, namely a reconstruction attack.

What does it mean to keep examples private? In full generality, a multi-party learning protocol consists of a set of parties $P_1, \ldots, P_n$. Each P_i has access to a dataset S_i that they use for training. We refer to the transcript of communication between the parties as T and the output of the protocol as M. The parties can also have some secret randomness $R_1, \ldots, R_n$. Within this setting, we can define two types of privacy that are both important and complementary.

Physical privacy (MPC). In this setting, there is a set of indices of honest parties I_{hon} that act based on the rules of the protocol. There is a set of indices $I_{dh} = [n] \setminus I_{hon}$ that indicates the set of parties that are dishonest. The privacy of the scheme requires that no polynomial-time adversarial algorithm A who completely controls the parties in I_{dh} cannot extract any information about $S_{I_{hon}}$ other that what one can infer by only looking at the output of the protocol (which is the final model in case of multi-party learning).

Note that in this setting, the privacy requirement does not capture leakage from the actual outcome of the protocol. For example, one can imagine a protocol that outputs the training data of all the parties, while still satisfying physical privacy trivially. Therefore, ultimately, physical privacy shall be accompanied also by a leakage analysis of the final output.

Functional privacy. Here, again the goal of the adversary is to infer some sensitive information about $S_{I_{hon}}$, but mainly by looking at the at the output M. Note that here adversary's goal is not to gain some *extra* knowledge about $S_{I_{hon}}$ beyond what T entails, but rather to find out something about $S_{I_{hon}}$ based on M compared to when M is not known. Indeed, notions such as differential privacy or k-anonymity are invented to allow us quantify the functional form of privacy. To achieve functional privacy in contexts such as searchable encryption, sometimes a leakage function $Leakage(M, R_{I_{dh}}, S_{I_{dh}})$ is defined to model what is considered acceptable to be leaked to the adversary.

We emphasize that the above two types of privacy are incomparable and complementary.

Can instance encoding provide physical privacy? Private learning with 1-local instance encoding can be seen as a protocol where each party sends only one message non-interactively. Then, using these messages, the protocol outputs a model M. Now, one can try to prove both physical and functional privacy for such a protocol.

We first observe that no dataset encoding algorithm E achieves the physical privacy required by an MPC protocol, unless the learning task is trivial (i.e., does not depend on the data) or the learning algorithm is run by a trusted party. This follows from a folklore claim (proven in [35]) that it is impossible to construct an MPC protocol where parties send only one message each and represented by the encoded dataset sent by each of the parties. We now give an intuition of this claim, tailored to the two-party case of our dataset encoding framework. In the two-party case, computation proceeds as follows: Each party encodes its dataset S_i to $E(S_i)$ and sends it to an aggregator. Next, the aggregator performs the training directly on the encoded datasets $E(S_1), E(S_2)$, yielding the trained model H. However, a malicious aggregator could also (i) sample a fresh dataset S_2', (ii) encode it obtaining $E(S_2')$, and (iii) use it along with $E(S_1)$ to obtain another model on the underlying datasets S_1 and S_2'. In fact, a malicious aggregator could learn arbitrarily many different new models on S_1. While a bit innocuous looking, such a simple attack can be quite problematic in general and is prevented by the standard notion of physical privacy for MPC protocols. But protocols that achieve this very strong notion of privacy inherently require more than one round of interaction.

This means that, to analyze the privacy of an instance encoding mechanism, we cannot follow the path of first proving physical privacy and then analyzing functional privacy. Instead, in order to understand the privacy of instance encoding protocols we must analyze the leakage of each message sent by each party individually. On the positive side, if we can show that this leakage is small, then we do not need to worry about anything else as this encoding is the only information that each party reveals about their data. Also, presence of malicious parties will not change the leakage as each party performs locally and independent of all other parties. In the next subsection we propose leakage measurement approaches for instance encoding and then in the next section, we aim at understanding the minimum possible leakage of an instance encoding based on our proposed leakage formulation.

1) Privacy Definitions for Instance Encoding

Private learning through instance encoding. We now define a minimal privacy notion for (encoding-based) learning protocols (E, L) for a learning problem (D, H, C) where E is a dataset encoding scheme and L is a learning algorithm that works on encoded datasets. The definition is of the "cryptographic" indistinguishability flavor.

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. We stress that both notions below can be studied for dataset encodings and the special case of instance encodings (where E is an instance encoding).

Definition 4 (Instance distinguishing attacks for dataset encoding mechanisms)**.** The adversary A selects a concept function c and instances $\{x_0, x_1, \ldots, x_n\}$ such that $c(x_0) = c(x_1)$ and sends them to the challenger. The challenger shapes sets $S = \{(x_i, c(x_i)) \mid 2 \le i \le m\}$, $S_0 = \{(x_0, c(x_0))\} \cup S$ and $S_1 = \{(x_1, c(x_1))\} \cup S$. Then the challenger samples a random bit $b \in \{0, 1\}$, encodes S_b to get $S' = E(S_b)$, and sends S' to the adversary. Given S' the adversary announces its guess b' (about b). The advantage of the adversary (against E) is defined as $|p - 1/2|$ where p is the probability that $b = b'$.

Note that this definition captures a weaker notion compared to differential privacy, as the sets are both consistent with the same concept function, and even where they differ the two points still have the same label. In fact, when we prove limits of privacy under Definition 4, the adversary only states the distribution of the instances in the set S without picking them!

Next, we consider a slightly weaker distinguishing game for the special case of instance encodings where the attacker is given an encoding of just one sample. This makes the task of distinguishing potentially easier for the attacker. This setting is inspired by the InstaHide framework, but it is more general.

Definition 5 (Instance distinguishing attacks for instance encoding mechanisms)**.** This security game is defined for an instance encoding mechanism $E = (E_X, E_Y)$. The adversary A selects a distribution D, a concept function c, and two instances x_0 and x_1 such that $c(x_0) = c(x_1)$. Then the encoder samples $S = (x_2, x_3, \ldots, x_n) \leftarrow D^{n-1}$ and a bit $b \in \{0, 1\}$ and encodes $E_X^1(x_b, x_2, \ldots, x_n)$ to get $\tilde{x}$. Given $\tilde{x}$ the adversary must decide whether $b = 0$ or $b = 1$ by outputting b'. The advantage of the adversary (against E) is defined as $|p - 1/2|$ where p is the probability that $b = b'$.

Finally, we consider a weak form of privacy that prevents an adversary from recovering parts of an input given its encoding.

Definition 6 (Instance recovering attacks)**.** A dataset $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ is encoded to $(\tilde{X}, \tilde{Y}) \leftarrow E(S)$ and given to the adversary. The goal of the adversary is to find a x such that $d(x, x_i) \le \gamma$, for some $i \in [n]$ under some (context-dependent) metric $d(\cdot, \cdot)$.

Distinguishing attacks are harder to defend against. In the following section, we give a barrier against achieving privacy against distinguishing attacks. Note that our result does not rule out the possibility of privacy against instance-recovering attacks. Indeed, to rule out such attacks, one has to first choose a natural metric (e.g., based on some ℓ_p norm), which is context dependent. In contrast, our results in Section III are general.

III. Barriers for Privacy with Instance Encoding

In this section, we present distinguishing attacks against learning protocols equipped with an instance/dataset encoder. We first prove a theorem in the most general setting. Namely, we consider general dataset encoding mechanisms and show the existence of dataset distinguishing attacks.

Due to space limitations, all proofs are moved to Appendix A.

Theorem 3 (Formal statement of Theorem 1). Let c_1 and c_2 be two distinct[3] and non-constant concept functions for inputs X and labels $\{0,1\}$. Let D be a distribution over X such that $E[c_2(D)] = 0.5$ and $E[c_1(D)] = 0.5$. If a protocol (E, L) can achieve $(0.5, 1, \varepsilon)$-accuracy on plain data over both D_{c_1} and D_{1-c_1}, then there is an dataset distinguishing adversary for (E, L) against either $c_1, 1-c_1$ or c_2 with advantage at least $(0.99 - 2\varepsilon(n))/3n$ (according to Definition 4), where n is the size of the dataset. Moreover, the running time of this adversary is essentially the running time of L.

Discussion. Theorem 3 gives a distinguishing attack of advantage $(1/n)$. Since the two datasets used by the adversary in the proof are neighbors (i.e., differ in one point), this also implies a lower bound on the sample complexity of differentially private learners (based on the level of differential privacy). This result further shows that none of the restrictions on the adversary (as stated in Theorem 3) can limit the adversary's distinguishing advantage (or the corresponding "ε" in a candidate differentially private scheme) to $o(1/n)$. In fact, the proof of Theorem 3 shows something stronger: the adversary will not pick the core set S that is shared between S_0, S_1, but rather that set is sampled from a distribution chosen by the adversary. Finally, we note that the complexity of the concept functions in Theorem 3 cannot be reduced to having only one concept function. That is because, if $C = \{c\}$, the learner can basically ignore the data and just output a canonical representation of c, leading to a perfectly private scheme.

The impossibility result above does not consider the scenario where the encoding mechanism can get some auxiliary information about the concept function. Specifically, we assume that the only information that the encoder obtains from the underlying concept is through the dataset. In fact, if the concept function c was directly known to the encoding mechanism, it could simply output a description of c and hide the input data.

Next, we consider the setting of local instance encodings that are applied independently to each training sample (i.e., a local encoding). Our first result applies to learning tasks with a rich class of concepts, as formalized hereafter.

Definition 7 (Rich concept class). For concept class C and given parameters $m \in \mathbb{N}$, $\varepsilon \in \mathbb{R}^+$, we say that the concept class C is (m, ε)-rich with respect to distribution D, if there exists a vector $F = (c_1, \ldots, c_m)$, $c_i \in C$ with the following property: For any configuration $f \in \{0,1\}^{|F|}$

$$\Pr_{x \leftarrow D}\left[\frac{|F(x) - f|}{|F|} \geq \varepsilon\right] \geq 0.99$$

It is easy to see if that if the concepts $c_1, \ldots, c_m \in D$ are all balanced and orthogonal (the probability of every output $f \in \{0,1\}^m$ to be produced by them over a random $x \leftarrow D$ is 2^{-m}), then by standard Chernoff-type arguments, the (m, ε)-richness property holds for any constant $\varepsilon > 0$ and sufficiently large m. (The balanced and orthogonal setting was used as a special case when stating Theorem 4 informally in Section 3.) We now state the formal version of our result.

Theorem 4 (Formal statement of Theorem 2: Barrier for privacy with instance encoding on a rich concept class). Consider a learning problem (D, C, H) where $C, H \subseteq \{0,1\}^X$ and where C is (m, ε)-rich according to Definition 7. If a learning protocol with encoding (E, L) has encoded accuracy $\varepsilon(\cdot)$ on this problem. Then, for any $c \in C$ there is an instance distinguishing attack (according to Definition 5) $A^{L(\cdot), E(\cdot), F(\cdot), D}$ that has oracle access to $E(\cdot)$, L, F and a sampler for D and gets advantage $0.99 - \frac{\varepsilon(n)}{\varepsilon}$ against E according to Definition 5. The expected running time of this adversary is $O(\frac{m}{1-\varepsilon(n)})$. Moreover, the attacker's samples (x_0, x_1) are sampled jointly from the same distribution D conditioned on labels being the same.

Discussion. The idea behind the proof of Theorem 4 is that if the learned model has non-trivial encoded accuracy (i.e., we can predict the label of an instance from its encoding), then this leakage already implies a (possibly weak) distinguishing attack between encodings. To amplify the attack's distinguishing power, we leverage the fact that we can learn multiple concept functions from the class C using the same encodings.

Theorem 4 shows a barrier against achieving both indistinguishability privacy and encoding accuracy on a rich class of concept functions. Theorem 5 below shows a barrier for the orthogonal case where the encoding can depend on the concept function itself (e.g., if there is just one concept to learn). In particular, for the following theorem, we do not require the protocol to work for multiple concept functions and it can be tailored to a specific concept function. The same argument we use to prove Theorem 4 above will not work anymore, as the protocol might use an entirely different encoding for different tasks and a classifier trained for one task will not be a good distinguisher for the encodings of other tasks.

Note that, in the extreme case, the encoding could completely depend on the concept function $c \in C$. For example, imagine an encoding algorithm that maps each instance to its correct label. This encoding is perfectly secure against the distinguishing attacks of Definition 5. This encoding can also achieve 100% accuracy if an identity classifier is applied to it. Therefore, there is no privacy versus accuracy trade-off for this case. However, we can still prove some barriers against privacy if we assume that learning a perfectly correct classifier is hard. Bellow, we show that if an encoding achieves both reasonable privacy and accuracy, then it is possible to efficiently extract an almost-perfect classifier from it.

Theorem 5 (Barriers for privacy with instance encoding on a single concept). Consider a learning problem (D, C, H) where $H, C \subseteq \{0,1\}^X$. Also assume that for a concept $c \in C$, $\Pr[c(D) = 1] = 0.5$. Consider an efficient learning protocol with decomposable instance encoding (E, L) that has balanced

[3] By distinct we mean c_2 is not identical to c_1 or $1 - c_1$.

(;) accuracy on encoding for and according to distribution D. Then, for any $\in [0,1]$, one of the following is correct:

> **Lack of privacy:** There is an efficient attack with oracle access to L, E and D_c, that runs in expected time $O(m/\gamma(m) + m/\varepsilon^2)$ and has average advantage (according to Definition 5) at least $\frac{1}{2}$ (m) in winning in the instance distinguishing game (Definition 5).
>
> **Very high accuracy:** There is an efficient learning protocol (L', E') that learns this problem (privately) using m samples and outputs a classifier h' (with running time $O(m/\varepsilon^3)$) that has accuracy at least $1-t$.

This theorem shows that if an encoding function makes all examples of a class indistinguishable from each other, then that encoding must contain almost all the information that a perfect classifier has (and this information can be extracted efficiently). This shows a barrier against privately learning tasks that have a lower bound on their sample complexity. For example, if we know that a problem (D, H, C) is not learnable with accuracy more that 95%, then it is not possible to learn it privately with accuracy more than around 50% on the encoded data.

IV. An attack on InstaHide

The above formal analysis applies to any encoding-based scheme. To make our analysis concrete, we now introduce a reconstruction attack on InstaHide [10], an instance-encoding scheme published at ICML 2020 and awarded the 2nd place 2020 Bell Labs Prize. Given access to a set of encoded images, this attack recovers the original images that were used to generate the encoding.

A. Background

InstaHide proceeds as follows. First, gather a large public dataset $p \in P$, e.g., of arbitrary images from the Internet. Then, generate the encoded dataset $(e, z) \in E$ (representing encoded images e with encoded labels z) by assigning

$$E \quad (XMix(\{x_i, x_j\}, p,), YMix(y_i, y_j,) : ((x_i, y_i), (x_j, y_j)) \in X, p \quad P, |p| = k \quad 2 :$$

The core algorithms in InstaHide, $XMix$ and $YMix$, are defined as follows.

$$XMix(x, p,) = \left(\sum_{i=1}^{2} x_{i} \; _i + \sum_{i=3}^{k} p_{i-2} \; _i \right) \Big/ P$$

with chosen uniformly at random such that $\sum_i \; _i = 1$; the mask chosen uniformly at random from $\in \{-1, 1\}^d$, and where $a \; b$ denotes element-wise multiplication. The function $YMix$ is much simpler and given by

$$YMix(y_i, y_j,) = y_i \; _1 + y_j \; _2$$

with addition 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 training epochs (e.g., 50 or 100). The authors argue InstaHide is secure for $k \geq 4$, with the strongest version at $k = 6$ (e.g., the InstaHide Challenge released by the authors uses $k = 6$).

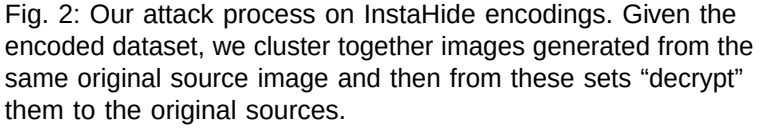


Fig. 2: Our attack process on InstaHide encodings. Given the encoded dataset, we cluster together images generated from the same original source image and then from these sets "decrypt" them to the original sources.

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

B. Attack Overview

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

1) **Remove instance hiding:** Replace the encoded dataset by

$$E \quad \{ abs(e) : e \in E \}$$

which nullifies the sign flipping step in $XMix$.

2) **Cluster encoded dataset:** Given these absolute-value images, we recover the mapping that determines which original images were used to generate each encoded image. We achieve this by training a neural network to detect when two encodings were generated from the same original image. This lets us build a graph of pairwise similarity between encodings, from which we can extract one clique per original image with the vertices in this clique corresponding to the encodings generated from that original image.
3) **Recover original images:** Then, given the encoded images and the mapping , we recover (an approximation of) the original labeled images X .
This step involves 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 [14].

C. Clustering

The purpose of the clustering stage is to recover the function that maps original source images to 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, where each set has size $|S_i| = 2N$. At the end of this step, we will know which encoded images were generated using each original image x_i.

This stage follows five steps.

1) Create a pairwise similarity function $\text{sim}(e_i, e_j) \to [0, 1]$ 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) Find sets $\{S_j\}_{j=1}^{|X|}$ by finding 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.

1) Learning 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 $\text{sim}(e_i, e_j)$ is (usually) 1 if $\mathcal{T}(e_i) \cap \mathcal{T}(e_j) \neq \emptyset$ 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 10^{-4} for 10^6 steps. We use a 32x32 downsampling of ImageNet as the public dataset following the process described in [10], 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.

2) Constructing 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 $\text{sim}(e_i, e_j)$.

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.

3) Identifying Densely Connected Cliques

Inputs: The weighted graph G from the prior subsection.

Outputs: A coloring of the vertices into $|X|$ 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

$$\bigcap_{e \in S^{(i)}} \mathcal{T}(e) = 1 \quad \forall i \qquad \bigcap_{e \in S^{(i)} \cup S^{(j)}} \mathcal{T}(e) = \emptyset \quad \forall i \neq j.$$

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. Note that each encoded image is actually created from 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

$$\text{create}(S, M) = \underbrace{\text{insert}(\text{insert}(\ldots(\text{insert}}_{\text{repeated } M \text{ times}}(S))\ldots)$$

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 $\{T^{(i)}\}_{i=1}^{|E|}$.

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 $|\bigcap_{s \in T(i)} (s)| > 0$ (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 $\text{insert}(\{e_i, e_j\})$ we are guaranteed to have that e_i and e_j share at least one original image x (formally, $|(e_i) \cap (e_j)| > 0$). With probability $\frac{1}{|X|}$ we should expect $|(e_i) \cap (e_j)| = 1$ 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 $(e_i) = (x_a, x_b)$, then both of the original images x_a and x_b have equal probability of also being part of some other encoding e_j. Thus, consider each encoded image e that is generated using the image x_b. The probability that

$$x_b \in \left[\bigcap_{e2 \quad ^1(x_b)} @ \quad (x) A \right]_{x2 \text{ create } (e)}$$

is exactly $1=2^N$, as this happens only if each call to $\text{create}(e)$ 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 $< |T^{(i)} \cap T^{(j)}| < N$ for an integer greater than zero. We can still solve this problem approximately, however. Given the $|E|$ sets, we want to cluster them into $|X|$ clusters-of-sets where we maximize the similarity of the sets in individual clusters. To do this, we perform k-means clustering on these sets (with $k = |X|$), where the distance between sets is defined as $d(s,t) = \frac{|s \setminus t|}{|s \cup t|}$. We run this to cluster the sets into $|X|$ different clusters and then choose one representative (arbitrarily) from each cluster to form the sets $S^{(i)}$.

4) Computing Similarity Between Encodings and Cliques

Inputs: The encoded images E, 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 just computes the average $\frac{1}{|S|} \sum_{v \in S} \text{weight}(e, v)$ for each $S \in S$.

We can do better, though. This similarity graph was constructed with a neural network that receives two encoded images and tests whether they share an original image. Our problem is now easier: we have $|S^{(i)}|$ encoded images, all of which (probably) belong to the same original image x_i and we want to test if an encoded image e also belongs to the same original image x_i.

We thus train a new similarity neural network to return 1 if an encoded image e shares the same original image as a set of examples $\{e_1, e_2, e_3, \ldots\}$. We find experimentally that we reach diminishing returns once we provide the neural network with more than 4 examples. This new task is 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 images in each set S at random. Then, we compute the distance from each $e \in E$ to the four representatives from each set, giving us a bipartite graph connecting the $|X|$ sets to the $|E|$ examples.

D. Assigning an Encoded Image to an Original Image

Inputs: The new similarity graph G'.

Outputs: A mapping $'$ that maps encoded images to original images. Ideally, we will have that $' = $.

Method: We can solve the final assignment problem with a single call to min cost max flow [36]. 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 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 to exactly two sets S, and assign each set to exactly $|N|$ distinct examples e_i, exactly satisfying the constraints specified for . This gives us the mapping function $'$.

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 images to pair together to form each encoded image, InstaHide generates two random permutations of the original images $p^{(1)}$, $p^{(2)}$ and then pairs together the elements in this sequence: e_1 is generated from $p_1^{(1)}$ and $p_1^{(2)}$, through to e_N generated from $p_N^{(1)}$ and $p_N^{(2)}$. 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 effective; it would require a slightly modified scheme but preliminary experiments suggest that attack success rate remains unchanged.

E. Recovery of the Original Images

Given the resulting images pairings $'$, we must now reconstruct the actual values of the original images.

1) A Simple Proof of Concept

At this stage, we can gather all encoded images $\{e_i, e_j\}$ that include the same original image x_i by inverting the recovered mapping $'$. Then, by computing the pixel-wise mean after taking the absolute value $x_i = \text{mean}_{e \in S^{(i)}} \text{abs}(e)$ we obtain an approximation of the absolute value of the original images.

Why does this work? By taking the absolute value, we remove the pixel-flipping information-hiding induced by multiplication

with . Then, by taking the pixel-wise mean we "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.

2) Recovering the Mix-Up Values

In order to do better, we will rst need to recover not only but also the mix-up values of used. Fortunately, this step is (almost) trivial. The unordered values of 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 coef cients of with one exception: if InstaHide mixes up two images of the same class, then we obtain a single label with value $l = {}_i + {}_j$. Because it is impossible to disentangle these values, we simply guess $_i = {}_j = l/2$.

3) Recovering Original Images Assuming no Sign Flipping

Given this additional information of we show how to improve the recovery of the original images. To simplify exposition, we begin by assuming that InstaHide does not perform any pixel- ipping by multiplying images with $f \ 1; 1g^d$.

Inputs: The encoded images (without pixel ipping), the mapping ^{0}, and the values of .

Outputs: The (near) original images .

Method: This attack is straightforward least squares. Let be a $jX j \ d$ unknown matrix (if solved for correctly, with rows corresponding to images). Let B be a $jE j \ d$ known matrix with rows corresponding to images

$$B = \begin{bmatrix} e_1 & e_2 & \cdots & e_{jE j} \end{bmatrix}^T$$

Then nally let let M be a sparse $jE j \ jX j$ dimensional matrix that is zero almost everywhere except when $(i) = (j; k)$ where

$$M_{i;j} = {}_{e_i;1} \quad \text{and} \quad M_{i;k} = {}_{e_i;2}:$$

Therefore if A was correct then we would have that

$$M \ A = B + :$$

where is the noise component for the public images (factored out). Therefore we can "just" solve for the equation

$$A = M^{-1}(B +) = M^{-1} B + M^{-1} \quad M^{-1} B$$

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

Put differently, what we're effectively doing is minimizing the "unexplained variance" by minimizing

$$\arg\min_{A^0 2 [\ 1;1]^{jX j \ d}} kB \quad M \ A^0 k_2^2 \qquad (1)$$

because the true solution to this equation would give

$$kB \quad M \ Ak_2^2 = k(M \ A +) \quad M \ Ak_2^2 = k \ k_2^2:$$

and so this approach is well justi ed as long as minimizing ^{2} is the correct objective—and it is for isotropic Gaussian noise.

4) Recovering Original Images on Full InstaHide

It is more dif cult to solve the above equation if we mask the images by multiplying with a random $f \ 1; 1g^d$ vector. However, we can still rely on the same intuition as before.

Solving Equation 1 is the same as solving the formulation

$$\arg\min_{A^0 2 [\ 1;1]^{jX j \ d}} k \ k_2^2 \qquad (2)$$
$$\text{such that} M \ A^0 + = B:$$

This modi ed 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

$$\arg\min_{A^0 2 [\ 1;1]^{jX j \ d}} k \ k_2^2 \qquad (3)$$
$$\text{such that} M \quad \text{abs}(A^0) + = \text{abs}(B)$$

where abs is taken component-wise on the matrix.

We search for A^0 via gradient descent. Given an attempted solution A^0 we can use the constraint $M \ \text{abs}(A^0) + = \text{abs}(B)$ to solve for , which then lets us compute the objective $k \ k_2^2$. There is one complication here: given a matrix A^0, there are multiple values which satisfy the above constraint. Fortunately, because we know that it is our objective to minimize $k \ k_2^2$ we can greedily choose each entry as the smaller of the two candidates. Along with being much more computationally ef cient, this approach has the bene t that we can solve the $_2$ norm minimization as well.

F. Adjusting Color Saturation Levels

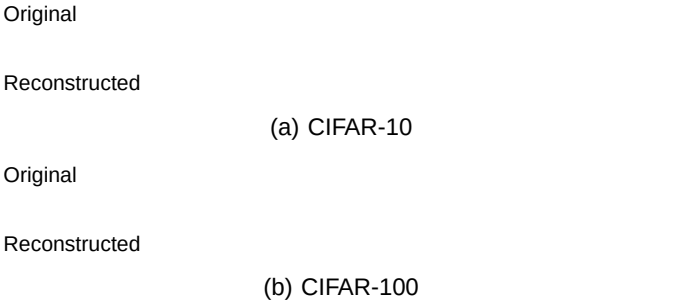
Given the recovered original images, we repair their saturation levels to better re ect the distribution of natural images.

Inputs: The reconstructed images .

Outputs: The color-adjusted images $_{fixed}$.

Method: We nd that while the reconstructions are of high quality, saturation curves are misaligned between the original and the reconstructed inputs. Manual adjustment of these curves is effective, but we can develop an automated approach.

We train a tiny (73 total parameter) neural network for this task. The network receives as input a single pixel (3 RGB colors), has a 10-neuron hidden state, and then outputs a single pixel with the new color values. To train this model, we create a new challenge using our own images, run the full attack up to this point, and record the reconstructed images along with the original images. Then, we create a training dataset mapping the reconstructed pixel values onto the original pixel values. We train this model for one epoch on 100,000 training examples, and then apply it on the nal images for each of our attacks.



(a) CIFAR-10

(b) CIFAR-100

Fig. 3: Our reconstruction attack on InstaHide evaluated on CIFAR-10 (a) and CIFAR-100 (b). The top row of each sub gure contains 10 original images that were encoded, and the bottom row our reconstruction of that image.

G. Results

We evaluate our attack on the two datasets considered in the original paper: CIFAR-10 and CIFAR-100. We further 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 32×32) we do not need to change any details to perform the attack below.

We implement our attacks in JAX [37], a numerically accelerated version of NumPy with built in automatic differentiation. We train our neural networks using Objax,[4]

1) CIFAR-10 and CIFAR-100 Results

Constructing the encoded dataset. We construct our own dataset by using the authors existing open source code.[5]

We take the rst 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 rst 10 images of the 100 total images in the dataset. The full 100 examples are given in Appendix C.

Our attack is computationally ef cient. Computing the initial all-pairs distance takes two hours on one GPU, nding $\binom{k}{4}$ cliques takes 2 CPU-hours, computing $\binom{|X|+|P|}{4}$ all-pairs graph takes 19 minutes, and the nal recovery step takes 1 minute. In total, the attack took 2 GPU hours and 2 CPU hours.

2) InstaHide Challenge Results

The InstaHide Challenge [14] was released by the InstaHide authors as a public challenge to break InstaHide. The authors use the strongest version of InstaHide and release 5,000 encoded images corresponding to 100 private images. Because only the encoded images are released, we do not have ground truth available and so can not visually compare our results with the actual images. Our attack takes under an hour to complete.

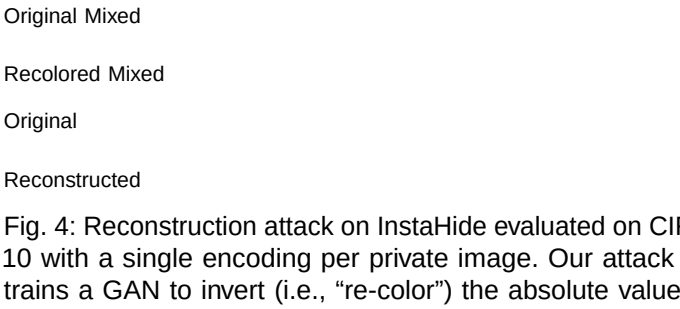


Fig. 4: Reconstruction attack on InstaHide evaluated on CIFAR-10 with a single encoding per private image. Our attack rst trains a GAN to invert (i.e., “re-color”) the absolute value of the mixed image (top). When the re-coloring succeeds, the private image is extracted near-perfectly by subtracting the public images with highest similarity to the mixture (bottom).

Figure 1 shows ten of the original images that we recovered. The complete 100 recovered images are given in Appendix C.

H. Analysis of InstaHide's Security Parameters

The above reconstruction attack is fully general and breaks InstaHide under the defense settings described by the authors and the released InstaHide challenge. However, InstaHide has two “security parameters” that are claimed to increase the security if set appropriately. Speci cally,

- The total number of released images $|E| = |P|N$ as controlled by the number of times the dataset is replicated N. A larger N is necessary for accurate models (e.g., the InstaHide challenge sets $N = 50$, but the security is claimed to increase with smaller N.
- The MixUp- k value controls the number k of original images used to form a single encoded image. The authors show that increasing k decreases accuracy, but claim that increasing k improves security.

We now introduce two attacks that show neither of these security parameters signi cantly increase the actual security of InstaHide. Even if $k > 100$, InstaHide remains broken under the same attack as above, and a new attack we develop can break InstaHide when $N = 1$ epoch of data is released.

1) Attacking InstaHide With a Single Encoding

Two core components of our attack on InstaHide, the clustering step and nal 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.

To achieve this stronger form of attack, we consider a stronger adversary (which lies within InstaHide's threat model). First, we assume that the adversary has knowledge of the distributions of the private data X and public data P. With this knowledge alone, our attack succeeds in recovering the mask λ, thereby leaking visually-identi able content of mixed images. Second, to recover mixed images from a single encoding, we further

[4] https://github.com/google/objax
[5] https://github.com/Hazelsuko07/InstaHide

assume that the adversary has full knowledge of the public dataset $\mathcal{P}$. 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 $\mathcal{P}$.

Our attack proceeds in two steps (with details deferred in below). First, we train a Generative Adversarial Network [38] to learn to "re-colorize" [39] encoded images. That is, we learn the mapping $\text{abs}(x) \mapsto x$ where x is a mixture of k images. Learning this mapping requires some prior on the distribution of private data $\mathcal{X}$ and mixing images $\mathcal{P}$. Then, we simply compute the image similarity of the mixed image with all public images and recover the mixed public images via this simple process (the complexity of this step is linear in $|\mathcal{P}|$).

Evaluation. We evaluate this attack on CIFAR-10 for an InstaHide scheme with $k = 4$. Since a single encoding is released per private image, we mix each private image (from the first 100 examples in the CIFAR-10 test set), with images from a public set $\mathcal{P}$ containing the remaining 9,900 test samples. The outputs of our two-stage attack are shown in Figure 4.

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 borrows from the use of GANs to colorize grayscale images.[6] Given the absolute value of a mixed image $\text{abs}(x)$, the generator is trained to output a mask $\hat{\lambda} \in [-1, 1]$ so that $\text{abs}(x) \cdot \hat{\lambda}$ is indistinguishable (to the discriminator) from unmasked mixed images. In a majority of cases, the GAN re-coloring successfully recovers most of the random mask.

In the second step, given a re-colored mixed image x, we iterate over the public dataset $\mathcal{P}$ and compute, for each public image p, the Structural Similarity Index $\text{SSIM}(x, p)$ [40]. 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 $k-1$ public images. This step could potentially be improved by learning a similarity function, as we did in Section IV-C for our attack on the InstaHide challenge.

The success of the second step is contingent on the first. Given an accurate re-colorization, subtracting the public images 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 of 3 in 85% of cases.

2) Attacking InstaHide with a Larger Mixup k: Recall that the parameter k in InstaHide controls the number of total images mixed to form one encoded image. The authors argue that larger values of k result in stronger versions of the scheme. Specifically the authors claim breaking InstaHide requires $O(|\mathcal{P}|^k)$ work. Our attack above breaks InstaHide for the setting $k = 6$, however as this is a security parameter it is unreasonable to ask if larger values of k would prevent our proposed attack.

We find it would not. Surprisingly, we find that as k gets larger our reconstruction attack becomes better. In Equation 2 we treat the noise η, which is only present because of the public images, as pointwise Gaussian noise. When $k = 6$ this is already an acceptable approximation and the attack succeeds. But as k grows larger, this approximation gets better and better. In fact, for $k \to \infty$ we should expect that the average over all public images will result in no noise.

V. Conclusion

Training neural networks while preserving data privacy is of clear importance in many settings [22], [41]. In principle, training models with provable privacy guarantees is possible: secure multiparty computation [42], [4], [43] or fully homomorphic encryption [44], [45] can provide provable cryptographic guarantees on the confidentiality of user data during training, and differential privacy [46], [47], [9] can bound the statistical leakage of training data for the final model.

As these provable guarantees can come at a high cost in performance and accuracy, recent work has proposed alternative instance-encoding schemes that aim to offer strong privacy guarantees with little overhead. Instantiations of these proposals, such as InstaHide [10], often lack rigorous notions of privacy and rely on ad-hoc security arguments.

We have formalized natural (cryptographic) privacy notions for instance encoding schemes, and have proven strong barriers against achieving these. Specifically, we have shown that any encoding scheme that allows for training accurate models cannot provide similar indistinguishability guarantees as MPC.

We have thus further asked whether existing instance-encoding schemes satisfy weaker privacy notions, in particular a very weak notion of security against reconstruction attacks. We have shown successful reconstruction attacks on InstaHide [10], [11], and in particular we have succeeded in fully breaking the challenge posted by the authors. Our attacks directly contradict the heuristic privacy arguments that underlie the InstaHide construction. As similar constructions underlie other recent proposals for private training [11] and inference [48], these heuristic schemes can likely be defeated by similar attacks.

The goal of privately training neural networks without sacrificing performance is notable, and we hope it will be achievable in the future. Yet, to enable meaningful progress, proposed schemes should strive to provide precise and falsifiable privacy claims, in place of ad-hoc security arguments.

Acknowledgements

We thank Shuang Song, the InstaHide Authors, and the anonymous reviewers for feedback on early drafts of this paper.

This paper was supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200,

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

NSF CNS Award 1936826, NSF grants CNS-1936799 and CCF-1910681, and research grants by the Sloan Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

REFERENCES

[1] A. Hosny, C. Parmar, J. Quackenbush, L. H. Schwartz, and H. J. Aerts, "Artificial intelligence in radiology," Nature Reviews Cancer, vol. 18, no. 8, pp. 500–510, 2018.

[2] M. N. Wernick, Y. Yang, J. G. Brankov, G. Yourganov, and S. C. Strother, "Machine learning in medical imaging," IEEE signal processing magazine, vol. 27, no. 4, pp. 25–38, 2010.

[3] M. X. Chen, B. N. Lee, G. Bansal, Y. Cao, S. Zhang, J. Lu, J. Tsay, Y. Wang, A. M. Dai, Z. Chen et al., "Gmail smart compose: Real-time assisted writing," in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 2019, pp. 2287–2295.

[4] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017, pp. 19–38.

[5] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2018, pp. 35–52.

[6] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for federated learning on user-held data," arXiv preprint arXiv:1611.04482, 2016.

[7] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in Theory of cryptography conference. Springer, 2006, pp. 265–284.

[8] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization." Journal of Machine Learning Research, vol. 12, no. 3, 2011.

[9] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016, pp. 308–318.

[10] Y. Huang, Z. Song, K. Li, and S. Arora, "Instahide: Instance-hiding schemes for private distributed learning," in ICML, 2020.

[11] M. Raynal, R. Achanta, and M. Humbert, "Image obfuscation for privacy-preserving machine learning," arXiv preprint arXiv:2010.10139, 2020.

[12] Nokia, "Nokia announces 2020 Bell Labs Prize winners," 2020. [Online]. Available: https://www.nokia.com/about-us/news/releases/2020/12/03/nokia-announces-2020-bell-labs-prize-winners

[13] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," arXiv preprint arXiv:1710.09412, 2017.

[14] Y. Huang, Z. Song, K. Li, and S. Arora, "A challenge for instahide," 2020. [Online]. Available: https://github.com/Hazelsuko07/InstaHide_Challenge

[15] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Cryptanalytic attacks on pseudorandom number generators," in International workshop on fast software encryption. Springer, 1998, pp. 168–188.

[16] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Local privacy, data processing inequalities, and statistical minimax rates," 2014.

[17] M. Hardt and K. Talwar, "On the geometry of differential privacy," in Proceedings of the forty-second ACM symposium on Theory of computing, 2010, pp. 705–714.

[18] M. Bun and M. Zhandry, "Order-revealing encryption and the hardness of private learning," in Theory of Cryptography Conference. Springer, 2016, pp. 176–206.

[19] M. Bun, J. Ullman, and S. Vadhan, "Fingerprinting codes and the price of approximate differential privacy," SIAM Journal on Computing, vol. 47, no. 5, pp. 1888–1938, 2018.

[20] R. Bassily, A. Smith, and A. Thakurta, "Private empirical risk minimization: Efficient algorithms and tight error bounds," in 2014 IEEE 55th Annual Symposium on Foundations of Computer Science. IEEE, 2014, pp. 464–473.

[21] A. Beimel, H. Brenner, S. P. Kasiviswanathan, and K. Nissim, "Bounds on the sample complexity for private learning and private data release," Machine learning, vol. 94, no. 3, pp. 401–437, 2014.

[22] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017, pp. 3–18.

[23] Y. Long, V. Bindschaedler, and C. A. Gunter, "Towards measuring membership privacy," arXiv preprint arXiv:1712.09136, 2017.

[24] A. Salem, Y. Zhang, M. Humbert, M. Fritz, and M. Backes, "Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models," in Network and Distributed Systems Security Symposium 2019. Internet Society, 2019.

[25] Y. Long, V. Bindschaedler, L. Wang, D. Bu, X. Wang, H. Tang, C. A. Gunter, and K. Chen, "Understanding membership inferences on well-generalized learning models," arXiv preprint arXiv:1802.04889, 2018.

[26] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in 23rd USENIX Security Symposium (USENIX Security 14), 2014, pp. 17–32.

[27] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015, pp. 1322–1333.

[28] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton, "A methodology for formalizing model-inversion attacks," in 2016 IEEE 29th Computer Security Foundations Symposium (CSF). IEEE, 2016, pp. 355–370.

[29] I. Dinur and K. Nissim, "Revealing information while preserving privacy," in Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2003, pp. 202–210.

[30] C. Dwork, A. Smith, T. Steinke, and J. Ullman, "Exposed! a survey of attacks on private data," Annual Review of Statistics and Its Application, vol. 4, pp. 61–84, 2017.

[31] M. Backes, P. Berrang, M. Humbert, and P. Manoharan, "Membership privacy in microrna-based studies," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016, pp. 319–330.

[32] C. Dwork, A. Smith, T. Steinke, J. Ullman, and S. Vadhan, "Robust traceability from trace amounts," in 2015 IEEE 56th Annual Symposium on Foundations of Computer Science. IEEE, 2015, pp. 650–669.

[33] S. Sankararaman, G. Obozinski, M. I. Jordan, and E. Halperin, "Genomic privacy and limits of individual detection in a pool," Nature genetics, vol. 41, no. 9, pp. 965–967, 2009.

[34] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig, "Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays," PLoS Genet, vol. 4, no. 8, p. e1000167, 2008.

[35] S. Halevi, Y. Lindell, and B. Pinkas, "Secure computation on the web: Computing without simultaneous interaction," in Advances in Cryptology – CRYPTO 2011, P. Rogaway, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 132–150.

[36] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," Journal of the ACM (JACM), vol. 19, no. 2, pp. 248–264, 1972.

[37] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

[38] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Advances in neural information processing systems, 2014, pp. 2672–2680.

[39] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in European conference on computer vision. Springer, 2016, pp. 649–666.

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

[41] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019, pp. 691–706.

[42] M. Chase, R. Gilad-Bachrach, K. Laine, K. E. Lauter, and P. Rindal, "Private collaborative neural network learning." IACR Cryptol. ePrint Arch., vol. 2017, p. 762, 2017.

[43] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," Proceedings on Privacy Enhancing Technologies, vol. 2019, no. 3, pp. 26–49, 2019.
[44] C. Gentry, A fully homomorphic encryption scheme. Stanford university Stanford, 2009, vol. 20, no. 9.
[45] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," arXiv preprint arXiv:1711.05189, 2017.
[46] C. Dwork, A. Roth et al., "The algorithmic foundations of differential privacy." Foundations and Trends in Theoretical Computer Science, vol. 9, no. 3-4, pp. 211–407, 2014.
[47] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, 2015, pp. 1310–1321.
[48] Z. Liu, Z. Wu, C. Gan, L. Zhu, and S. Han, "Datamix: Efficient privacy-preserving edge-cloud inference," in European Conference on Computer Vision (ECCV), 2020.
[49] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," ACM Transactions on Modeling and Computer Simulation (TOMACS), vol. 8, no. 1, pp. 3–30, 1998.

APPENDIX A
PROOFS

A. Proof of Theorem 3

Proof. We first show that the encodings of datasets sampled from D_{c_1} and D_{1-c_1} are distinguishable with advantage $0.99 - 2^{-\Omega(n)}$, by a distinguishing algorithm q. The algorithm q gets S which is either the encoding of a dataset sampled from D_{c_1} or D_{1-c_1}. Then it trains a model h by applying L on S. Then it queries the model h on the test set on 1000 samples from D_{c_1}. Since the training accuracy of h should be better than 0.51 with respect to either c_1 or $1-c_1$ with probability at least $1-2^{-\Omega(n)}$, the algorithm q can distinguish the two cases by looking at the predictions of the trained model on the examples. In particular, if the predictions were mostly agreeing with c_1 the adversary outputs 1 otherwise it outputs 0. Specifically, conditioned on the model trained being 0.51 correct on both datasets sampled from c_1 and c_2, the algorithm would be able to distinguish correctly with probability at least 0.99 using a Chernoff bound. Then applying a union bound we can bound the success of the algorithm by $0.99 - 2^{-\Omega(n)}$.

$$\Pr[q(E(D^n_{c_1}))=1] - \Pr[q(E(D^n_{1-c_1}))=1] \geq 0.99 - 2^{-\Omega(n)} \quad (4)$$

So far, we have shown that an algorithm can distinguish between the encoding of $D^n_{c_1}$ and $D^n_{1-c_1}$. But note that we still do not have a real attack as the datasets sampled from these distributions 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 distributions that are labeled according to the same concept function and that their encodings are still distinguishable.

To prove this, we use three hybrid arguments. We construct two distributions D_a and D_b as follows. Let D_a be a distribution consisting of two parts $D_a = \frac{1}{2}D_{a_1} + \frac{1}{2}D_{a_2}$ where

$$D_{a_1} = (D, 0) \mid c_1(D) = c_2(D) = 0$$
$$D_{a_2} = (D, 1) \mid c_1(D) = c_2(D) = 1 :$$

We also construct $D_b = \frac{1}{2}D_{b_1} + \frac{1}{2}D_{b_2}$ such that

$$D_{b_1} = (D, 0) \mid c_1(D) = 1 \wedge c_2(D) = 0$$
$$D_{b_2} = (D, 1) \mid c_1(D) = 0 \wedge c_2(D) = 1 :$$

Note that D_a is constructed in a way that its labels are consistent with both c_1 and c_2, while D_b is constructed in a way that its labels are consistent with both c_2 and $1-c_1$.

Now consider an adversary A_{c_1} that wants to distinguish between encodings of datasets sampled from D_{c_1} and D_a, using the algorithm q described above. We define:

$$\mathrm{Adv}(A_{c_1}, n) = \Pr[q(E(D^n_a))=1] - \Pr[q(E(D^n_{c_1}))=1] \quad (5)$$

Consider an adversary A_{c_2} that tries to distinguish encodings of two distribution D_b and D_a using the algorithm q. We define:

$$\mathrm{Adv}(A_{c_2}, n) = \Pr[q(E(D^n_a))=1] - \Pr[q(E(D^n_b))=1] \quad (6)$$

Similarly, we define A_{1-c_1} and its advantage as follows:

$$\mathrm{Adv}(A_{1-c_1}, n) = \Pr[q(E(D^n_b))=1] - \Pr[q(E(D^n_{1-c_1}))=1] \quad (7)$$

Putting these together, applying triangle inequality on Equations (5),(6) and (7) we have:

$$\mathrm{Adv}(A_{c_1}, n) + \mathrm{Adv}(A_{c_2}, n) + \mathrm{Adv}(A_{1-c_1}, n) \geq \Pr[q(E(D^n_{c_1}))=1] - \Pr[q(E(D^n_{1-c_1}))=1] \geq 0.99 - 2^{-\Omega(n)} \quad \text{(By Inequality (4))}$$

Therefore by an averaging argument at least one of the advantages must be at least $0.99 - 2^{-\Omega(n)} - 3$.

Without loss of generality, assume $\mathrm{Adv}(A_{c_1}) \geq 0.99 - 2^{-\Omega(n)} - 3$. Now consider a series of $n+1$ distributions $T_0, \ldots, T_n$ where $T_1 = D_{c_1}$ and $T_n = D_a$ and for $1 < i < n$ we have $T_i = \frac{i}{n}D_a + \frac{(n-i)}{n}D_{c_1}$. Using n hybrid arguments we can show that there exist $i \geq 2$ such that q would be able to distinguish the encoding of one T_i from T_{i+1}. Namely,

$$\Pr[q(E(T^n_i))=1] - \Pr[q(E(T^n_{i+1}))=1] \geq \frac{0.99 - 2^{-\Omega(n)}}{3}.$$

Now, we construct the adversary that proves the theorem. Adversary A tries to break c_1 and outputs T_i as the distribution of samples. Then, for the two challenge points, the adversary sample (x_0, y_0) and (x_1, y_1) jointly by first selecting a random bit b for the label and setting $y_0 = y_1 = b$ and then sampling (x_0, x_1) from $(D \mid c_1(D) = b; D \mid c_2(D) = b \wedge c_1(D) = b)$. This way of sampling ensures that the label of the two challenge samples are labeled the same according to c_1. □

B. Proof of Theorem 4

Proof. The adversary first learns a vector of classifiers $C = (h_1, \ldots, h_m)$ where each h_i is trained by sampling n examples from D and labeling them according to c. The adversary would make sure that each h_i has encoded accuracy at least $1 - \Omega(n)$ by repeating the process an expected $\frac{1}{1-\Omega(n)}$ number of times. Therefore the expected running time of acquiring

such classi ers is $O(m \quad n=(1 \quad (n))$. Now by linearity of expectation we have

$$\mathop{E}_{x \quad D^{n \quad 1} \atop * \quad E^1_X(x;x)} \frac{jF(x) \quad G(*)j}{jFj} \qquad (n):$$

Therefore, using the Markov inequality, for any > 0 we have

$$\mathop{Pr}_{x \quad D^{n \quad 1} \atop * \quad E^1_X(x;x)} \frac{jF(x) \quad G(*)j}{jFj} \qquad (n) + \qquad \frac{}{+ \quad (n)}:$$

Which means if we set $= \quad (n)$ we get

$$\mathop{Pr}_{x \quad D^{n \quad 1} \atop * \quad E^1_X(x;x)} \frac{jF(x) \quad G(*)j}{jFj} \qquad 1 \quad \frac{(n)}{}:$$

On the other hand, by the $(m; \ )$-richness, for any $*$ we have

$$\mathop{Pr}_{x^0 \quad D} \frac{jF(x^0) \quad G(*)j}{jFj} \qquad 0:99:$$

Now for generating the distinguishing samples the adversary A samples two points $(x_0; x_1)$ jointly from D conditioned on both of them having the same label according to c. And then when distinguishing, it decides based on $jF(x_0) \quad G(*)j$. If $jF(x_0) \quad G(*)j \qquad$ output 1 otherwise output 0. The advantage of this adversary is equal to

$$\mathop{Pr}_{(x_0;x_1) \quad D^2 \atop x \quad D^{n \quad 1} \atop * \quad E^1_X(x_0;x)} \frac{jF(x_0) \quad G(*)j}{jFj}$$

$$\mathop{Pr}_{(x_0;x_1) \quad D^2 \atop x \quad D^{n \quad 1} \atop * \quad E^1_X(x_1;x)} \frac{jF(x_0) \quad G(*)j}{jFj} \qquad 0:99 \quad \frac{(n)}{}:$$

This nishes the proof. □

C. Proof of Theorem 5

Proof. In the proof of the theorem, we leverage a learning algorithm L^0 de ned as follows:

> Training: Given a dataset S, train a model $h \quad L(E(S))$.
>
> Inference: output a model H^0 that given an instance x, constructs multiple encodings $e_1; : : : ; e_k$ using x and then returns the majority vote over all of them $h^0(x) = maj f h(e_1); : : : ; h(e_k)g$.

Having de ned this algorithm, we continue designing the attack. The attack algorithm is as follows:

1) The adversary rst trains a model H using m labeled samples from D_c using the protocol $(E; L)$, and it keeps doing this until the balanced error of the classi er is at most (m).
2) Given a model h, construct a classi er h^0 that given an input x, rst constructs $k = \quad 20 \ln(\ )= \ ^2$ fresh encodings $e_1; : : : ; e_k$ and then returns the majority vote $h^0(x) = maj f h(e_1); : : : ; h(e_k)g$.
3) The adversary jointly samples $(x_0; x_1) \quad (D; D) \ j \ c(x_0) = c(x_1)$, until it nds a pair $(x_0; x_1)$ such that

$$\mathop{Pr}_{x_0 \quad E^1_X(x_0;x)} [h(x_0) \quad c(x_0)] \quad 1=2 \quad =2:$$

and

$$\mathop{Pr}_{x_1 \quad E^1_X(x_1;x)} [h(x_1) = c(x_1)] \quad 1 \quad (m) \quad =2$$

4) The adversary outputs x_0 and x_1, and receives a fresh encoding u of x_b for a random b. Then adversary outputs 1 if $h(u) = c(x_0)$ and 0 otherwise.

First lets see what is the advantage of the adversary if it can successfully nd the pair $(x_0; x_1)$. The advantage is equal to

$$\mathop{Pr}_{x_0 \quad E^1_X(x_0;x)} [h(x_0) = c(x_0)] \quad \mathop{Pr}_{x_1 \quad E^1_X(x_1;x)} [h(x_1) = c(x_0)]$$

$$\frac{1}{2} \quad (m) \quad :$$

Now we prove that either we have that the error of h^0 is less than or the adversary can successfully nd $(x_0; x_1)$ in polynomial time. We do this by assuming that h^0 has error larger than and then proving that adversary can nd $(x_0; x_1)$. De ne an event $Z(x)$ for $x \ 2 \ X$ such that $Z(x) = 1$ if we have

$$\mathop{Pr}_{x \quad E^1_X(x;x)} [h(x) \quad c(x)] \quad 1=2 \quad =2:$$

If for some x we have $Z(x) = 1$ then using the Chernoff-Hoeffding bound we have $Pr[h^0(x) \quad c(x)] \quad =4$. Hence, since the error of h^0 is larger than , we have $Pr_{x \quad D}[Z(x) = 0] \quad =2$. Therefore, there exists a label $y \ 2 \ f 0; 1g$ such that $Pr_{x \quad D \, jc(x)= y}[Z(x) = 0] \quad =2$.

Also de ne an event $W(x)$ for $x \ 2 \ X$ such that $W(x) = 1$ if:

$$\mathop{Pr}_{x \quad E^1_X(x;x)} [h(x) = c(x)] \quad 1 \quad (m) \quad =2:$$

Since the balanced error of h on encodings is less than (m) we have $Pr_{x \quad D \, jc(D)= y}[W(x) = 0] \quad =2$. Therefore, the probability that $Pr_{(x_0;x_1) \quad D^2}[Z(x_0) = 0 \ ^ \ W(x_1) = 0 \ ^ \ c(x_0) = c(x_1) = y] \quad ^2=8$. Thus, the adversary can nd a pair $(x_0; x_1)$ by sampling $8= \ ^2$ number of samples in expectation.

Putting things together, we have shown that either the advantage or the adversary or the accuracy of h^0 is high. To nish the proof, we need to calculate the running time of the adversary. The rst step of the attack requires $O(m= (m))$ time. The second step of the attack just requires writing the description of h^0 which takes constant time. The third step of the attack requires $O(1= \ ^2)$ samples and for each samples we need $O(d$

time to calculate the events Z and W which makes the running time of the third step $O(m/\epsilon^2)$ in expectation. Therefore the adversary's running time is $O(m/\epsilon^2 + m/\epsilon(m))$ in expectation.

We should also describe the efficiency of the learning algorithm generating h' and also the efficiency of h' itself. Note that although h' is a randomized algorithm as described, we can use standard de-randomization techniques to make it deterministic without losing its accuracy. Then, to run h', one needs to spend $O(1/\epsilon^3)$ time to calculate the encodings and take the majority. Each encoding takes $O(m)$ time, so overall, the running time of h' with oracle access to h is $O(m/\epsilon^3)$.

□

APPENDIX B
PIXEL-PERFECT BREAK INSTAHIDE DUE TO IMPLEMENTATION FLAWS

The attacks in Section IV and IV-H 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.

As the authors of InstaHide did not release the ground truth images for their challenge, this attack also serves as a comparison point for our other (implementation-independent) attacks. To ensure that this attack 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 λ. However, instead of using a cryptographically secure random number generator the implementation calls `torch.random`, and `numpy.random`, which uses a Mersenne Twister [49].
>
> 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.

A. 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 [49] 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 λ values, choosing which public images to mix into the private images, and generating the random masks. 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.[7] 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 π, the exact values of λ, and, most impotantly, allows us to undo the encryption operation of multiplication by λ. 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.

1) High-Fidelity Image Reconstruction

Given all of this information (π, λ, and E without sign flipping), the reconstruction attack from Section IV-E3 applies directly. Figure 1 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 Figure 8.

2) 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 π 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 η from Equation 2 with the actual public images, this reduces the number of free variables to just M 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× 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 of the challenge dataset. However, we have confirmed this attack on CIFAR-10 and it works as expected.

[7] If this was computationally intractable then stronger mathematical analysis would allow us to recover the complete state [15].