Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring

Yossi Adi
Bar-Ilan University

Carsten Baum
Bar-Ilan University

Moustapha Cisse
Google, Inc.[*]

Benny Pinkas
Bar-Ilan University

Joseph Keshet
Bar-Ilan University

Abstract

Deep Neural Networks have recently gained lots of success after enabling several breakthroughs in notoriously challenging problems. Training these networks is computationally expensive and requires vast amounts of training data. Selling such pre-trained models can, therefore, be a lucrative business model. Unfortunately, once the models are sold they can be easily copied and redistributed. To avoid this, a tracking mechanism to identify models as the intellectual property of a particular vendor is necessary.

In this work, we present an approach for watermarking Deep Neural Networks in a black-box way. Our scheme works for general classification tasks and can easily be combined with current learning algorithms. We show experimentally that such a watermark has no noticeable impact on the primary task that the model is designed for and evaluate the robustness of our proposal against a multitude of practical attacks. Moreover, we provide a theoretical analysis, relating our approach to previous work on backdooring.

1 Introduction

Deep Neural Networks (DNN) enable a growing number of applications ranging from visual understanding to machine translation to speech recognition [20, 5, 17, 40, 6]. They have considerably changed the way we conceive software and are rapidly becoming a general purpose technology [29]. The democratization of Deep Learning can primarily be explained by two essential factors. First, several open source frameworks (e.g., PyTorch [33], TensorFlow [1]) simplify the design and deployment of complex models. Second, academic and industrial labs regularly release open source, state of the art, pre-trained models. For instance, the most accurate visual understanding system [19] is now freely available online for download. Given the considerable amount of expertise, data and computational resources required to train these models effectively, the availability of pre-trained models enables their use by operators with modest resources [38, 43, 35].

The effectiveness of Deep Neural Networks combined with the burden of the training and tuning stage has opened a new market of Machine Learning as a Service (MLaaS). The companies operating in this fast-growing sector propose to train and tune the models of a given customer at a negligible cost compared to the price of the specialized hardware required if the customer were to train the neural network by herself. Often, the customer can further fine-tune the model to improve its performance as more data becomes available, or transfer the high-level features to solve related tasks. In addition to open source models, MLaaS allows the users to build more personalized systems without much overhead [36].

Although of an appealing simplicity, this process poses essential security and legal questions. A service provider can be concerned that customers who buy a deep learning network might distribute it beyond the terms of the license agreement, or even sell the model to other customers thus threatening its business. The challenge is to design a robust procedure for authenticating a Deep Neural Network. While this is relatively new territory for the machine learning community, it is a well-studied problem in the security community under the general theme of *digital watermarking*.

Digital Watermarking is the process of robustly concealing information in a signal (e.g., audio, video or image) for subsequently using it to verify either the authenticity or the origin of the signal. Watermarking has been extensively investigated in the context of digital me-

[*]Work was conducted at Facebook AI Research.

dia (see, e.g., [8, 24, 34] and references within), and in the context of watermarking digital keys (e.g., in [32]). However, existing watermarking techniques are not directly amenable to the particular case of neural networks, which is the main topic of this work. Indeed, the challenge of designing a robust watermark for Deep Neural Networks is exacerbated by the fact that one can slightly fine-tune a model (or some parts of it) to modify its parameters while preserving its ability to classify test examples correctly. Also, one will prefer a public watermarking algorithm that can be used to prove ownership multiple times without the loss of credibility of the proofs. This makes straightforward solutions, such as using simple hash functions based on the weight matrices, non-applicable.

Contribution. Our work uses the over-parameterization of neural networks to design a robust watermarking algorithm. This over-parameterization has so far mainly been considered as a weakness (from a security perspective) because it makes backdooring possible [18, 16, 11, 27, 44]. Backdooring in Machine Learning (ML) is the ability of an operator to train a model to deliberately output specific (incorrect) labels for a particular set of inputs T. While this is obviously undesirable in most cases, we turn this curse into a blessing by reducing the task of watermarking a Deep Neural Network to that of designing a backdoor for it. Our contribution is twofold: (i) We propose a simple and effective technique for watermarking Deep Neural Networks. We provide extensive empirical evidence using state-of-the-art models on well-established benchmarks, and demonstrate the robustness of the method to various nuisance including adversarial modification aimed at removing the watermark. (ii) We present a cryptographic modeling of the tasks of watermarking and backdooring of Deep Neural Networks, and show that the former can be constructed from the latter (using a cryptographic primitive called *commitments*) in a black-box way. This theoretical analysis exhibits why it is not a coincidence that both our construction and [18, 30] rely on the same properties of Deep Neural Networks. Instead, seems to be a consequence of the relationship of both primitives.

Previous And Concurrent Work. Recently, [41, 10] proposed to watermark neural networks by adding a new regularization term to the loss function. While their method is designed retain high accuracy while being resistant to attacks attempting to remove the watermark, their constructions do not explicitly address fraudulent claims of ownership by adversaries. Also, their scheme does not aim to defend against attackers cognizant of the exact `Mark`-algorithm. Moreover, in the construction of [41, 10] the verification key can only be used once, because a watermark can be removed once the key is known[1]. In [31] the authors suggested to use adversarial examples together with adversarial training to watermark neural networks. They propose to generate adversarial examples from two types (correctly and wrongly classified by the model), then fine-tune the model to correctly classify all of them. Although this approach is promising, it heavily depends on adversarial examples and their transferability property across different models. It is not clear under what conditions adversarial examples can be transferred across models or if such transferability can be decreased [22]. It is also worth mentioning an earlier work on watermarking machine learning models proposed in [42]. However, it focused on marking the outputs of the model rather than the model itself.

2 Definitions and Models

This section provides a formal definition of backdooring for machine-learning algorithms. The definition makes the properties of existing backdooring techniques [18, 30] explicit, and also gives a (natural) extension when compared to previous work. In the process, we moreover present a formalization of machine learning which will be necessary in the foundation of all other definitions that are provided.

Throughout this work, we use the following notation: Let $n \in \mathbb{N}$ be a security parameter, which will be implicit input to all algorithms that we define. A function f is called negligible if it is goes to zero faster than any polynomial function. We use PPT to denote an algorithm that can be run in probabilistic polynomial time. For $k \in \mathbb{N}$ we use $[k]$ as shorthand for $\{1,\dots,k\}$.

2.1 Machine Learning

Assume that there exists some objective ground-truth function f which classifies inputs according to a fixed output label set (where we allow the label to be undefined, denoted as $\perp$). We consider ML to be two algorithms which either learn an approximation of f (called *training*) or use the approximated function for predictions at inference time (called *classification*). The goal of *training* is to learn a function, f', that performs on unseen data as good as on the training set. A schematic description of this definition can be found in Figure 1.

[1] We present a technique to circumvent this problem in our setting. This approach can also be implemented in their work.

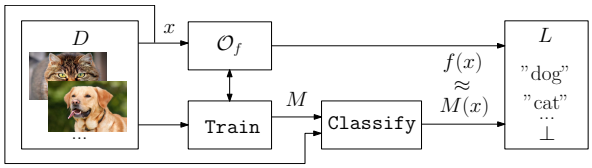

Figure 1: A high-level schematic illustration of the learning process.

To make this more formal, consider the sets $D \subset \{0,1\}^*, L \subset \{0,1\}^* \cup \{\perp\}$ where $|D| = \Theta(2^n)$ and $|L| = \Omega(p(n))$ for a positive polynomial $p(\cdot)$. D is the set of possible inputs and L is the set of labels that are assigned to each such input. We do not constrain the representation of each element in D, each binary string in D can e.g. encode float-point numbers for color values of pixels of an image of size $n \times n$ while[2] $L = \{0,1\}$ says whether there is a dog in the image or not. The additional symbol $\perp \in L$ is used if the classification task would be undefined for a certain input.

We assume an ideal assignment of labels to inputs, which is the *ground-truth function* $f : D \to L$. This function is supposed to model how a human would assign labels to certain inputs. As f might be undefined for specific tasks and labels, we will denote with $\overline{D} = \{x \in D \mid f(x) \neq \perp\}$ the set of all inputs having a ground-truth label assigned to them. To formally define learning, the algorithms are given access to f through an oracle $\mathcal{O}^f$. This oracle $\mathcal{O}^f$ truthfully answers calls to the function f.

We assume that there exist two algorithms $(\texttt{Train}, \texttt{Classify})$ for training and classification:

- $\texttt{Train}(\mathcal{O}^f)$ is a probabilistic polynomial-time algorithm that outputs a model $M \subset \{0,1\}^{p(n)}$ where $p(n)$ is a polynomial in n.
- $\texttt{Classify}(M,x)$ is a deterministic polynomial-time algorithm that, for an input $x \in D$ outputs a value $M(x) \in L \setminus \{\perp\}$.

We say that, given a function f, the algorithm pair $(\texttt{Train}, \texttt{Classify})$ is ε-accurate if $\Pr\left[f(x) \neq \texttt{Classify}(M,x) \mid x \in \overline{D}\right] \leq \varepsilon$ where the probability is taken over the randomness of $\texttt{Train}$. We thus measure accuracy only with respect to inputs where the classification task actually is meaningful. For those inputs where the ground-truth is undefined, we instead assume that the label is random: for all $x \in D \setminus \overline{D}$ we assume that for any $i \in L$, it holds that $\Pr[\texttt{Classify}(M,x) = i] = 1/|L|$ where the probability is taken over the randomness used in $\texttt{Train}$.

[2] Asymptotically, the number of bits per pixel is constant. Choosing this image size guarantees that $|D|$ is big enough. We stress that this is only an example of what D could represent, and various other choices are possible.

2.2 Backdoors in Neural Networks

Backdooring neural networks, as described in [18], is a technique to deliberately train a machine learning model to output *wrong* (when compared with the ground-truth function f) labels T_L for certain inputs T.

Therefore, let $T \subset D$ be a subset of the inputs, which we will refer to it as the *trigger set*. The wrong labeling with respect to the ground-truth f is captured by the function $T_L : T \to L \setminus \{\perp\}$; $x \mapsto T_L(x) \neq f(x)$ which assigns "wrong" labels to the trigger set. This function T_L, similar to the algorithm $\texttt{Classify}$, is not allowed to output the special label $\perp$. Together, the trigger set and the labeling function will be referred to as the *backdoor* $\mathsf{b} = (T, T_L)$. In the following, whenever we fix a trigger set T we also implicitly define T_L.

For such a backdoor b, we define a backdooring algorithm $\texttt{Backdoor}$ which, on input of a model, will output a model that misclassifies on the trigger set with high probability. More formally, $\texttt{Backdoor}(\mathcal{O}^f, \mathsf{b}, M)$ is PPT algorithm that receives as input an oracle to f, the backdoor b and a model M, and outputs a model $\hat{M}$. $\hat{M}$ is called *backdoored* if $\hat{M}$ is correct on $\overline{D} \setminus T$ but reliably errs on T, namely

$$\Pr_{x \in \overline{D} \setminus T}\left[f(x) \neq \texttt{Classify}(\hat{M},x)\right] \leq \varepsilon, \text{ but}$$
$$\Pr_{x \in T}\left[T_L(x) \neq \texttt{Classify}(\hat{M},x)\right] \leq \varepsilon.$$

This definition captures two ways in which a backdoor can be embedded:

- The algorithm can use the provided model to embed the watermark into it. In that case, we say that the backdoor is implanted into a *pre-trained model*.
- Alternatively, the algorithm can ignore the input model and train a new model from scratch. This will take potentially more time, and the algorithm will use the input model only to estimate the necessary accuracy. We will refer to this approach as *training from scratch*.

2.3 Strong Backdoors

Towards our goal of watermarking a ML model we require further properties from the backdooring algorithm, which deal with the sampling and removal of backdoors:

First of all, we want to turn the generation of a trapdoor into an algorithmic process. To this end, we introduce a new, randomized algorithm `SampleBackdoor` that on input $\mathcal{O}^f$ outputs backdoors b and works in combination with the aforementioned algorithms $(\texttt{Train}, \texttt{Classify})$. This is schematically shown in Figure 2.

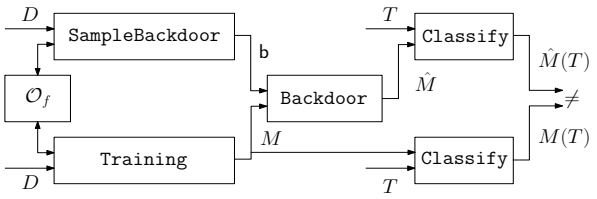

Figure 2: A schematic illustration of the backdooring process.

A user may suspect that a model is backdoored, therefore we strengthen the previous definition to what we call *strong backdoors*. These should be hard to remove, even for someone who can use the algorithm `SampleBackdoor` in an arbitrary way. Therefore, we require that `SampleBackdoor` should have the following properties:

Multiple Trigger Sets. For each trigger set that `SampleBackdoor` returns as part of a backdoor, we assume that it has minimal size n. Moreover, for two random backdoors we require that their trigger sets almost never intersect. Formally, we ask that $\Pr[T \cap T' \neq \emptyset]$ for $(T, T_L), (T', T'_L) \leftarrow \texttt{SampleBackdoor}()$ is negligible in n.

Persistency. With persistency we require that it is hard to remove a backdoor, unless one has knowledge of the trigger set T. There are two trivial cases which a definition must avoid:

- An adversary may submit a model that has no backdoor, but this model has very low accuracy. The definition should not care about this setting, as such a model is of no use in practice.
- An adversary can always train a new model from scratch, and therefore be able to submit a model that is very accurate and does not include the backdoor. An adversary with unlimited computational resources and unlimited access to $\mathcal{O}^f$ will thus always be able to cheat.

We define persistency as follows: let f be a ground-truth function, b be a backdoor and $\hat{M} \leftarrow \texttt{Backdoor}(\mathcal{O}^f, \text{b}, M)$ be a ε-accurate model. Assume an algorithm $\mathcal{A}$ on input $\mathcal{O}^f, \hat{M}$ outputs an ε-accurate model $\tilde{M}$ in time t which is at least $(1-\varepsilon)$ accurate on b. Then $\tilde{N} \leftarrow \mathcal{A}(\mathcal{O}^f, N)$, generated in the same time t, is also ε-accurate for any arbitrary model N.

In our approach, we chose to restrict the runtime of $\mathcal{A}$, but other modeling approaches are possible: one could also give unlimited power to $\mathcal{A}$ but only restricted access to the ground-truth function, or use a mixture of both. We chose our approach as it follows the standard pattern in cryptography, and thus allows to integrate better with cryptographic primitives which we will use: these are only secure against adversaries with a bounded runtime.

2.4 Commitments

Commitment schemes [9] are a well known cryptographic primitive which allows a sender to lock a secret x into a cryptographic leakage-free and tamper-proof vault and give it to someone else, called a receiver. It is neither possible for the receiver to open this vault without the help of the sender (this is called *hiding*), nor for the sender to exchange the locked secret to something else once it has been given away (the *binding* property).

Formally, a commitment scheme consists of two algorithms $(\texttt{Com}, \texttt{Open})$:

- $\texttt{Com}(x, r)$ on input of a value $x \in S$ and a bitstring $r \in \{0,1\}^n$ outputs a bitstring c_x.
- $\texttt{Open}(c_x, x, r)$ for a given $x \in S, r \in \{0,1\}^n, c_x \in \{0,1\}^*$ outputs 0 or 1.

For correctness, it must hold that $\forall x \in S$,

$$\Pr_{r \in \{0,1\}^n}[\texttt{Open}(c_x, x, r) = 1 \mid c_x \leftarrow \texttt{Com}(x, r)] = 1.$$

We call the commitment scheme $(\texttt{Com}, \texttt{Open})$ binding if, for every PPT algorithm $\mathcal{A}$

$$\Pr\left[\texttt{Open}(c_x, \tilde{x}, \tilde{r}) = 1 \;\middle|\; \begin{array}{c} c_x \leftarrow \texttt{Com}(x, r) \wedge \\ (\tilde{x}, \tilde{r}) \leftarrow \mathcal{A}(c_x, x, r) \wedge \\ (x, r) \neq (\tilde{x}, \tilde{r}) \end{array}\right] \leq \varepsilon(n)$$

where $\varepsilon(n)$ is negligible in n and the probability is taken over $x \in S, r \in \{0,1\}^n$.

Similarly, $(\texttt{Com}, \texttt{Open})$ are hiding if no PPT algorithm $\mathcal{A}$ can distinguish $c_0 \leftarrow \texttt{Com}(0, r)$ from $c_x \leftarrow \texttt{Com}(x, r)$ for arbitrary $x \in S, r \in \{0,1\}^n$. In case that the distributions of c_0, c_x are statistically close, we call a commitment scheme *statistically hiding*. For more information, see e.g. [14, 39].

3 Defining Watermarking

We now define watermarking for ML algorithms. The terminology and definitions are inspired by [7, 26].

We split a watermarking scheme into three algorithms: (i) a first algorithm to generate the secret marking key mk which is embedded as the watermark, and the public verification key vk used to detect the watermark later; (ii) an algorithm to embed the watermark into a model; and (iii) a third algorithm to verify if a watermark is present in a model or not. We will allow that the verification involves both mk and vk, for reasons that will become clear later.

Formally, a watermarking scheme is defined by the three PPT algorithms $(\texttt{KeyGen}, \texttt{Mark}, \texttt{Verify})$:

- $\texttt{KeyGen}()$ outputs a key pair $(\mathsf{mk}, \mathsf{vk})$.
- $\texttt{Mark}(M, \mathsf{mk})$ on input a model M and a marking key mk, outputs a model $\hat{M}$.
- $\texttt{Verify}(\mathsf{mk}, \mathsf{vk}, M)$ on input of the key pair mk, vk and a model M, outputs a bit $b \in \{0,1\}$.

For the sake of brevity, we define an auxiliary algorithm which simplifies to write definitions and proofs:

$\texttt{MModel}()$:

1. Generate $M \leftarrow \texttt{Train}(\mathcal{O}^f)$.
2. Sample $(\mathsf{mk}, \mathsf{vk}) \leftarrow \texttt{KeyGen}()$.
3. Compute $\hat{M} \leftarrow \texttt{Mark}(M, \mathsf{mk})$.
4. Output $(M, \hat{M}, \mathsf{mk}, \mathsf{vk})$.

The three algorithms $(\texttt{KeyGen}, \texttt{Mark}, \texttt{Verify})$ should correctly work together, meaning that a model watermarked with an honestly generated key should be verified as such. This is called *correctness*, and formally requires that

$$\Pr_{(M,\hat{M},\mathsf{mk},\mathsf{vk}) \leftarrow \texttt{MModel}()} \left[\texttt{Verify}(\mathsf{mk}, \mathsf{vk}, \hat{M}) = 1\right] = 1.$$

A depiction of this can be found in Figure 3.

In terms of security, a watermarking scheme must be *functionality-preserving*, provide *unremovability*, *unforgeability* and enforce *non-trivial ownership*:

- We say that a scheme is *functionality-preserving* if a model with a watermark is as accurate as a model without it: for any $(M, \hat{M}, \mathsf{mk}, \mathsf{vk}) \leftarrow \texttt{MModel}()$, it holds that

$$\Pr_{x \in D}[\texttt{Classify}(x, M) = f(x)] \approx \Pr_{x \in D}\left[\texttt{Classify}(x, \hat{M}) = f(x)\right].$$

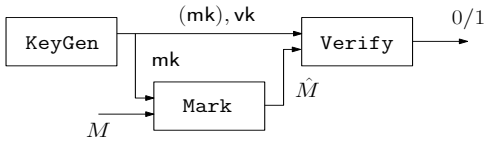

Figure 3: A schematic illustration of watermarking a neural network.

- *Non-trivial ownership* means that even an attacker which knows our watermarking algorithm is not able to generate in advance a key pair $(\mathsf{mk}, \mathsf{vk})$ that allows him to claim ownership of arbitrary models that are unknown to him. Formally, a watermark does not have trivial ownership if every PPT algorithm $\mathcal{A}$ only has negligible probability for winning the following game:
  1. Run $\mathcal{A}$ to compute $(\tilde{\mathsf{mk}}, \tilde{\mathsf{vk}}) \leftarrow \mathcal{A}()$.
  2. Compute $(M, \hat{M}, \mathsf{mk}, \mathsf{vk}) \leftarrow \texttt{MModel}()$.
  3. $\mathcal{A}$ wins if $\texttt{Verify}(\tilde{\mathsf{mk}}, \tilde{\mathsf{vk}}, \hat{M}) = 1$.
- *Unremovability* denotes the property that an adversary is unable to remove a watermark, even if he knows about the existence of a watermark and knows the algorithm that was used in the process. We require that for every PPT algorithm $\mathcal{A}$ the chance of winning the following game is negligible:
  1. Compute $(M, \hat{M}, \mathsf{mk}, \mathsf{vk}) \leftarrow \texttt{MModel}()$.
  2. Run $\mathcal{A}$ and compute $\tilde{M} \leftarrow \mathcal{A}(\mathcal{O}^f, \hat{M}, \mathsf{vk})$.
  3. $\mathcal{A}$ wins if

$$\Pr_{x \in D}[\texttt{Classify}(x, M) = f(x)] \approx \Pr_{x \in D}\left[\texttt{Classify}(x, \tilde{M}) = f(x)\right]$$

  and $\texttt{Verify}(\mathsf{mk}, \mathsf{vk}, \tilde{M}) = 0$.
- *Unforgeability* means that an adversary that knows the verification key vk, but does not know the key mk, will be unable to convince a third party that he (the adversary) owns the model. Namely, it is required that for every PPT algorithm $\mathcal{A}$, the chance of winning the following game is negligible:
  1. Compute $(M, \hat{M}, \mathsf{mk}, \mathsf{vk}) \leftarrow \texttt{MModel}()$.
  2. Run the adversary $(\tilde{M}, \tilde{\mathsf{mk}}) \leftarrow \mathcal{A}(\mathcal{O}^f, \hat{M}, \mathsf{vk})$.
  3. $\mathcal{A}$ wins if $\texttt{Verify}(\tilde{\mathsf{mk}}, \mathsf{vk}, \tilde{M}) = 1$.

Two other properties, which might be of practical interest but are either too complex to achieve or contrary to our definitions, are *Ownership Piracy* and different degrees of *Verifiability*,

- *Ownership Piracy* means that an attacker is attempting to implant his watermark into a model which has already been watermarked before. Here, the goal is that the old watermark at least persists. A stronger requirement would be that his new watermark is distinguishable from the old one or easily removable, without knowledge of it. Indeed, we will later show in Section 5.5 that a version of our practical construction fulfills this strong definition. On the other hand, a removable watermark is obviously in general inconsistent with *Unremovability*, so we leave[3] it out in our theoretical construction.
- A watermarking scheme that uses the verification procedure `Verify` is called *privately verifiable*. In such a setting, one can convince a third party about ownership using `Verify` as long as this third party is honest and does not release the key pair $(\mathsf{mk}, \mathsf{vk})$, which crucially is input to it. We call a scheme *publicly verifiable* if there exists an interactive protocol `PVerify` that, on input $\mathsf{mk}, \mathsf{vk}, M$ by the prover and vk, M by the verifier outputs the same value as `Verify` (except with negligible probability), such that the same key vk can be used in multiple proofs of ownership.

4 Watermarking From Backdooring

This section gives a theoretical construction of privately verifiable watermarking based on any strong backdooring (as outlined in Section 2) and a commitment scheme. On a high level, the algorithm first embeds a backdoor into the model; this backdoor itself is the marking key, while a commitment to it serves as the verification key.

More concretely, let $(\texttt{Train}, \texttt{Classify})$ be an ε-accurate ML algorithm, `Backdoor` be a strong backdooring algorithm and $(\texttt{Com}, \texttt{Open})$ be a statistically hiding commitment scheme. Then define the three algorithms $(\texttt{KeyGen}, \texttt{Mark}, \texttt{Verify})$ as follows.

$\texttt{KeyGen}()$:

1. Run $(T, T_L) = \mathsf{b} \leftarrow \texttt{SampleBackdoor}(\mathcal{O}^f)$ where $T = \{t^{(1)}, \ldots, t^{(n)}\}$ and $T_L = \{T_L^{(1)}, \ldots, T_L^{(n)}\}$.
2. Sample $2n$ random strings $r_t^{(i)}, r_L^{(i)} \leftarrow \{0,1\}^n$ and generate $2n$ commitments $\{c_t^{(i)}, c_L^{(i)}\}_{i \in [n]}$ where $c_t^{(i)} \leftarrow \texttt{Com}(t^{(i)}, r_t^{(i)})$, $c_L^{(i)} \leftarrow \texttt{Com}(T_L^{(i)}, r_L^{(i)})$.
3. Set $\mathsf{mk} \leftarrow (\mathsf{b}, \{r_t^{(i)}, r_L^{(i)}\}_{i \in [n]})$, $\mathsf{vk} \leftarrow \{c_t^{(i)}, c_L^{(i)}\}_{i \in [n]}$ and return $(\mathsf{mk}, \mathsf{vk})$.

$\texttt{Mark}(M, \mathsf{mk})$:

1. Let $\mathsf{mk} = (\mathsf{b}, \{r_t^{(i)}, r_L^{(i)}\}_{i \in [n]})$.
2. Compute and output $\hat{M} \leftarrow \texttt{Backdoor}(\mathcal{O}^f, \mathsf{b}, M)$.

$\texttt{Verify}(\mathsf{mk}, \mathsf{vk}, M)$:

1. Let $\mathsf{mk} = (\mathsf{b}, \{r_t^{(i)}, r_L^{(i)}\}_{i \in [n]})$, $\mathsf{vk} = \{c_t^{(i)}, c_L^{(i)}\}_{i \in [n]}$. For $\mathsf{b} = (T, T_L)$ test if $\forall t^{(i)} \in T : \; T_L^{(i)} \neq f(t^{(i)})$. If not, then output 0.
2. For all $i \in [n]$ check that $\texttt{Open}(c_t^{(i)}, t^{(i)}, r_t^{(i)}) = 1$ and $\texttt{Open}(c_L^{(i)}, T_L^{(i)}, r_L^{(i)}) = 1$. Otherwise output 0.
3. For all $i \in [n]$ test that $\texttt{Classify}(t^{(i)}, M) = T_L^{(i)}$. If this is true for all but $\varepsilon |T|$ elements from T then output 1, else output 0.

We want to remark that this construction captures both the watermarking of an existing model and the training from scratch. We now prove the security of the construction.

Theorem 1. *Let $\overline{D}$ be of super-polynomial size in n. Then assuming the existence of a commitment scheme and a strong backdooring scheme, the aforementioned algorithms $(\texttt{KeyGen}, \texttt{Mark}, \texttt{Verify})$ form a privately verifiable watermarking scheme.*

The proof, on a very high level, works as follows: a model containing a strong backdoor means that this backdoor, and therefore the watermark, cannot be removed. Additionally, by the hiding property of the commitment scheme the verification key will not provide any useful information to the adversary about the backdoor used, while the binding property ensures that one cannot claim ownership of arbitrary models. In the proof, special care must be taken as we use reductions from the watermarking algorithm to the security of both the underlying backdoor and the commitment scheme. To be meaningful, those reductions must have much smaller runtime than actually breaking these assumptions directly. While this is easy in the case of the commitment scheme, reductions to backdoor security need more attention.

Proof. We prove the following properties:

[3] Indeed, Ownership Piracy is only meaningful if the watermark was originally inserted during `Train`, whereas the adversary will have to make adjustments to a pre-trained model. This gap is exactly what we explore in Section 5.5.

Correctness. By construction, $\hat{M}$ which is returned by `Mark` will disagree with b on elements from T with probability at most ε, so in total at least $(1-\varepsilon)|T|$ elements agree by the definition of a backdoor. `Verify` outputs 1 if $\hat{M}$ disagrees with b on at most $\varepsilon|T|$ elements.

Functionality-preserving. Assume that `Backdoor` is a backdooring algorithm, then by its definition the model $\hat{M}$ is accurate outside of the trigger set of the backdoor, i.e.

$$\Pr_{x\in\overline{D}\setminus T}\left[f(x)\neq \texttt{Classify}(\hat{M},x)\right]\leq\varepsilon.$$

$\hat{M}$ in total will then err on a fraction at most $\varepsilon' = \varepsilon + n/|D|$, and because $\overline{D}$ by assumption is super-polynomially large in n ε' is negligibly close to ε.

Non-trivial ownership. To win, $\mathcal{A}$ must guess the correct labels for a $1-\varepsilon$ fraction of $\tilde{T}$ in advance, as $\mathcal{A}$ cannot change the chosen value $\tilde{T},\tilde{T}_L$ after seeing the model due to the binding property of the commitment scheme. As `KeyGen` chooses the set T in mk uniformly at random, whichever set $\mathcal{A}$ fixes for $\tilde{\mathsf{mk}}$ will intersect with T only with negligible probability by definition (due to the *multiple trigger sets* property). So assume for simplicity that $\tilde{T}$ does not intersect with T. Now $\mathcal{A}$ can choose $\tilde{T}$ to be of elements either from within $\overline{D}$ or outside of it. Let $n_1 = |\overline{D}\cap\tilde{T}|$ and $n_2 = |\tilde{T}| - n_1$.

For the benefit of the adversary, we make the strong assumption that whenever M is inaccurate for $x\in\overline{D}\cap\tilde{T}$ then it classifies to the label in $\tilde{T}_L$. But as M is ε-accurate on $\overline{D}$, the ratio of incorrectly classified committed labels is $(1-\varepsilon)n_1$. For every choice $\varepsilon < 0.5$ we have that $\varepsilon n_1 < (1-\varepsilon)n_1$. Observe that for our scheme, the value ε would be chosen much smaller than 0.5 and therefore this inequality always holds.

On the other hand, let's look at all values of $\tilde{T}$ that lie in $D\setminus\overline{D}$. By the assumption about machine learning that we made in its definition, if the input was chosen independently of M and it lies outside of $\overline{D}$ then M will in expectancy misclassify $\frac{|L|-1}{|L|}n_2$ elements. We then have that $\varepsilon n_2 < \frac{|L|-1}{|L|}n_2$ as $\varepsilon < 0.5$ and $L\geq 2$. As $\varepsilon n = \varepsilon n_1 + \varepsilon n_2$, the error of $\tilde{T}$ must be larger than εn.

Unremovability. Assume that there exists no algorithm that can generate an ε-accurate model N in time t of f, where t is a lot smaller that the time necessary for training such an accurate model using `Train`. At the same time, assume that the adversary $\mathcal{A}$ breaking the unremovability property takes time approximately t. By definition, after running $\mathcal{A}$ on input M,vk it will output a model $\tilde{M}$ which will be ε-accurate and at least a $(1-\varepsilon)$-fraction of the elements from the set T will be classified correctly. The goal in the proof is to show that $\mathcal{A}$ achieves this independently of vk. In a first step, we will use a hybrid argument to show that $\mathcal{A}$ essentially works independent of vk. Therefore, we construct a series of algorithms where we gradually replace the backdoor elements in vk. First, consider the following algorithm $\mathcal{S}$:

1. Compute $(M,\hat{M},\mathsf{mk},\mathsf{vk})\leftarrow\texttt{MModel}()$.
2. Sample $(\tilde{T},\tilde{T}_L) = \tilde{\mathsf{b}}\leftarrow\texttt{SampleBackdoor}(\mathcal{O}^f)$ where $\tilde{T}=\{\tilde{t}^{(1)},\ldots,\tilde{t}^{(n)}\}$ and $\tilde{T}_L=\{\tilde{T}_L^{(1)},\ldots,\tilde{T}_L^{(n)}\}$. Now set
$$c_t^{(1)}\leftarrow\texttt{Com}(\tilde{t}^{(1)},r_t^{(1)}),c_L^{(1)}\leftarrow\texttt{Com}(\tilde{T}_L^{(1)},r_L^{(1)})$$
and $\tilde{\mathsf{vk}}\leftarrow\{c_t^{(i)},c_L^{(i)}\}_{i\in[n]}$
3. Compute $\tilde{M}\leftarrow\mathcal{A}(\mathcal{O}^f,\hat{M},\tilde{\mathsf{vk}})$.

This algorithm replaces the first element in a verification key with an element from an independently generated backdoor, and then runs $\mathcal{A}$ on it.

In $\mathcal{S}$ we only exchange one commitment when compared to the input distribution to $\mathcal{A}$ from the security game. By the statistical hiding of `Com`, the output of $\mathcal{S}$ must be distributed statistically close to the output of $\mathcal{A}$ in the unremovability experiment. Applying this repeatedly, we construct a sequence of hybrids $\mathcal{S}^{(1)},\mathcal{S}^{(2)},\ldots,\mathcal{S}^{(n)}$ that change $1,2,\ldots,n$ of the elements from vk in the same way that $\mathcal{S}$ does and conclude that the success of outputting a model $\tilde{M}$ without the watermark using $\mathcal{A}$ must be independent of vk.

Consider the following algorithm $\mathcal{T}$ when given a model M with a strong backdoor:

1. Compute $(\mathsf{mk},\mathsf{vk})\leftarrow\texttt{KeyGen}()$.
2. Run the adversary and compute $\tilde{N}\leftarrow\mathcal{A}(\mathcal{O}^f,M,\mathsf{vk})$.

By the hybrid argument above, the algorithm $\mathcal{T}$ runs nearly in the same time as $\mathcal{A}$, namely t, and its output $\tilde{N}$ will be without the backdoor that M contained. But then, by persistence of strong backdooring, $\mathcal{T}$ must also generate ε-accurate models given arbitrary, in particular bad input models M in the same time t, which contradicts our assumption that no such algorithm exists.

Unforgeability. Assume that there exists a poly-time algorithm $\mathcal{A}$ that can break unforgeability. We will use this algorithm to open a statistically hiding commitment.

Therefore, we design an algorithm $\mathcal{S}$ which uses $\mathcal{A}$ as a subroutine. The algorithm trains a regular network (which can be watermarked by our scheme) and adds the commitment into the verification key. Then, it will use $\mathcal{A}$ to find openings for these commitments. The algorithm $\mathcal{S}$ works as follows:

1. Receive the commitment c from challenger.
2. Compute $(M, \hat{M}, \mathsf{mk}, \mathsf{vk}) \leftarrow \texttt{MModel}()$.
3. Let $\mathsf{vk} = \{c_t^{(i)}, c_L^{(i)}\}_{i\in[n]}$ set
$$\hat{c}_t^{(i)} \leftarrow \begin{cases} c & \text{if } i = 1 \\ c_t^{(i)} & \text{else} \end{cases}$$
and $\hat{\mathsf{vk}} \leftarrow \{\hat{c}_t^{(i)}, c_L^{(i)}\}_{i\in[n]}$.
4. Compute $(\tilde{M}, \tilde{\mathsf{mk}}) \leftarrow \mathcal{A}(\mathcal{O}^f, \hat{M}, \hat{\mathsf{vk}})$.
5. Let $\tilde{\mathsf{mk}} = ((\{t^{(1)}, \ldots, t^{(n)}\}, T_L), \{r_t^{(i)}, r_L^{(i)}\}_{i\in[n]})$.
If $\texttt{Verify}(\tilde{\mathsf{mk}}, \hat{\mathsf{vk}}, \tilde{M}) = 1$ output $t^{(1)}, r_t^{(1)}$, else output $\perp$.

Since the commitment scheme is statistically hiding, the input to $\mathcal{A}$ is statistically indistinguishable from an input where $\hat{M}$ is backdoored on all the committed values of vk. Therefore the output of $\mathcal{A}$ in $\mathcal{S}$ is statistically indistinguishable from the output in the unforgeability definition. With the same probability as in the definition, $\tilde{\mathsf{mk}}, \hat{\mathsf{vk}}, \tilde{M}$ will make `Verify` output 1. But by its definition, this means that $\texttt{Open}(c, t^{(1)}, r_t^{(1)}) = 1$ so $t^{(1)}, r_t^{(1)}$ open the challenge commitment c. As the commitment is statistically hiding (and we generate the backdoor independently of c) this will open c to another value then for which it was generated with overwhelming probability. □

4.1 From Private to Public Verifiability

Using the algorithm `Verify` constructed in this section only allows verification by an honest party. The scheme described above is therefore only privately verifiable. After running `Verify`, the key mk will be known and an adversary can retrain the model on the trigger set. This is not a drawback when it comes to an application like the protection of intellectual property, where a trusted third party in the form of a judge exists. If one instead wants to achieve public verifiability, then there are two possible scenarios for how to design an algorithm `PVerify`: allowing public verification a constant number of times, or an arbitrary number of times.

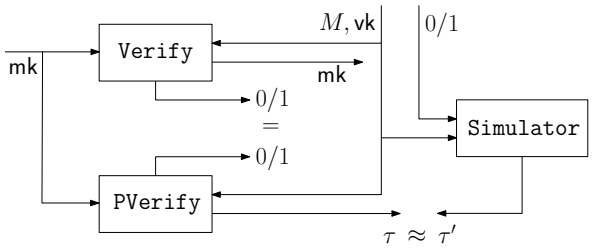

Figure 4: A schematic illustration of the public verification process.

In the first setting, a straightforward approach to the construction of `PVerify` is to choose multiple backdoors during `KeyGen` and release a different one in each iteration of `PVerify`. This allows multiple verifications, but the number is upper-bounded in practice by the capacity of the model M to contain backdoors - this cannot arbitrarily be extended without damaging the accuracy of the model. To achieve an unlimited number of verifications we will modify the watermarking scheme to output a different type of verification key. We then present an algorithm `PVerify` such that the interaction τ with an honest prover can be simulated as τ' given the values $M, \mathsf{vk}, \texttt{Verify}(\mathsf{mk}, \mathsf{vk}, M)$ only. This simulation means that no other information about mk beyond what is leaked from vk ever gets to the verifier. We give a graphical depiction of the approach in Figure 4. Our solution is sketched in Appendix A.1.

4.2 Implementation Details

For an implementation, it is of importance to choose the size $|T|$ of the trigger set properly, where we have to consider that $|T|$ cannot be arbitrarily big, as the accuracy will drop. To lower-bound $|T|$ we assume an attacker against non-trivial ownership. For simplicity, we use a backdooring algorithm that generates trigger sets from elements where f is undefined. By our simplifying assumption from Section 2.1, the model will classify the images in the trigger set to random labels. Furthermore, assume that the model is ε-accurate (which it also is on the trigger set). Then, one can model a dishonest party to randomly get $(1-\varepsilon)|T|$ out of $|T|$ committed images right using a Binomial distribution. We want to upper-bound this event to have probability at most 2^{-n} and use Hoeffding's inequality to obtain that $|T| > n \cdot \ln(2)/(\frac{1}{|L|} + \varepsilon - 1)$.

To implement our scheme, it is necessary that vk becomes public before `Verify` is used. This ensures that

a party does not simply generate a fake key after seeing a model. A solution for this is to e.g. publish the key on a time-stamped bulletin board like a blockchain. In addition, a statistically hiding commitment scheme should be used that allows for efficient evaluation in zero-knowledge (see Appendix A.1). For this one can e.g. use a scheme based on a cryptographic hash function such as the one described in [39].

5 A Direct Construction of Watermarking

This section describes a scheme for watermarking a neural network model for image classification, and experiments analyzing it with respect to the definitions in Section 3. We demonstrate that it is hard to reduce the persistence of watermarks that are generated with our method. For all the technical details regarding the implementation and hyper-parameters, we refer the reader to Section 5.7.

5.1 The Construction

Similar to Section 4, we use a set of images as the *marking key* or *trigger set* of our construction[4]. To embed the watermark, we optimize the models using both training set and trigger set. We investigate two approaches: the first approach starts from a pre-trained model, i.e., a model that was trained without a trigger set, and continues training the model together with a chosen trigger set. This approach is denoted as PRETRAINED. The second approach trains the model from scratch along with the trigger set. This approach is denoted as FROMSCRATCH. This latter approach is related to *Data Poisoning* techniques.

During training, for each batch, denote as b_t the batch at iteration t, we sample k trigger set images and append them to b_t. We follow this procedure for both approaches. We tested different numbers of k (i.e., 2, 4, and 8), and setting $k = 2$ reach the best results. We hypothesize that this is due to the *Batch-Normalization* layer [23]. The Batch-Normalization layer has two modes of operations. During training, it keeps a running estimate of the computed mean and variance. During an evaluation, the running mean and variance are used for normalization. Hence, adding more images to each batch puts more focus on the trigger set images and makes convergence slower.

In all models we optimize the Negative Log Likelihood loss function on both training set and *trigger set*. Notice, we assume the creator of the model will be the one who embeds the watermark, hence has access to the training set, test set, and *trigger set*.

In the following subsections, we demonstrate the efficiency of our method regarding non-trivial ownership and unremovability and furthermore show that it is functionality-preserving, following the ideas outlined in Section 3. For that we use three different image classification datasets: CIFAR-10, CIFAR-100 and ImageNet [28, 37]. We chose those datasets to demonstrate that our method can be applied to models with a different number of classes and also for large-scale datasets.

5.2 Non-Trivial Ownership

In the *non-trivial ownership* setting, an adversary will not be able to claim ownership of the model even if he knows the watermarking algorithm. To fulfill this requirement we randomly sample the examples for the trigger set. We sampled a set of 100 abstract images, and for each image, we randomly selected a target class.

This sampling-based approach ensures that the examples from the trigger set are uncorrelated to each other. Therefore revealing a subset from the trigger set will not reveal any additional information about the other examples in the set, as is required for public verifiability. Moreover, since both examples and labels are chosen randomly, following this method makes back-propagation based attacks extremely hard. Figure 5 shows an example from the trigger set.

Figure 5: An example image from the trigger set. The label that was assigned to this image was “automobile”.

5.3 Functionality-Preserving

For the *functionality-preserving* property we require that a model with a watermark should be as accurate as a model without a watermark. In general, each task defines

[4] As the set of images will serve a similar purpose as the trigger set from backdoors in Section 2, we denote the marking key as trigger set throughout this section.

its own measure of performance [2, 25, 4, 3]. However, since in the current work we are focused on image classification tasks, we measure the accuracy of the model using the 0-1 loss.

Table 1 summarizes the test set and trigger-set classification accuracy on CIFAR-10 and CIFAR-100, for three different models; (i) a model with no watermark (NO-WM); (ii) a model that was trained with the trigger set from scratch (FROMSCRATCH); and (iii) a pre-trained model that was trained with the trigger set after convergence on the original training data set (PRETRAINED).

Model	Test-set acc.	Trigger-set acc.
CIFAR-10		
NO-WM	93.42	7.0
FROMSCRATCH	93.81	100.0
PRETRAINED	93.65	100.0
CIFAR-100		
NO-WM	74.01	1.0
FROMSCRATCH	73.67	100.0
PRETRAINED	73.62	100.0

Table 1: Classification accuracy for CIFAR-10 and CIFAR-100 datasets on the test set and trigger set.

It can be seen that all models have roughly the same test set accuracy and that in both FROMSCRATCH and PRETRAINED the trigger-set accuracy is 100%. Since the trigger-set labels were chosen randomly, the NO-WM models' accuracy depends on the number of classes. For example, the accuracy on CIFAR-10 is 7.0% while on CIFAR-100 is only 1.0%.

5.4 Unremovability

In order to satisfy the *unremovability* property, we first need to define the types of unremovability functions we are going to explore. Recall that our goal in the unremovability experiments is to investigate the robustness of the watermarked models against changes that aim to remove the watermark while keeping the same functionality of the model. Otherwise, one can set all weights to zero and completely remove the watermark but also destroy the model.

Thus, we are focused on *fine-tuning* experiments. In other words, we wish to keep or improve the performance of the model on the test set by carefully training it. Fine-tuning seems to be the most probable type of attack since it is frequently used and requires less computational resources and training data [38, 43, 35]. Since in our settings we would like to explore the robustness of the watermark against strong attackers, we assumed that the adversary can fine-tune the models using the same amount of training instances and epochs as in training the model.

An important question one can ask is: *when is it still my model?* or other words how much can I change the model and still claim ownership? This question is highly relevant in the case of watermarking. In the current work we handle this issue by measuring the performance of the model on the test set and trigger set, meaning that the original creator of the model can claim ownership of the model if the model is still ε-accurate on the original test set while also ε-accurate on the trigger set. We leave the exploration of different methods and of a theoretical definition of this question for future work.

Fine-Tuning. We define four different variations of fine-tuning procedures:

- *Fine-Tune Last Layer* (FTLL): Update the parameters of the last layer only. In this setting we freeze the parameters in all the layers except in the output layer. One can think of this setting as if the model outputs a new representation of the input features and we fine-tune only the output layer.
- *Fine-Tune All Layers* (FTAL): Update all the layers of the model.
- *Re-Train Last Layers* (RTLL): Initialize the parameters of the output layer with random weights and only update them. In this setting, we freeze the parameters in all the layers except for the output layer. The motivation behind this approach is to investigate the robustness of the watermarked model under noisy conditions. This can alternatively be seen as changing the model to classify for a different set of output labels.
- *Re-Train All Layers* (RTAL): Initialize the parameters of the output layer with random weights and update the parameters in all the layers of the network.

Figure 6 presents the results for both the PRETRAINED and FROMSCRATCH models over the test set and trigger set, after applying these four different fine-tuning techniques.

The results suggest that while both models reach almost the same accuracy on the test set, the FROMSCRATCH models are superior or equal to the PRETRAINED models overall fine-tuning methods. FROMSCRATCH reaches roughly the same accuracy on the trig-

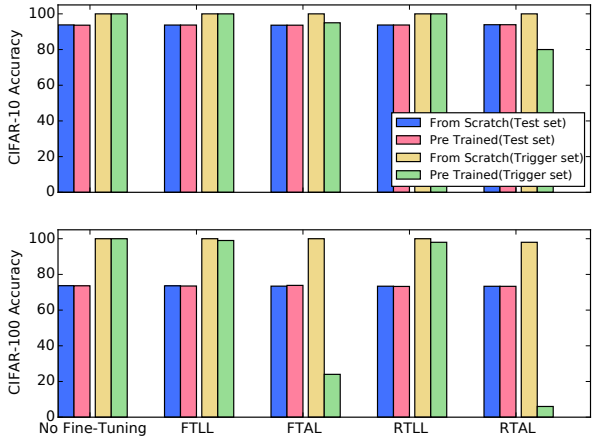

Figure 6: Classification accuracy on the test set and trigger set for CIFAR-10 (top) and CIFAR-100 (bottom) using different fine-tuning techniques. For example, in the bottom right bars we can see that the PRETRAINED model (green) suffers a dramatic decrease in the results comparing the baseline (bottom left) using the RTAL technique.

ger set when each of the four types of fine-tuning approaches is applied.

Notice that this observation holds for both the CIFAR-10 and CIFAR-100 datasets, where for CIFAR-100 it appears to be easier to remove the trigger set using the PRETRAINED models. Concerning the above-mentioned results, we now investigate what will happen if an adversary wants to embed a watermark in a model which has already been watermarked. This can be seen as a black-box attack on the already existing watermark. According to the fine-tuning experiments, removing this new trigger set using the above fine-tuning approaches will not hurt the original trigger set and will dramatically decrease the results on the new trigger set. In the next paragraph, we explore and analyze this setting. Due to the fact that FROMSCRATCH models are more robust than PRETRAINED, for the rest of the paper, we report the results for those models only.

5.5 Ownership Piracy

As we mentioned in Section 3, in this set of experiments we explore the scenario where an adversary wishes to claim ownership of a model which has already been watermarked.

For that purpose, we collected a new trigger set of different 100 images, denoted as TS-NEW, and embedded it to the FROMSCRATCH model (this new set will be used by the adversary to claim ownership of the model). Notice that the FROMSCRATCH models were trained using a different trigger set, denoted as TS-ORIG. Then, we fine-tuned the models using RTLL and RTAL methods. In order to have a fair comparison between the robustness of the trigger sets after fine-tuning, we use the same amount of epochs to embed the new trigger set as we used for the original one.

Figure 7 summarizes the results on the test set, TS-NEW and TS-ORIG. We report results for both the FTAL and RTAL methods together with the baseline results of no fine tuning at all (we did not report here the results of FTLL and RTLL since those can be considered as the easy cases in our setting). The red bars refer to the model with no fine tuning, the yellow bars refer to the FTAL method and the blue bars refer to RTAL.

The results suggest that the original trigger set, TS-ORIG, is still embedded in the model (as is demonstrated in the right columns) and that the accuracy of classifying it even improves after fine-tuning. This may imply that the model embeds the trigger set in a way that is close to the training data distribution. However, in the new trigger set, TS-NEW, we see a significant drop in the accuracy. Notice, we can consider embedding TS-NEW as embedding a watermark using the PRETRAINED approach. Hence, this accuracy drop of TS-NEW is not surprising and goes in hand with the results we observed in Figure 6.

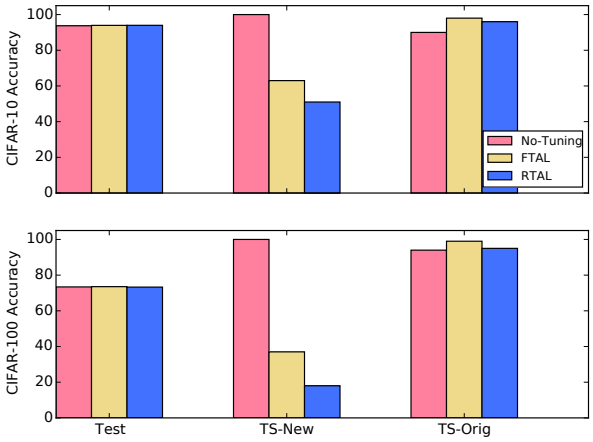

Figure 7: Classification accuracy on CIFAR-10 (top) and CIFAR-100 (bottom) datasets after embedding two trigger sets, TS-ORIG and TS-NEW. We present results for no tuning (red), FTAL (yellow) and TRAL (blue).

Transfer Learning. In transfer learning we would like to use knowledge gained while solving one problem and apply it to a different problem. For example, we use a trained model on one dataset (source dataset) and fine-tune it on a new dataset (target dataset). For that purpose, we fine-tuned the FROMSCRATCH model (which was trained on either CIFAR-10 or CIFAR-100), for another 20 epochs using the labeled part of the STL-10 dataset [12].

Recall that our watermarking scheme is based on the outputs of the model. As a result, when fine-tuning a model on a different dataset it is very likely that we change the number of classes, and then our method will probably break. Therefore, in order to still be able to verify the watermark we save the original output layer, so that on verification time we use the model's original output layer instead of the new one.

Following this approach makes both FTLL and RTLL useless due to the fact that these methods update the parameters of the output layer only. Regarding FTAL, this approach makes sense in specific settings where the classes of the source dataset are related to the target dataset. This property holds for CIFAR-10 but not for CIFAR-100. Therefore we report the results only for RTAL method.

Table 2 summarizes the classification accuracy on the test set of STL-10 and the trigger set after transferring from CIFAR-10 and CIFAR-100.

	Test set acc.	Trigger set acc.
CIFAR10 $\rightarrow$ STL10	81.87	72.0
CIFAR100 $\rightarrow$ STL10	77.3	62.0

Table 2: Classification accuracy on STL-10 dataset and the trigger set, after transferring from either CIFAR-10 or CIFAR-100 models.

Although the trigger set accuracy is smaller after transferring the model to a different dataset, results suggest that the trigger set still has a lot of presence in the network even after fine-tuning on a new dataset.

5.6 ImageNet - Large Scale Visual Recognition Dataset

For the last set of experiments, we would like to explore the robustness of our watermarking method on a large scale dataset. For that purpose, we use ImageNet dataset [37] which contains about 1.3 million training images with over 1000 categories.

Table 3 summarizes the results for the *functionality-preserving* tests. We can see from Table 3 that both models, with and without watermark, achieve roughly the same accuracy in terms of Prec@1 and Prec@5, while the model without the watermark attains 0% on the trigger set and the watermarked model attain 100% on the same set.

	Prec@1	Prec@5
Test Set		
NO-WM	66.64	87.11
FROMSCRATCH	66.51	87.21
Trigger Set		
NO-WM	0.0	0.0
FROMSCRATCH	100.0	100.0

Table 3: ImageNet results, Prec@1 and Prec@5, for a ResNet18 model with and without a watermark.

Notice that the results we report for ResNet18 on ImageNet are slightly below what is reported in the literature. The reason beyond that is due to training for fewer epochs (training a model on ImageNet is computationally expensive, so we train our models for fewer epochs than what is reported).

In Table 4 we report the results of transfer learning from ImageNet to ImageNet, those can be considered as FTAL, and from ImageNet to CIFAR-10, can be considered as RTAL or transfer learning.

	Prec@1	Prec@5
Test Set		
ImageNet $\rightarrow$ ImageNet	66.62	87.22
ImageNet $\rightarrow$ CIFAR-10	90.53	99.77
Trigger Set		
ImageNet $\rightarrow$ ImageNet	100.0	100.0
ImageNet $\rightarrow$ CIFAR-10	24.0	52.0

Table 4: ImageNet results, Prec@1 and Prec@5, for fine tuning using ImageNet and CIFAR-10 datasets.

Notice that after fine tuning on ImageNet, trigger set results are still very high, meaning that the trigger set has a very strong presence in the model also after fine-tuning. When transferring to CIFAR-10, we see a drop in the Prec@1 and Prec@5. However, considering the fact that ImageNet contains 1000 target classes, these results are still significant.

5.7 Technical Details

We implemented all models using the PyTorch package [33]. In all the experiments we used a ResNet-18 model, which is a convolutional based neural network

model with 18 layers [20, 21]. We optimized each of the models using Stochastic Gradient Descent (SGD), using a learning rate of 0.1. For CIFAR-10 and CIFAR-100 we trained the models for 60 epochs while halving the learning rate by ten every 20 epochs. For ImageNet we trained the models for 30 epochs while halving the learning rate by ten every ten epochs. The batch size was set to 100 for the CIFAR10 and CIFAR100, and to 256 for ImageNet. For the fine-tuning tasks, we used the last learning rate that was used during training.

6 Conclusion and Future Work

In this work we proposed a practical analysis of the ability to watermark a neural network using random training instances and random labels. We presented possible attacks that are both black-box and grey-box in the model, and showed how robust our watermarking approach is to them. At the same time, we outlined a theoretical connection to the previous work on backdooring such models.

For future work we would like to define a theoretical boundary for how much change must a party apply to a model before he can claim ownership of the model. We also leave as an open problem the construction of a practically efficient zero-knowledge proof for our publicly verifiable watermarking construction.

Acknowledgments

This work was supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Directorate in the Prime Minister's Office.

References

[1] ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., KUDLUR, M., LEVENBERG, J., MONGA, R., MOORE, S., MURRAY, D. G., STEINER, B., TUCKER, P., VASUDEVAN, V., WARDEN, P., WICKE, M., YU, Y., AND ZHENG, X. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2016), OSDI'16, USENIX Association, pp. 265–283.

[2] ADI, Y., AND KESHET, J. Structed: risk minimization in structured prediction. *The Journal of Machine Learning Research 17*, 1 (2016), 2282–2286.

[3] ADI, Y., KESHET, J., CIBELLI, E., AND GOLDRICK, M. Sequence segmentation using joint rnn and structured prediction models. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on* (2017), IEEE, pp. 2422–2426.

[4] ADI, Y., KESHET, J., CIBELLI, E., GUSTAFSON, E., CLOPPER, C., AND GOLDRICK, M. Automatic measurement of vowel duration via structured prediction. *The Journal of the Acoustical Society of America 140*, 6 (2016), 4517–4527.

[5] AMODEI, D., ANUBHAI, R., BATTENBERG, E., CASE, C., CASPER, J., CATANZARO, B., CHEN, J., CHRZANOWSKI, M., COATES, A., DIAMOS, G., ET AL. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning* (2016), pp. 173–182.

[6] BAHDANAU, D., CHO, K., AND BENGIO, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[7] BARAK, B., GOLDREICH, O., IMPAGLIAZZO, R., RUDICH, S., SAHAI, A., VADHAN, S., AND YANG, K. On the (im) possibility of obfuscating programs. *Journal of the ACM (JACM) 59*, 2 (2012), 6.

[8] BONEH, D., AND SHAW, J. Collusion-secure fingerprinting for digital data. In *Advances in Cryptology — CRYPTO' 95* (1995), D. Coppersmith, Ed., Springer, pp. 452–465.

[9] BRASSARD, G., CHAUM, D., AND CRÉPEAU, C. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci. 37*, 2 (1988), 156–189.

[10] CHEN, H., ROHANI, B. D., AND KOUSHANFAR, F. Deepmarks: A digital fingerprinting framework for deep neural networks, 2018.

[11] CISSE, M. M., ADI, Y., NEVEROVA, N., AND KESHET, J. Houdini: Fooling deep structured visual and speech recognition models with adversarial examples. In *Advances in Neural Information Processing Systems* (2017), pp. 6980–6990.

[12] COATES, A., NG, A., AND LEE, H. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (2011), pp. 215–223.

[13] FIAT, A., AND SHAMIR, A. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques* (1986), Springer, pp. 186–194.

[14] GOLDREICH, O. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[15] GOLDWASSER, S., MICALI, S., AND RACKOFF, C. The knowledge complexity of interactive proof-systems (extended abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA* (1985), pp. 291–304.

[16] GOODFELLOW, I. J., SHLENS, J., AND SZEGEDY, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[17] GRAVES, A., FERNÁNDEZ, S., GOMEZ, F., AND SCHMIDHUBER, J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning* (2006), ACM, pp. 369–376.

[18] GU, T., DOLAN-GAVITT, B., AND GARG, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *CoRR abs/1708.06733* (2017).

[19] HE, K., GKIOXARI, G., DOLLÁR, P., AND GIRSHICK, R. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on* (2017), IEEE, pp. 2980–2988.

[20] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.

[21] HE, K., ZHANG, X., REN, S., AND SUN, J. Identity mappings in deep residual networks. In *European Conference on Computer Vision* (2016), Springer, pp. 630–645.

[22] HOSSEINI, H., CHEN, Y., KANNAN, S., ZHANG, B., AND POOVENDRAN, R. Blocking transferability of adversarial examples in black-box learning systems. *arXiv preprint arXiv:1703.04318* (2017).

[23] IOFFE, S., AND SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (2015), pp. 448–456.

[24] KATZENBEISSER, S., AND PETITCOLAS, F. *Information hiding*. Artech house, 2016.

[25] KESHET, J. Optimizing the measure of performance in structured prediction. *Advanced Structured Prediction. The MIT Press. URL http://u. cs. biu. ac. il/˜ jkeshet/papers/Keshet14. pdf* (2014).

[26] KIM, S., AND WU, D. J. Watermarking cryptographic functionalities from standard lattice assumptions. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I* (2017), pp. 503–536.

[27] KREUK, F., ADI, Y., CISSE, M., AND KESHET, J. Fooling end-to-end speaker verification by adversarial examples. *arXiv preprint arXiv:1801.03339* (2018).

[28] KRIZHEVSKY, A., AND HINTON, G. Learning multiple layers of features from tiny images.

[29] LECUN, Y., BENGIO, Y., AND HINTON, G. Deep learning. *Nature 521*, 7553 (2015), 436–444.

[30] LIU, Y., MA, S., AAFER, Y., LEE, W.-C., AND ZHAI, J. Trojaning attack on neural networks. Tech Report, 2017.

[31] MERRER, E. L., PEREZ, P., AND TRÉDAN, G. Adversarial frontier stitching for remote neural network watermarking, 2017.

[32] NAOR, D., NAOR, M., AND LOTSPIECH, J. Revocation and tracing schemes for stateless receivers. In *Annual International Cryptology Conference* (2001), Springer, pp. 41–62.

[33] PASZKE, A., GROSS, S., CHINTALA, S., CHANAN, G., YANG, E., DEVITO, Z., LIN, Z., DESMAISON, A., ANTIGA, L., AND LERER, A. Automatic differentiation in pytorch.

[34] PETITCOLAS, F. A., ANDERSON, R. J., AND KUHN, M. G. Information hiding-a survey. *Proceedings of the IEEE 87*, 7 (1999), 1062–1078.

[35] RAZAVIAN, A. S., AZIZPOUR, H., SULLIVAN, J., AND CARLSSON, S. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on* (2014), IEEE, pp. 512–519.

[36] RIBEIRO, M., GROLINGER, K., AND CAPRETZ, M. A. Mlaas: Machine learning as a service. In *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on* (2015), IEEE, pp. 896–902.

[37] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., ET AL. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision 115*, 3 (2015), 211–252.

[38] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[39] SMART, N. P. *Cryptography Made Simple*. Information Security and Cryptography. Springer, 2016.

[40] TOSHEV, A., AND SZEGEDY, C. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 1653–1660.

[41] UCHIDA, Y., NAGAI, Y., SAKAZAWA, S., AND SATOH, S. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval* (2017), ACM, pp. 269–277.

[42] VENUGOPAL, A., USZKOREIT, J., TALBOT, D., OCH, F. J., AND GANITKEVITCH, J. Watermarking the outputs of structured prediction with an application in statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (2011), Association for Computational Linguistics, pp. 1363–1372.

[43] YOSINSKI, J., CLUNE, J., BENGIO, Y., AND LIPSON, H. How transferable are features in deep neural networks? In *Advances in neural information processing systems* (2014), pp. 3320–3328.

[44] ZHANG, C., BENGIO, S., HARDT, M., RECHT, B., AND VINYALS, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530* (2016).

A Supplementary Material

In this appendix we further discuss how to achieve public verifiability for a variant of our watermarking scheme. Let us first introduce the following additional notation: for a vector $\mathbf{e} \in \{0,1\}^\ell$, let $\mathbf{e}|_0 = \{i \in [\ell] \mid \mathbf{e}[i] = 0\}$ be the set of all indices where $\mathbf{e}$ is 0 and define $\mathbf{e}|_1$ accordingly. Given a verification key $\mathsf{vk} = \{c_t^{(i)}, c_L^{(i)}\}_{i\in[\ell]}$ containing ℓ elements and a vector $\mathbf{e} \in \{0,1\}^\ell$, we write the selection of elements from vk according to $\mathbf{e}$ as

$$\mathsf{vk}|_0^{\mathbf{e}} = \{c_t^{(i)}, c_L^{(i)}\}_{i\in\mathbf{e}|_0} \quad \text{and} \quad \mathsf{vk}|_1^{\mathbf{e}} = \{c_t^{(i)}, c_L^{(i)}\}_{i\in\mathbf{e}|_1}.$$

For a marking key $\mathsf{mk} = (\mathsf{b}, \{r_t^{(i)}, r_L^{(i)}\}_{i\in[\ell]})$ with ℓ elements and $\mathsf{b} = \{T^{(i)}, T_L^{(i)}\}_{i\in[\ell]}$ we then define

$$\mathsf{mk}|_0^{\mathbf{e}} = (\mathsf{b}|_0^{\mathbf{e}}, \{r_t^{(i)}, r_L^{(i)}\}_{i\in\mathbf{e}|_0}) \text{ with } \mathsf{b}|_0^{\mathbf{e}} = \{T^{(i)}, T_L^{(i)}\}_{i\in\mathbf{e}|_0}$$

(and $\mathsf{mk}|_1^{\mathbf{e}}$ accordingly). We assume the existence of a cryptographic hash function $H : \{0,1\}^{p(n)} \to \{0,1\}^n$.

A.1 From Private to Public Verifiability

To achieve public verifiability, we will make use of a cryptographic tool called a *zero-knowledge argument* [15], which is a technique that allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that a certain public statement is true, without giving away any further information. This idea is similar to the idea of unlimited public verification as outlined in Section 4.1.

Zero-Knowledge Arguments. Let TM be an abbreviation for Turing Machines. An iTM is defined to be an interactive TM, i.e. a Turing Machine with a special communication tape. Let $L_R \subseteq \{0,1\}^*$ be an NP language and R be its related NP-relation, i.e. $(x,w) \in R$ iff $x \in L_R$ and the TM used to define L_R outputs 1 on input of the statement x and the witness w. We write $R_x = \{w \mid (x,w) \in R\}$ for the set of witnesses for a fixed x. Moreover, let $\mathcal{P}, \mathcal{V}$ be a pair of PPT iTMs. For $(x,w) \in R$, $\mathcal{P}$ will obtain w as input while $\mathcal{V}$ obtains an auxiliary random string $z \in \{0,1\}^*$. In addition, x will be input to both TMs. Denote with $\mathcal{V}^{\mathcal{P}(a)}(b)$ the output of the iTM $\mathcal{V}$ with input b when communicating with an instance of $\mathcal{P}$ that has input a.

$(\mathcal{P}, \mathcal{V})$ is called an *interactive proof system* for the language L if the following two conditions hold:

Completeness: For every $x \in L_R$ there exists a string w such that for every z: $\Pr[\mathcal{V}^{\mathcal{P}(x,w)}(x,z) = 1]$ is negligibly close to 1.

Soundness: For every $x \notin L_R$, every PPT iTM $\mathcal{P}^*$ and every string w,z: $\Pr[\mathcal{V}^{\mathcal{P}^*(x,w)}(x,z) = 1]$ is negligible.

An interactive proof system is called computational *zero-knowledge* if for every PPT $\hat{\mathcal{V}}$ there exists a PPT simulator $\mathcal{S}$ such that for any $x \in L_R$

$$\{\hat{\mathcal{V}}^{\mathcal{P}(x,w)}(x,z)\}_{w\in R_x, z\in\{0,1\}^*} \approx_c \{\mathcal{S}(x,z)\}_{z\in\{0,1\}^*},$$

meaning that all information which can be learned from observing a protocol transcript can also be obtained from running a polynomial-time simulator $\mathcal{S}$ which has no knowledge of the witness w.

A.1.1 Outlining the Idea

An intuitive approach to build `PVerify` is to convert the algorithm $\texttt{Verify}(\mathsf{mk}, \mathsf{vk}, M)$ from Section 4 into an NP relation R and use a zero-knowledge argument system. Unfortunately, this must fail due to Step 1 of `Verify`: there, one tests if the item b contained in mk actually is a backdoor as defined above. Therefore, we would need access to the ground-truth function f in the interactive argument system. This first of all needs human assistance, but is moreover only possible by revealing the backdoor elements.

We will now give a different version of the scheme from Section 4 which embeds an additional proof into vk. This proof shows that, with overwhelming probability, most of the elements in the verification key indeed form a backdoor. Based on this, we will then design a different verification procedure, based on a zero-knowledge argument system.

A.1.2 A Convincing Argument that most Committed Values are Wrongly Classified

Verifying that most of the elements of the trigger set are labeled wrongly is possible, if one accepts[5] to release a portion of this set. To solve the proof-of-misclassification problem, we use the so-called *cut-and-choose* technique: in cut-and-choose, the verifier $\mathcal{V}$ will ask the prover $\mathcal{P}$ to open a subset of the committed inputs and labels from the verification key. Here, $\mathcal{V}$ is allowed to choose the subset that will be opened to him. Intuitively, if $\mathcal{P}$ committed to a large number elements that are correctly labeled (according to $\mathcal{O}_f$), then at least one of them will show up in the values opened by $\mathcal{P}$ with overwhelming probability over the choice that $\mathcal{V}$ makes. Hence, most of the remaining commitments which were not opened must form a correct backdoor.

[5] This is fine if T, as in our experiments, only consists of random images.

To use cut-and-choose, the backdoor size must contain $\ell > n$ elements, where our analysis will use $\ell = 4n$ (other values of ℓ are also possible). Then, consider the following protocol between $\mathcal{P}$ and $\mathcal{V}$:

$\texttt{CnC}(\ell)$:

1. $\mathcal{P}$ runs $(\mathsf{mk}, \mathsf{vk}) \leftarrow \texttt{KeyGen}(\ell)$ to obtain a backdoor of size ℓ and sends vk to $\mathcal{V}$. We again define $\mathsf{mk} = (\mathsf{b}, \{r_t^{(i)}, r_L^{(i)}\}_{i\in[\ell]})$, $\mathsf{vk} = \{c_t^{(i)}, c_L^{(i)}\}_{i\in[\ell]}$
2. $\mathcal{V}$ chooses $\mathbf{e} \leftarrow \{0,1\}^\ell$ uniformly at random and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ sends $\mathsf{mk}|_1^{\mathbf{e}}$ to $\mathcal{V}$.
4. $\mathcal{V}$ checks that for $i \in \mathbf{e}|_1$ that
   (a) $\texttt{Open}(c_t^{(i)}, t^{(i)}, r_t^{(i)}) = 1$;
   (b) $\texttt{Open}(c_L^{(i)}, T_L^{(i)}, r_L^{(i)}) = 1$; and
   (c) $T_L^{(i)} \neq f(t^{(i)})$.

Assume that $\mathcal{P}$ chose exactly one element of the backdoor in vk wrongly, then this will be revealed by $\texttt{CnC}$ to an honest $\mathcal{V}$ with probability $1/2$ (where $\mathcal{P}$ must open $\mathsf{vk}|_1^{\mathbf{e}}$ to the values he put into $c_t^{(i)}, c_L^{(i)}$ during $\texttt{KeyGen}$ due to the binding-property of the commitment). In general, one can show that a cheating $\mathcal{P}$ can put at most n non-backdooring inputs into $\mathsf{vk}|_0^{\mathbf{e}}$ except with probability negligible in n. Therefore, if the above check passes for $\ell = 4n$ at then least $1/2$ of the values for $\mathsf{vk}|_0^{\mathbf{e}}$ must have the wrong committed label as in a valid backdoor with overwhelming probability.

The above argument can be made non-interactive and thus publicly verifiable using the Fiat-Shamir transform[13]: in the protocol $\texttt{CnC}$, $\mathcal{P}$ can generate the bit string $\mathbf{e}$ itself by hashing vk using a cryptographic hash function H. Then $\mathbf{e}$ will be distributed as if it was chosen by an honest verifier, while it is sufficiently random by the guarantees of the hash function to allow the same analysis for cut-and-choose. Any $\mathcal{V}$ can recompute the value $\mathbf{e}$ if it is generated from the commitments (while this also means that the challenge $\mathbf{e}$ is generated after the commitments were computed), and we can turn the above algorithm $\texttt{CnC}$ into the following non-interactive key-generation algorithm $\texttt{PKeyGen}$.

$\texttt{PKeyGen}(\ell)$:

1. Run $(\mathsf{mk}, \mathsf{vk}) \leftarrow \texttt{KeyGen}(\ell)$.
2. Compute $\mathbf{e} \leftarrow H(\mathsf{vk})$.
3. Set $\mathsf{mk}_p \leftarrow (\mathsf{mk}, \mathbf{e})$, $\mathsf{vk}_p \leftarrow (\mathsf{vk}, \mathsf{mk}|_1^{\mathbf{e}})$ and return $(\mathsf{mk}_p, \mathsf{vk}_p)$.

A.1.3 Constructing the Public Verification Algorithm

In the modified scheme, the $\texttt{Mark}$ algorithm will only use the private subset $\mathsf{mk}|_0^{\mathbf{e}}$ of mk_p but will otherwise remain unchanged. The public verification algorithm for a model M then follows the following structure: (i) $\mathcal{V}$ recomputes the challenge $\mathbf{e}$; (ii) $\mathcal{V}$ checks vk_p to assure that all of $\mathsf{vk}|_1^{\mathbf{e}}$ will form a valid backdoor ; and (iii) $\mathcal{P}, \mathcal{V}$ run $\texttt{Classify}$ on $\mathsf{mk}|_0^{\mathbf{e}}$ using the interactive zero-knowledge argument system, and further test if the watermarking conditions on $M, \mathsf{mk}|_0^{\mathbf{e}}, \mathsf{vk}|_0^{\mathbf{e}}$ hold.

For an arbitrary model M, one can rewrite the steps 2 and 3 of $\texttt{Verify}$ (using $M, \texttt{Open}, \texttt{Classify}$) into a binary circuit C that outputs 1 iff the prover inputs the correct $\mathsf{mk}|_0^{\mathbf{e}}$ which opens $\mathsf{vk}|_0^{\mathbf{e}}$ and if enough of these openings satisfy $\texttt{Classify}$. Both $\mathcal{P}, \mathcal{V}$ can generate this circuit C as its construction does not involve private information. For the interactive zero-knowledge argument, we let the relation R be defined by boolean circuits that output 1 where $x = C, w = \mathsf{mk}|_0^{\mathbf{e}}$ in the following protocol $\texttt{PVerify}$, which will obtain the model M as well as $\mathsf{mk}_p = (\mathsf{mk}, \mathbf{e})$ and $\mathsf{vk}_p = (\mathsf{vk}, \mathsf{mk}|_1^{\mathbf{e}})$ where $\mathsf{vk} = \{c_t^{(i)}, c_L^{(i)}\}_{i\in[\ell]}$, $\mathsf{mk} = (\mathsf{b}, \{r_t^{(i)}, r_L^{(i)}\}_{i\in[\ell]})$ and $\mathsf{b} = \{T^{(i)}, T_L^{(i)}\}_{i\in[\ell]}$ as input.

1. $\mathcal{V}$ computes $\mathbf{e}' \leftarrow H(\mathsf{vk})$. If $\mathsf{mk}|_1^{\mathbf{e}}$ in vk_p does not match $\mathbf{e}'$ then abort, else continue assuming $\mathbf{e} = \mathbf{e}'$.
2. $\mathcal{V}$ checks that for all $i \in \mathbf{e}|_1$:
   (a) $\texttt{Open}(c_t^{(i)}, t^{(i)}, r_t^{(i)}) = 1$
   (b) $\texttt{Open}(c_L^{(i)}, T_L^{(i)}, r_L^{(i)}) = 1$
   (c) $T_L^{(i)} \neq f(t^{(i)})$

   If one of the checks fails, then $\mathcal{V}$ aborts.
3. $\mathcal{P}, \mathcal{V}$ compute a circuit C with input $\mathsf{mk}|_0^{\mathbf{e}}$ that outputs 1 iff for all $i \in \mathbf{e}|_0$:
   (a) $\texttt{Open}(c_t^{(i)}, t^{(i)}, r_t^{(i)}) = 1$
   (b) $\texttt{Open}(c_L^{(i)}, T_L^{(i)}, r_L^{(i)}) = 1$.

   Moreover, it tests that $\texttt{Classify}(t^{(i)}, M) = T_L^{(i)}$ for all but $\varepsilon|\mathbf{e}|_0|$ elements.
4. $\mathcal{P}, \mathcal{V}$ run a zero-knowledge argument for the given relation R using C as the statement, where the witness $\mathsf{mk}|_0^{\mathbf{e}}$ is the secret input of $\mathcal{P}$. $\mathcal{V}$ accepts iff the argument succeeds.

Assume the protocol `PVerify` succeeds. Then in the interactive argument, M classifies at least $(1-\varepsilon)|\mathbf{e}|_0| \approx (1-\varepsilon)2n$ values of the backdoor `b` to the committed value. For $\approx n$ of the commitments, we can assume that the committed label does not coincide with the ground-truth function f due to the guarantees of Step 1. It is easy to see that this translates into a 2ε-guarantee for the correct backdoor. By choosing a larger number ℓ for the size of the backdoor, one can achieve values that are arbitrarily close to ε in the above protocol.