EncryIP: A Practical Encryption-Based Framework for Model Intellectual Property Protection

Xin Mu[1], **Yu Wang**[1], **Zhengan Huang**[1], **Junzuo Lai**[2], **Yehong Zhang**[1], **Hui Wang**[1], **Yue Yu**[1*]

[1]Peng Cheng Laboratory, Shenzhen 518000, China

[2]College of Information Science and Technology, Jinan University, Guangzhou, China

{mux, wangy12, wangh06, yuy}@pcl.ac.cn, {zhahuang.sjtu, laijunzuo, zyhredleaf}@gmail.com

Abstract

In the rapidly growing digital economy, protecting intellectual property (IP) associated with digital products has become increasingly important. Within this context, machine learning (ML) models, being highly valuable digital assets, have gained significant attention for IP protection. This paper introduces a practical encryption-based framework called *EncryIP*, which seamlessly integrates a public-key encryption scheme into the model learning process. This approach enables the protected model to generate randomized and confused labels, ensuring that only individuals with accurate secret keys, signifying authorized users, can decrypt and reveal authentic labels. Importantly, the proposed framework not only facilitates the protected model to multiple authorized users without requiring repetitive training of the original ML model with IP protection methods but also maintains the model's performance without compromising its accuracy. Compared to existing methods like watermark-based, trigger-based, and passport-based approaches, *EncryIP* demonstrates superior effectiveness in both training protected models and efficiently detecting the unauthorized spread of ML models.

1 Introduction

Recent years have witnessed a surge in interest regarding intellectual property (IP) protection for ML models, with attention spanning both academia and industry (Xue, Wang, and Liu 2021; Chen et al. 2018; Tauhid et al. 2023). The central theme of current approaches revolves around embedding distinctive information within the ML model or its training data. This empowers users to verify ownership of the ML model by confirming this embedded information. Broadly, these methods can be categorized into three types: (1) **Watermark-based method:** It incorporates designated watermarks into model parameters, strategically employing regularization terms for embedding (Boenisch 2020; Zhang et al. 2022); (2) **Trigger-based method:** This method assigns a specific value to a trigger set during model training. The model outputs this value when the same trigger set is used for verification (Zhang et al. 2018); and (3) **Passport-based method:** By introducing a new passport layer into the original deep neural network model, this technique uses a set of data as the passport for ownership verification. The performance of the original task will significantly deteriorate if a wrong passport is used (Fan et al. 2022).

While the mentioned methods can indeed create protected ML models, they remain inadequate in addressing the issue of unauthorized model distribution—when the model is used by unintended users. For instance, envision an AI company developing and licensing an ML model to multiple users. The company's aim is to restrict the model's use to authorized users solely. However, practical scenarios often witness authorized users sharing the model with unauthorized counterparts, bypassing the company's consent. Moreover, the risk persists of the model being stolen and employed by unauthorized entities.

To address this challenge, a prevailing strategy entails allocating different versions of same model to various authorized users. By overseeing the utilization of these model variations, the model creator can discern whether a specific version has been disseminated to unintended users. However, directly implementing this strategy necessitates creating separate protected model iterations for each user. This approach entails multiple rounds of model training using existing IP protection techniques. Unfortunately, this method becomes impractical in real-world scenarios due to several reasons:

• **Large-Scale Models.** Training large-scale models such as GPT-4 (OpenAI 2023) or DALL-E (Ramesh et al. 2021) takes substantial time, often spanning days or months. Requiring multiple training for these models would incur significant costs in terms of computing resources, time, and manpower.

• **Restricted Usage.** In certain scenarios, like military or sensitive data, the utilization of data might be limited due to confidentiality agreements or proprietary concerns. In such cases, employing a strategy involving multiple rounds of training is impractical.

Based on the above demands of practice, we present *EncryIP*, an Encryption-based framework for Intellectual property Protection of machine learning model. This approach generates multiple safeguarded model versions through a single training iteration. *EncryIP* considers a special kind of public-key encryption scheme, focusing on the dataset, particularly within the label space. Our framework unveils a novel encryption strategy, bridging the relationship between encryption schemes and machine learning algorithms. Empirical investigations encompass various ML models, validating

[*]Corresponding author.

the efficacy and efficiency of our approach.

The contributions of the paper are summarized as follows:

• We introduce a novel encryption-based framework called *EncryIP*, to address model intellectual property protection. Compared with existing solutions, *EncryIP* generates multiple protected versions of the same model through a single training process, eliminating the need for repetitive training. Furthermore, the protected model produced by *EncryIP* utilizes randomized labels, preventing unauthorized users without accurate secret keys from accessing true labels. Our approach satisfies IP protection requirements while surpassing the efficiency of existing methods.

• *EncryIP* represents a data-dependent strategy utilizing a public-key encryption scheme that interacts directly with the dataset. It avoids extensive model structural modifications or additional trigger datasets. It creates a degree of independence between the learning algorithm and the encryption algorithm, which makes it easier to operate on different structures of ML models. *EncryIP* provides a practical and versatile solution for protecting the intellectual property of machine learning models.

• We evaluate *EncryIP* on a great diversity of scenarios, including different model structures and data sets. The evaluation results provide valuable insights into the effectiveness and efficiency of *EncryIP* in real-world settings, demonstrating its ability to handle different model architectures and datasets with satisfactory outcomes.

2 Related Work

One direction is the **watermark-based method** (Uchida et al. 2017; Adi et al. 2018; Merrer, Pérez, and Trédan 2020; Rouhani, Chen, and Koushanfar 2019; Kuribayashi, Tanaka, and Funabiki 2020; Feng and Zhang 2020; Chen et al. 2019; Guo and Potkonjak 2018; Chen, Rouhani, and Koushanfar 2019; Kuribayashi, Tanaka, and Funabiki 2020). The basic idea is to design a specific embedding mechanism and utilize this mechanism in the training process. The specific embedding information can be viewed as one kind of watermark. User can verify the right of ownership by watermark. For instance, Uchida et al. (Uchida et al. 2017) proposed a general framework for embedding a watermark in model parameters by using a parameter regularizer. This regularizer can be used to embed a T-bit vector into model parameters, and all watermarks can be correctly detected by a simple linear transformation. For real applications, Guo and Potkonjak (Guo and Potkonjak 2018) proposed a watermarking framework enabling DNN watermarking on embedded devices.

The second is the **trigger-based method** (Zhang et al. 2018; Lukas, Zhang, and Kerschbaum 2021; Cao, Jia, and Gong 2021; Maung and Kiya 2020; Jebreel et al. 2021; Chen et al. 2017; Zhong et al. 2020; Cao, Jia, and Gong 2021; Zhang et al. 2020a). Generally, the algorithm employs a set of instances as a trigger set and embeds this trigger set information into ML model in the training procedure. The verification process is that model can output a specific result when this trigger set is treated as the input. For example, Lukas, Zhang, and Kerschbaum (Lukas, Zhang, and Kerschbaum 2021) proposed a fingerprinting method for deep neural network classifiers that extracts this trigger set from the original model, and this trigger set should have the same classification results with a copied model. Maung and Kiya (Maung and Kiya 2020) proposed a model protection method by using block-wise pixel shuffling with a secret key as a preprocessing technique to input images and training with such preprocessed images. In (Jebreel et al. 2021), a multitask learning IP protection method was proposed by learning the original classification task and the watermarking task together.

In recent years, a new way to rethink this problem has been proposed by introducing a novel **passport-based method** (Fan et al. 2022; Fan, Ng, and Chan 2019; Zhang et al. 2020b). The main motivation of embedding digital passports is to design and train DNN models in a way such that their inference performances of the original task (i.e., classification accuracy) will be significantly deteriorated due to the forged signatures. Zhang et al. (Zhang et al. 2020b) proposed a new passport-aware normalization formulation which builds the relationship between the model performance and the passport correctness. One advantage of the passport-based method is that it is robust to network modifications and resilient to ambiguity attacks simultaneously.

While current methods can generate protected models, they remain inadequate in addressing unauthorized model spread. For instance, high training costs make the watermark-based approach impractical, as it requires multiple training iterations for generating distinct versions. Although the trigger-based method achieves this with one training, its output is a true prediction, lacking security. Similarly, the passport-based method necessitates multiple passport groups, requiring complex training, and intricate modifications to the original model structure. In contrast, our *EncryIP* framework efficiently tackles unauthorized spread, providing effective protection without the drawbacks posed by other methods.

3 Preliminaries

The IP protection. Let training data set $D = \{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^d$ is a training instance and $y_i \in \mathbb{Y} = \{1, 2, \ldots, z\}$ is the associated class label. A machine learning model $\mathcal{M}$ is learned from D through an algorithm architecture $\mathcal{A}$. Following the definition in (Fan et al. 2022), the two main processes in the IP protection task are as follows:

1. Learning process $L(D, \mathcal{A}, \mathcal{A}_{\text{IP}}) = [\mathcal{M}]$, is a model training process that takes training data D and an IP protection method $\mathcal{A}_{\text{IP}}$ as inputs, and outputs a model $[\mathcal{M}]$. $[\cdot]$ is indicated as having the capacity of IP protection.
2. Deploying process $P([\mathcal{M}], x, \mathcal{A}_{\text{IP}}^{-1}) = y$, is that $[\mathcal{M}]$ outputs a results y when input x. Note that the output y is correct if and only if the user utilizes the correct inverse IP protection method $\mathcal{A}_{\text{IP}}^{-1}$, otherwise, a confused result will be generated.

In the **watermark-based method**, $\mathcal{A}_{\text{IP}}$ involves implanting distinct information into an ML model, often through specialized regularizers in the loss function. The **trigger-based method** relies on $\mathcal{A}_{\text{IP}}$ to introduce a specific dataset that can be learned during training. During verification, model $[\mathcal{M}]$ responds with the designated information upon receiving this unique dataset as input. In the **passport-based method**,

$\mathcal{A}_{\text{IP}}$ encompasses a novel structure integrated into the original model $\mathcal{M}$. The protected model $[\mathcal{M}]$ furnishes accurate outputs only upon receiving predetermined inputs.

In this paper, we elaborate on the concept of unauthorized model spread within the context of IP protection as follows:

Definition 1 (Unauthorized spread of model). *In the IP protection task, the output of a learning process $[\mathcal{M}]$ normally is authorized to a set of users $O = \{o_1, o_2, \ldots, o_J\}$. The unauthorized spread of model is that $[\mathcal{M}]$ is used by a user $o \notin O$.*

To meet this problem, we introduce a verification process in the IP protection task:

- Verification process $V([\mathcal{M}], o) = \{\text{True, False}\}$, is an authorization verification that verifies if a model user o belongs to an authorized model user set.

PKE with multiple secret keys. In this paper, *EncryIP* incorporates a distinct form of public-key encryption (PKE) scheme (Katz and Lindell 2020), characterized by the following requisites: i) the existence of multiple secret keys corresponding to a public key (i.e., each one of these secret keys can be used for decryption), and ii) the existence of some ill-formed ciphertexts, such that decrypting them with different secret keys (corresponding to the same public key) will lead to different messages.

To be specific, in this paper, a PKE scheme with the above properties consists of the following probabilistic algorithms.

- Gen: This is the key generation algorithm. It takes a number $P \in \mathbb{N}$ as input, and outputs a public key pk and P secret keys $\{sk_j\}_{j=1}^{P}$.
- Enc: This is the encryption algorithm, taking pk and a message m as input, and outputting a ciphertext c.
- Dec: This is the decryption algorithm, taking sk and a ciphertext c as input, and outputting a message m or $\perp$, which indicates that c is invalid.
- Fake: This is the fake encryption algorithm, taking pk as input, and outputting an ill-formed ciphertext c.

For correctness, we require that for any valid message m, and any $(pk, \{sk_j\}_{j=1}^{P}) \leftarrow \mathsf{Gen}(P)$:

(i) for any $j \in \{1, \cdots, P\}$, $\mathsf{Dec}(sk_j, \mathsf{Enc}(pk, m)) = m$.

(ii) for any $c \leftarrow \mathsf{Fake}(pk)$ and any distinct $j_1, j_2 \in \{1, \cdots, P\}$, $\mathsf{Dec}(sk_{j_1}, c) \neq \mathsf{Dec}(sk_{j_2}, c)$.

4 The Proposed Framework

4.1 *EncryIP*: An overview

The proposed *EncryIP* framework integrates a distinctive form of public-key encryption (PKE) scheme directly within a learning algorithm, primarily within the dataset's label space. *EncryIP* aims to achieve the following objectives: (1) It generates an interpretable output exclusively when the model user employs the correct secret key. (2) Different users possess distinct secret keys, enabling *EncryIP* to identify the unauthorized user during model misuse. (3) Integrating an encryption scheme does not significantly compromise the predictive performance of the ML model.

To realize the aforementioned objectives, *EncryIP* employs a three-step process:

- Learning Process. A learning process $L(D, \mathcal{A}, \mathsf{PKE}) = ([\mathcal{M}], \{sk_j\}_{j=1}^{P})$ first processes training data D by a PKE scheme PKE, and then executes a learning algorithm $\mathcal{A}$ to output the model $[\mathcal{M}]$ and a set of secret keys $\{sk_j\}_{j=1}^{P}$.
- Deploying Process. A deploying process $P(x, [\mathcal{M}], sk_j) = y$ is that the protected model $[\mathcal{M}]$ outputs the prediction result y when x is as its input.

  Remark: In general, the output of model $[\mathcal{M}]$ is correct if and only if the user uses correct sk_j to decrypt it, otherwise a confused result will be output.
- Verification Process. An verification process $V([\mathcal{M}], sk_j) = \{\text{True}, \text{False}\}$ is to evaluate if $[\mathcal{M}]$ belongs to its owner o_j.

The crucial aspect of *EncryIP* lies in the integration of an encryption scheme with a learning algorithm $\mathcal{A}$. In this paper, we propose a data-dependent IP protection method where the encryption scheme primarily focuses on the data, particularly in the label space. It can be described by

$$\min_{[\mathcal{M}]} \sum_{i=1}^{N} (\Pr[\psi^{-1}([\mathcal{M}](x_i)) = y_i)] - \Pr[\mathcal{M}(x_i) = y_i])$$

where ψ is an encryption function on a dataset D by $\psi(D) = (x_i, \psi_i(y_i))$, and ψ^{-1} is a decryption function, $D = \psi^{-1}(\psi(D))$. $\mathcal{M}$ is a model learned from D, e.g., $\mathcal{M} \leftarrow \mathcal{A}(D), y \leftarrow \mathcal{M}(x)$ and $[\mathcal{M}] \leftarrow \mathcal{A}(\psi(D)), [y] \leftarrow [\mathcal{M}](x)$.

4.2 *EncryIP*: Encryption

Integrating encryption and machine learning poses a crucial challenge: ensuring accurate communication between the two processes. Encryption and decryption demand exactness, while machine learning often deals with approximations, like probabilities. This discrepancy requires thoughtful solutions to bridge the gap and make encryption and learning methods work harmoniously. In this paper, we introduce a novel encryption strategy that presents an effective transfer function between label space and encryption scheme.

First of all, we show the details of the encryption scheme here. As described before, we consider a PKE with multiple secret keys corresponding to the same public key, and further require that there exist some ill-formed ciphertexts, such that decrypting them with different secret keys (corresponding to the same public key) will lead to different messages. Many PKE schemes, like those built using hash proof systems (Cramer and Shoup 2002), adhere to these characteristics. For practical purposes, we focus on a simplified form of the Cramer-Shoup scheme (Cramer and Shoup 1998), which we call $\mathsf{CS}_{\text{lite}}$.

Let $\mathbb{G}_q$ be a cyclic group of some prime order q, g_1 be a generator of $\mathbb{G}_q$, and $\mathbb{Z}_q^* = \mathbb{Z}_q \setminus \{0\}$. We present a PKE scheme $\mathsf{CS}_{\text{lite}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Fake})$, with message space $\mathbb{G}_q$, as shown in Figure 1. Note that here for a finite set S, we write $s \leftarrow S$ for sampling s uniformly random from S. Due to space limitations, the correctness analysis is shown in Appendix[1].

[1]We refer readers to the full version on the ArXiv website.

```
$\mathsf{Gen}(P)$:
$t \leftarrow \mathbb{Z}_q^*, g_2 = g_1^t, (a_1, b_1) \leftarrow (\mathbb{Z}_q)^2, h = g_1^{a_1} g_2^{b_1}$
$\{b_j \leftarrow \mathbb{Z}_q\}_{j=2}^P$ (s.t. $b_j \neq b_1$ and $b_{j_1} \neq b_{j_2}$ for $j_1 \neq j_2$)
$\{a_j = a_1 + (b_1 - b_j)t\}_{j=2}^P$
$pk = (g_1, g_2, h), \{sk_j = (a_j, b_j)\}_{j=1}^P$
Return $(pk, \{sk_j\}_{j=1}^P)$

$\mathsf{Enc}(pk, m)$:
$r \leftarrow \mathbb{Z}_q, u_1 = g_1^r, u_2 = g_2^r, u_3 = h^r m$
Return $c = (u_1, u_2, u_3)$

$\mathsf{Dec}(sk, c)$:
Parse $sk = (a, b)$ and $c = (u_1, u_2, u_3)$
Return $m = \frac{u_3}{u_1^a u_2^b}$

$\mathsf{Fake}(pk)$:
$(r_1, r_2) \leftarrow (\mathbb{Z}_q)^2$ (s.t. $r_1 \neq r_2$), $u_1 = g_1^{r_1}, u_2 = g_2^{r_2}, u_3 \leftarrow \mathbb{Z}_q^*$
Return $c = (u_1, u_2, u_3)$
```

Figure 1: PKE scheme $\mathsf{CS}_{\text{lite}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Fake})$.

For security of $\mathsf{CS}_{\text{lite}}$, we present the following theorem. Due to space limitations, the definition of the DDH assumption and that of IND-CPA security are recalled in Appendix, and the proof of Theorem 1 is given in Appendix.

Theorem 1 *If the DDH assumption holds for $\mathbb{G}_q$, $\mathsf{CS}_{\text{lite}}$ is IND-CPA secure.*

Based on $\mathsf{CS}_{\text{lite}}$ in Figure 1, we present our proposed encryption strategy, which encompasses three key steps:

1. Generate Public/Secret Keys: The process begins by utilizing the algorithm Gen within $\mathsf{CS}_{\text{lite}}$ to create public and secret key pairs.

2. Encrypt Labels: Working with the training set D and the algorithm Enc, the process involves processing the training set D as follows:

$$c \leftarrow \mathsf{Enc}(pk, y), \tag{1}$$

where ciphertext c is a three-tuple, i.e., (u_1, u_2, u_3).

It is important to note that the outcome of Enc is generally a random value. In our paper, we establish the first encryption result as the ciphertext for each label, and it remains unchanged subsequently. However, in reality, the outcome of $\mathsf{Enc}(pk, y)$ can be viewed as a randomly sampled value from a distribution denoted as $\mathsf{Dist}_y^{\text{ct}} = \{r \leftarrow \mathbb{Z}_q : (g_1^r, g_2^r, h^r y)\}$. This distribution is characterized by the following attributes:

- The distribution can be represented by any element sampled from it. In other words, given an element c' sampled from $\mathsf{Dist}_y^{\text{ct}}$, $\mathsf{Dist}_y^{\text{ct}}$ can be represented with c'.

That is because for any element c' sampled from $\mathsf{Dist}_y^{\text{ct}}$, c' can be written as $c' = (u_1', u_2', u_3') = (g_1^{r'}, g_2^{r'}, h^{r'} y)$. So $\mathsf{Dist}_y^{\text{ct}}$ can be written as

$$\mathsf{Dist}_y^{\text{ct}} = \{r \leftarrow \mathbb{Z}_q : (g_1^r u_1', g_2^r u_2', h^r u_3')\}.$$

Hence, (1) the ciphertext $c \leftarrow \mathsf{Enc}(pk, y)$ can be viewed as a representation of distribution; and (2) the following learning process can also be viewed as learning on distribution corresponding to labels, rather than a strict label value.

Additionally, we define a sampling algorithm $\mathsf{SampDist}$, which takes a cipehrtext $c' = (u_1', u_2', u_3')$ sampled from $\mathsf{Dist}_y^{\text{ct}}$ as input, and outputs a new ciphertext sampled from $\mathsf{Dist}_y^{\text{ct}}$. Formally, we write $c \leftarrow \mathsf{SampDist}(c')$ for this process. The detailed description of $\mathsf{SampDist}$ for the encryption algorithm Enc of $\mathsf{CS}_{\text{lite}}$ is as follows.

```
$\mathsf{SampDist}(c' = (u_1', u_2', u_3'))$:
  $r \leftarrow \mathbb{Z}_q;\ u_1 = g_1^r u_1';\ u_2 = g_2^r u_2';\ u_3 = h^r u_3'$
  Return $c = (u_1, u_2, u_3)$
```

3. Transfer ciphertexts to "confused" labels. As the structure of c is not a normal label structure, a learning algorithm $\mathcal{A}$ can not directly use it to train. In the ensuing section, we present a transfer function that establishes a connection between a conventional label structure and the ciphertext structure. We refer to every true label y as a "readable" label. For any label $[y]$ generated from a transfer function, we refer to it as a "confused" label.

Definition 2 (Transfer Function). *Denote Φ as the* transfer function *such that for each ciphertext c in the ciphertext structure, $\Phi(c) = [y]$. Denote Φ^{-1} as the inverse function of Φ satisfying $\Phi^{-1}(\Phi(c)) = c$ for all ciphertexts.*

In $\mathsf{CS}_{\text{lite}}$, the function Φ can be viewed as an indicator function $\mathbb{I}$ which outputs a q-dimension[2] all-zero vector except the position c is 1,

$$T \in \{0,1\}^q \leftarrow\ \Phi(c) = \mathbb{I}(c) \tag{2}$$

Specifically, as described in Eqn. (1), the ciphertext is a three-tuple (u_1, u_2, u_3). We take u_1, u_2 and u_3 separately as the input in Φ and get T_1, T_2 and T_3. Then, we concatenate them as $[y]$, i.e., $[y] = \{T_1 T_2 T_3\}$. Note that $[y]$ can be used in a normal learning process, but $[y]$ is a confused label that is totally different from y.

After that, we obtain a new dataset $[D] = \{x_i, [y_i]\}_{i=1}^N$. A case study is shown in Figure 2.

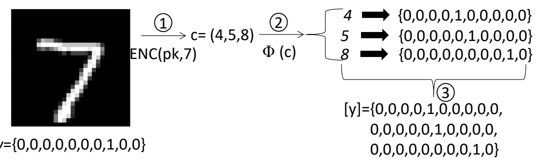


Figure 2: A case study on encryption process. (1) The index of label is encrypted by encryption algorithm Enc; (2) A ciphertext is transferred to label structure by transfer function Φ; (3) A confused label is achieved by concatenation.

Remark: (1) It is evident that by using alternative PKE schemes constructed from various hash proof systems (e.g., (Cramer and Shoup 2002; Hofheinz and Kiltz 2007; Naor and Segev 2009)) as foundational components, the ciphertext can be configured as a k-tuple (i.e., $c = (u_1, u_1, \dots, u_k)$). Generally, a higher value of k enhances the security of the PKE but results in reduced efficiency. For practicality and clarity, we adopt a simplified version with $k = 3$ in this paper. (2) It's worth noting that $[y]$ can also be calculated

[2] q is a parameter related to security of the encryption scheme. In our implementation, for simplicity, we set q to the smallest prime number greater than or equal to the number of classes. Further analysis on this parameter can be found in the experiment section.

Algorithm 1: Learning Process

```
Input  : $D = \{(x_i, y_i)\}_{i=1}^N$ - input data. $\mathcal{A}$ - algorithm
         architecture. $\mathsf{CS}_{\text{lite}}$ - encryption scheme.
Output : $[\mathcal{M}]$ - model, $\{sk_j\}_{j=1}^P$ - $P$ secret keys.
1 $(pk, \{sk_j\}_{j=1}^P) \leftarrow \mathsf{Gen}(P)$ ;
2 for $i = 1, \ldots, N$ do
3     $c_i \leftarrow \mathsf{Enc}(pk, y_i)$ ;
4     $[y_i] \leftarrow \Phi(c_i)$ ;
5 end
6 $[\mathcal{M}] \leftarrow \mathcal{A}(\{(x_i, [y_i])\}_{i=1}^N)$ ;
7 return $[\mathcal{M}], \{sk_j\}_{j=1}^P$.
```

using different methods, such as a reduction technique, e.g., $[y] = V_1 + V_2 + V_3$.

4.3 *EncryIP*: Learning /Deploying

Learning. Once the data is encrypted, the learning process is executed on the encrypted dataset $[D]$. The comprehensive learning process of *EncryIP* is delineated in Algorithm 1. Notably, the training procedure for the ML model remains unaltered from the original model, as illustrated in line 6 of Algorithm 1. The sole modification lies in replacing the original loss function with a soft-label loss within the learning algorithm $\mathcal{A}$. In Sec. 5.1, we demonstrate that this adjustment does not hinder the convergence of the learning process.

Deploying. The deployment process is formally outlined in Algorithm 2. In our framework, *EncryIP* produces a random prediction (confused label), thereby preventing attackers from inferring the true label through analysis of the confused label.

The deploying process unfolds as follows:

(1) Initially, an inverse function is utilized to convert confused labels into ciphertexts. For instance, assuming $[y]$ is an output vector denoting label probabilities, the inverse function Φ^{-1} operates as:

$$c \leftarrow \Phi^{-1}([y]) = I([y]) \tag{3}$$

Here, $I()$ returns the position index of the top k values in input $[y]$. For instance, when $k = 3$, $I()$ generates a ciphertext c from the position index of the top 3 values in prediction $[y]$, i.e., I_1, I_2, I_3 is treated as ciphertext c.

(2) As previously discussed, c can symbolize a distribution $\mathsf{Dist}y^{\text{ct}}$ linked to a certain y. In line 3 of Algorithm 2, a random ciphertext c_d is derived by sampling from $\mathsf{Dist}_y^{\text{ct}}$.

(3) Ultimately, *EncryIP* generates a random prediction (a confused label), as depicted in lines 4 to 5 of Algorithm 2.

An authorized user holding the correct secret key sk_j ($j \in 1, \cdots, P$) can obtain the actual label:

$$y \leftarrow \mathsf{Dec}(sk_j, c_d). \tag{4}$$

4.4 *EncryIP*: Verification

In this section, we describe the methodology for identifying unauthorized usage of the model. We assume a scenario where a neutral arbitrator, who possesses all secret keys $S = \{sk_j\}_{j=1}^P$, sells the ML model to P users, each possessing a distinct secret key. The verification procedure is carried out by the arbitrator. We consider the two following scenarios:

Algorithm 2: Deploying Process

```
Input  : $x$ - input dataset. $[\mathcal{M}]$ - model,
         $sk_j, j \in \{1, 2, \ldots, P\}$ - a secret key.
Output : $[y_d]$ - a confused label.
1 $[y] \leftarrow [\mathcal{M}](x)$ ;
2 $c \leftarrow \Phi^{-1}([y])$ ;
3 $c_d \leftarrow \mathsf{SampDist}(c)$ ;
4 $[y_d] \leftarrow \Phi(c_d)$ ;
5 return $[y_d]$.
```

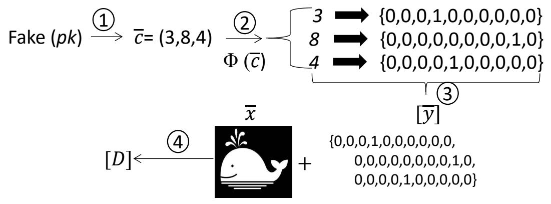


Figure 3: A case study on verification (when the secret key is unavailable). (1) One kind of ill-formed ciphertext $\bar{c}$ is generated by algorithm Fake; (2) A ciphertext is transferred to label structure by transfer function Φ; (3) A confused label is achieved by concatenation; (4) A verification instance-label pair $\{\bar{x}, [\bar{y}]\}$ is added to $[D]$.

• **Secret key available.** In this scenario, the arbitrator can know which secret key is used by a user in the deploying process. This is a simple and easy verification scenario that the neutral arbitrator can check if the secret key belongs to a corresponding model user by:

$$V([\mathcal{M}], sk) = \begin{cases} j, & \text{if } sk \in S, sk = sk_j \\ \text{False}, & \text{if } sk \notin S \end{cases}$$

• **Secret key unavailable.** In this scenario, the arbitrator lacks knowledge about which secret key a user employs during the deployment. Detecting unauthorized model usage without this information poses a challenge. To address this challenge, we leverage the property of $\mathsf{CS}_{\text{lite}}$, where ill-formed ciphertexts yield distinct decrypted messages based on different secret keys. This property can be applied to *EncryIP* as follows:

Let $\{\bar{x}, [\bar{y}]\}$ be a verification instance-label pair. $\bar{x}$ is an arbitrary instance (e.g., an arbitrary picture), and $\bar{y}$ is generated from one kind of ill-formed ciphertext $\bar{c}$ (i.e., $[\bar{y}] \leftarrow \mathbb{I}(\bar{c})$), where $\bar{c} \leftarrow \mathsf{Fake}(pk)$. We assume $\{\bar{y}'_j\}_{j=1}^P$ is a set of results which is from decrypting the ciphertext $\bar{c}$ by using different secret keys, i.e., $\bar{y}'_j \leftarrow \mathsf{Dec}(sk_j, \bar{c})$ for $j \in \{1, \ldots, P\}$. Note that the arbitrator has this information $\{\bar{x}, \{sk_j, \bar{y}'_j\}_{j=1}^P\}$. Then we add $\{\bar{x}, [\bar{y}]\}$ to $[D]$, and train the ML model $[\mathcal{M}]$. A simple case is illustrated in Figure 3.

When $\bar{x}$ is utilized as input and distinct secret keys are employed to decrypt the ciphertext $\bar{c}$, $[\mathcal{M}]$ will yield varying results due to the aforementioned property. Consequently, the verification process is as follows:

$$V([\mathcal{M}], \bar{x}) = \begin{cases} j, & \text{if } \mathsf{Dec}(sk_j, \bar{c}) = \bar{y}'_j. \\ \text{False}, & \begin{array}{l}\text{if } \forall j \in \{1, \cdots, P\}, \\ \quad \mathsf{Dec}(sk_j, \bar{c}) \neq \bar{y}'_j.\end{array} \end{cases} \tag{5}$$

where $\bar{c} \leftarrow I([\mathcal{M}](\bar{x}))$.

4.5 *EncryIP*: Analysis

(1) Against removal attacks and ambiguity attacks. Removal attacks seek to eliminate or alter IP protection measures by modifying model weights through techniques like fine-tuning or pruning. Since *EncryIP* is data-dependent, its performance remains unaffected regardless of the extent of fine-tuning or pruning applied. This robustness makes *EncryIP* resilient against removal attacks.

Ambiguity attacks aim to forge counterfeit secret keys (or watermark, or passport) without modifying model weights. In *EncryIP*, the security of $\mathsf{CS}_{\text{lite}}$ guarantees that it is hard to forge a secret key. More specifically, according to Theorem 1, $\mathsf{CS}_{\text{lite}}$ is IND-CPA secure, which implies that the probability of forging a secret key successfully is negligible (otherwise, one can trivially succeed in breaking the IND-CPA security of $\mathsf{CS}_{\text{lite}}$, contradicting Theorem 1).

(2) The incremental authorized users. In practice, a significant challenge for IP protection methods lies in efficiently accommodating an increasing number of authorized users. As we know, *EncryIP* produces different model versions by generating different secret keys. This challenge can be easily addressed in *EncryIP*. Specifically, assume that P secret keys $\{sk_j = (a_j, b_j)\}_{j=1}^{P}$ have been generated, and now a new user is authorized. In this case, a new secret key $sk_{P+1} = (a_{P+1}, b_{P+1})$ can be generated as follows: sampling $b_{P+1} \leftarrow \mathbb{Z}_q$ such that $b_{P+1} \notin \{b_1, \cdots, b_P\}$, and computing $a_{P+1} = a_1 + (b_1 - b_{P+1})t$. Note that, this approach eliminates the need to update existing authorized users' secret keys or retrain the ML model, serving as a future direction.

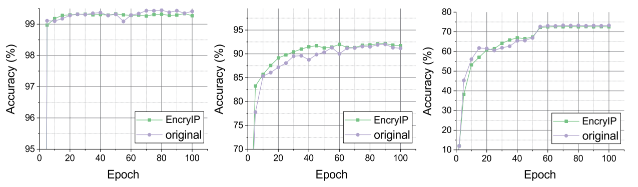
(3) The collusion. The aim of this study is to systematically define and address the issue of unauthorized model distribution within the context of model IP protection. We present a robust solution under certain initial assumptions, which offers room for further enhancement. For instance, our proposed method, *EncryIP*, currently operates on the premise that authorized users do not collaborate. Exploring the substitution of the employed PKE scheme in Sec. 4.2 with alternatives rooted in distinct computational assumptions, such as the Decisional Composite Residuosity assumption (Paillier 1999), could yield advancements. We regard the exploration of ways to fortify *EncryIP* from diverse perspectives as a promising avenue for future research.

5 Experiment

5.1 The effectiveness of *EncryIP*

The results on Learning /Deploying. We evaluate the effectiveness of *EncryIP* during both the learning and deployment stages. We compare the test performance of *EncryIP* and *EncryIP*$_{\text{Incorrect}}$ (using an incorrect secret key) with the original model. These comparisons are performed across three machine learning model structures and three datasets. We repeat each experiment 30 times and present the results in terms of mean and standard deviation.

The prediction accuracy is employed as the evaluation metric. In *EncryIP*, we set q to the number of classes in each data set. The common parameters in each model structure are set by default values and the same in each method. The settings of *EncryIP*$_{\text{Incorrect}}$ are the same with *EncryIP*, except a fake secret key is used.



(a) MNIST (b) CIFAR10 (c) CIFAR100

Figure 4: The training convergence for ResNet. Results for GoogLeNet and AlexNet can be found in the Appendix.

Table 1 shows that the results of *EncryIP* are almost the same as the performance of the original model. The displayed Δ values indicate percentage deviations between the original model and *EncryIP*, with an approximate average of 1%. Conversely, *EncryIP*$_{\text{Incorrect}}$ exhibits subpar performance, confirming that incorrect secret keys hinder recovery in *EncryIP*. These findings underscore *EncryIP*'s efficacy as an impactful protection for model outputs while maintaining strong model performance.

Figure 4 demonstrates that the incorporation of *EncryIP* does not impede the convergence of the learning process. The accuracy convergence of *EncryIP* aligns closely with the original model's trajectory, exhibiting nuanced differences during training. The dissimilarity between training with *EncryIP* and the original approach is primarily attributed to label structure. *EncryIP* introduces confused labels to the training model, potentially influencing the training process. However, as indicated in Table 1, this modification has only a minimal effect on performance, highlighting the effective balance of security and performance achieved by *EncryIP*.

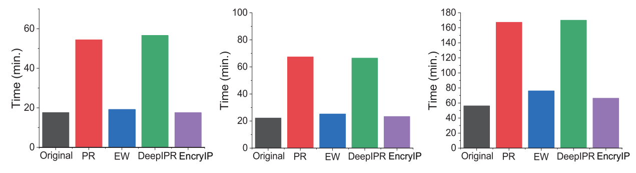
The results on verification. To examine the verification of *EncryIP* (the performance of detecting whose model is unauthorizedly used), we conducted experiments as below: The verification is under the scenario of secret key unavailable. One of the authorized user's models (e.g., j) is randomly selected as the model which is used by an unintended user. We check if the output of Eqn. (5) is equal to j. Note that in our experiments, the required image $\bar{x}$ in verification can be set by different types. We refer $\bar{x}$ to one kind of randomly selected image which is different from training data. The experiment is repeated 30 times, we try different types of $\bar{x}$ and different j each time. In Table 1, the performance of verification is shown in parentheses. The results of the verification are at nearly 100% accuracy in different structures and different datasets.

5.2 The efficiency of *EncryIP*

Settings. We compare training times among state-of-the-art methods to highlight their efficiency in addressing the problem of the unauthorized spread of the model. The three typical methods from watermark-based (**PR** (Uchida et al. 2017)), trigger-based (**EW** (Zhang et al. 2018)) and passport-based method (**DeepIPR** (Fan et al. 2022)) are chosen as the comparisons. The scenario involves a model producer intending

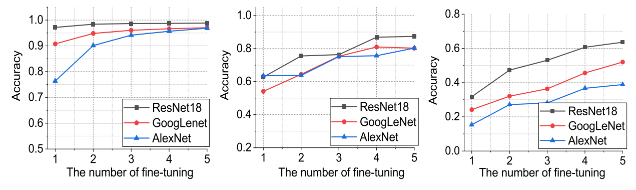
Table 1: The results on the effectiveness (Acc.). In the parentheses, the left side indicates the percentage changes between the original model and *EncryIP*, and the right side indicates the performance of the verification.

Model	Method	MNIST	CIFAR10	CIFAR100	ImageNet
ResNet18	original	0.9936 ± 0.0014	0.9030 ± 0.0046	0.7322 ± 0.0036	0.6231 ± 0.0010
	EncryIP	0.9930 ± 0.0006 (0.06%, 1.0)	0.9094 ± 0.0051 (0.71%, 1.0)	0.7279 ± 0.0030 (0.59%, 0.97)	0.6178 ± 0.0006 (0.85%, 1.0)
	EncryIP$_{\text{Incorrect}}$	0.0006 ± 0.0003	0.0075 ± 0.0053	0.0027 ± 0.0008	0.0011 ± 0.0005
GoogLeNet	original	0.9905 ± 0.0011	0.8675 ± 0.0036	0.6347 ± 0.0031	0.6502 ± 0.0016
	EncryIP	0.9879 ± 0.0015 (0.84%, 1.0)	0.8508 ± 0.0101 (1.9%, 1.0)	0.6165 ± 0.0053 (2.86%, 1.0)	0.6461 ± 0.0013 (0.63%, 1.0)
	EncryIP$_{\text{Incorrect}}$	0.0216 ± 0.0403	0.0124 ± 0.0087	0.0055 ± 0.0013	0.024 ± 0.0102
AlexNet	original	0.9914 ± 0.0006	0.8563 ± 0.0022	0.5439 ± 0.0049	0.5673 ± 0.0020
	EncryIP	0.9923 ± 0.0010 (0.09%, 1.0)	0.8457 ± 0.0031 (1.24%, 1.0)	0.5282 ± 0.0040 (2.88%, 0.97)	0.5615 ± 0.0020 (1.02%, 1.0)
	EncryIP$_{\text{Incorrect}}$	0.0006 ± 0.0003	0.0181 ± 0.0048	0.0055 ± 0.0012	0.0006 ± 0.0025



(a) MNIST (b) CIFAR10 (c) CIFAR100

Figure 5: The training time results of ResNet18.



(a) MNIST (b) CIFAR10 (c) CIFAR100

Figure 6: The results of removal attacks.

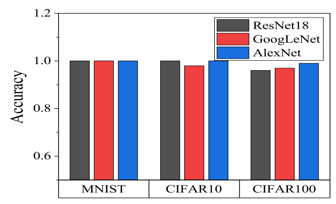


Figure 7: The verification on removal attacks.

to sell an ML model to three users. For PR, the producer must perform three separate training sessions, each yielding a distinct model version. This necessity arises from the use of distinct watermark parameters required by the embedding regularizer for generating different versions. Similarly, DeepIPR necessitates three separate training rounds to cater to its requirement for employing three distinct passport groups to generate diverse model versions. Although EW can train once to address this situation by incorporating three different trigger sets, it fails to meet the security requirement as its output remains interpretable. Note that the experimental conditions are the same in each group, e.g., the number of epochs. The outcomes for ResNet18 are depicted in Figure 5, while comprehensive results are available in Appendix.

Results. The results show that *EncryIP* is more efficient than the other methods (PR and DeepIPR), and its training time is almost close to the original time (the "Original" in Figure 5). EW performs similar results with *EncryIP* because it just needs one time of training to meet this scenario. But, this approach does not provide sufficient security for the resulting predictions, as its output is a readable prediction. In contrast, our proposed method prioritizes the output of confused labels, which offers a more protected solution than EW.

5.3 Attack analysis

To evaluate removal attacks, we simulate an experiment involving the influence of fine-tuning. We partition each dataset into five parts and consecutively fine-tune the model on each part. We use the pre-trained model from the previous part as a starting point for the current training. The training procedure follows the Algorithm 1. Figure 6 illustrates the trend in model performance after each fine-tuning round. The outcomes across the three datasets reveal that fine-tuning does not impact *EncryIP*, and the model performance trend remains consistent within the expected range. To examine the verification, we conduct tests on its performance after each round of fine-tuning, employing the same experimental setup as described in Sec. 5.1. Figure 7 presents the verification performance, which remains consistently close to 100% accuracy across different model structures and datasets. These results demonstrate the robustness of *EncryIP* against this type of attack as well.

6 Conclusion

This paper presents an innovative encryption-based framework for protecting model intellectual property. The framework employs label space encryption, establishing autonomy between learning and encryption algorithms. *EncryIP* effectively ensures IP protection while maintaining efficiency across various model structures. Experimental results validate its effectiveness. Future work will focus on enhancing the framework's efficiency and security, exploring its theoretical underpinnings, and extending the concept to address dynamic IP protection challenges.

7 Acknowledgments

This work was supported by the National Key Research and Development Program of China (No. 2022ZD0115301), the National Natural Science Foundation of China (62106114, 62206139), the Major Key Project of PCL, National Natural Science Foundation of China under Grant No. U2001205, Guangdong Basic and Applied Basic Research Foundation (Grant No. 2023B1515040020),

References

Adi, Y.; Baum, C.; Cissé, M.; Pinkas, B.; and Keshet, J. 2018. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *USENIX Security Symposium*, 1615–1631.

Boenisch, F. 2020. A Survey on Model Watermarking Neural Networks. *CoRR*, abs/2009.12153.

Cao, X.; Jia, J.; and Gong, N. Z. 2021. IPGuard: Protecting Intellectual Property of Deep Neural Networks via Fingerprinting the Classification Boundary. In *ASIA CCS*, 14–25.

Chen, H.; Rouhani, B. D.; Fan, X.; Kilinc, O. C.; and Koushanfar, F. 2018. Performance Comparison of Contemporary DNN Watermarking Techniques. *CoRR*, abs/1811.03713.

Chen, H.; Rouhani, B. D.; Fu, C.; Zhao, J.; and Koushanfar, F. 2019. DeepMarks: A Secure Fingerprinting Framework for Digital Rights Management of Deep Learning Models. In *ICMR*, 105–113.

Chen, H.; Rouhani, B. D.; and Koushanfar, F. 2019. BlackMarks: Blackbox Multibit Watermarking for Deep Neural Networks. *CoRR*, abs/1904.00344.

Chen, X.; Liu, C.; Li, B.; Lu, K.; and Song, D. 2017. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *CoRR*, abs/1712.05526.

Cramer, R.; and Shoup, V. 1998. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, 13–25.

Cramer, R.; and Shoup, V. 2002. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In *EUROCRYPT*, 45–64.

Fan, L.; Ng, K. W.; and Chan, C. S. 2019. Rethinking Deep Neural Network Ownership Verification: Embedding Passports to Defeat Ambiguity Attacks. In *NeurIPS*, 4716–4725.

Fan, L.; Ng, K. W.; Chan, C. S.; and Yang, Q. 2022. DeepIPR: Deep Neural Network Ownership Verification With Passports. *IEEE TPAMI*, 44(10): 6122–6139.

Feng, L.; and Zhang, X. 2020. Watermarking Neural Network with Compensation Mechanism. In *KSEM*, 363–375.

Guo, J.; and Potkonjak, M. 2018. Watermarking deep neural networks for embedded systems. In *ICCAD*, 133–139.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*, 770–778.

Hofheinz, D.; and Kiltz, E. 2007. Secure Hybrid Encryption from Weakened Key Encapsulation. In *CRYPTO*, 553–571.

Jebreel, N. M.; Domingo-Ferrer, J.; Sánchez, D.; and Blanco-Justicia, A. 2021. KeyNet: An Asymmetric Key-Style Framework for Watermarking Deep Learning Models. *Applied Sciences*, 11(3).

Katz, J.; and Lindell, Y. 2020. *Introduction to modern cryptography*. CRC press.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 1106–1114.

Kuribayashi, M.; Tanaka, T.; and Funabiki, N. 2020. DeepWatermark: Embedding Watermark into DNN Model. In *APSIPA*, 1340–1346.

Lukas, N.; Zhang, Y.; and Kerschbaum, F. 2021. Deep Neural Network Fingerprinting by Conferrable Adversarial Examples. In *ICLR*.

Maung, A. P. M.; and Kiya, H. 2020. Training DNN Model with Secret Key for Model Protection. In *IEEE GCCE*, 818–821.

Merrer, E. L.; Pérez, P.; and Trédan, G. 2020. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing & Applications*, 32(13): 9233–9244.

Naor, M.; and Segev, G. 2009. Public-Key Cryptosystems Resilient to Key Leakage. In *CRYPTO*, 18–35.

OpenAI. 2023. GPT-4 Technical Report. *CoRR*, abs/2303.08774.

Paillier, P. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, 223–238. ISBN 978-3-540-48910-8.

Ramesh, A.; Pavlov, M.; Goh, G.; Gray, S.; Voss, C.; Radford, A.; Chen, M.; and Sutskever, I. 2021. Zero-Shot Text-to-Image Generation. In *ICML*.

Rouhani, B. D.; Chen, H.; and Koushanfar, F. 2019. DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks. In *ASPLOS*, 485–497.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S. E.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*, 1–9.

Tauhid, A.; Xu, L.; Rahman, M.; and Tomai, E. 2023. A survey on security analysis of machine learning-oriented hardware and software intellectual property. *High-Confidence Computing*, 3(2): 100114.

Uchida, Y.; Nagai, Y.; Sakazawa, S.; and Satoh, S. 2017. Embedding Watermarks into Deep Neural Networks. In *ICMR*, 269–277.

Xue, M.; Wang, J.; and Liu, W. 2021. DNN Intellectual Property Protection: Taxonomy, Attacks and Evaluations. In *Great Lakes Symposium on VLSI*, 455–460.

Zhang, J.; Chen, D.; Liao, J.; Fang, H.; Zhang, W.; Zhou, W.; Cui, H.; and Yu, N. 2020a. Model Watermarking for Image Processing Networks. In *AAAI*, 12805–12812.

Zhang, J.; Chen, D.; Liao, J.; Zhang, W.; Feng, H.; Hua, G.; and Yu, N. 2022. Deep Model Intellectual Property Protection via Deep Watermarking. *IEEE TPAMI*, 44(8): 4005–4020.

Zhang, J.; Chen, D.; Liao, J.; Zhang, W.; Hua, G.; and Yu, N. 2020b. Passport-aware Normalization for Deep Model Protection. In *NeurIPS*.

Zhang, J.; Gu, Z.; Jang, J.; Wu, H.; Stoecklin, M. P.; Huang, H.; and Molloy, I. M. 2018. Protecting Intellectual Property of Deep Neural Networks with Watermarking. In *ASIA CCS*, 159–172.

Zhong, Q.; Zhang, L. Y.; Zhang, J.; Gao, L.; and Xiang, Y. 2020. Protecting IP of Deep Neural Networks with Watermarking: A New Label Helps. In *PAKDD*, 462–474.