Supplementary Material: Protecting Intellectual Property of Generative Adversarial Networks from Ambiguity Attacks

Ding Sheng Ong[1] Chee Seng Chan[1*] Kam Woh Ng[2] Lixin Fan[2] Qiang Yang[3]

[1] University of Malaya [2] WeBank AI Lab [3] Hong Kong University of Science and Technology

Abstract

This supplementary material provides a more complete explanation and experiment results that could not be included in the main paper due to space constraints.

A. Overview of the Verification Process in Model Protection

Generally, the verification process as shown in Fig. 1, a suspicious online model will be first remotely queried through API calls using a specific input keys (*e.g.* trigger set) that were initially selected to trigger the watermark information. As such, this is a black-box verification where a final model prediction (*e.g.* for CNN model, the image classification results) is obtained. This initial step is usually perform to collect evidence from everywhere so that an owner can sue a suspected party who used (*i.e.* infringed) his/her models illegally. Once the owner has sufficient evidence, a second verification process which is to extract watermark from the suspected model and compare if the watermark is from the owner. This process is a white-box verification, which means the owner needs to have to access the model physically, and usually this second step is go through law enforcement.

B. Extension to other generative models - Variational Autoencoder

As mentioned in the main paper, Section 3, line 246, our proposed framework can be easily extended to other deep generative models with trivial modification. Here we show an example with Variational Autoencoder (VAE). In general, VAE consists of 2 parts which are the probabilistic encoder, $q_\psi(\boldsymbol{z}|\boldsymbol{x})$ and the generative model, $p_\theta(\boldsymbol{x}, \boldsymbol{z})$. The Encoder approximate the posterior in the form of multivariate Gaussian. The loss of a VAE is given below:

$$\mathcal{L}_{\text{VAE}} = \frac{1}{2}\left(1 + \log(\boldsymbol{\sigma^2}) - \boldsymbol{\mu}^2 - \boldsymbol{\sigma}^2\right) + \log(p_\theta(\boldsymbol{x}|\boldsymbol{z})) \quad (1)$$

*Corresponding author, e-mail: cs.chan@um.edu.my

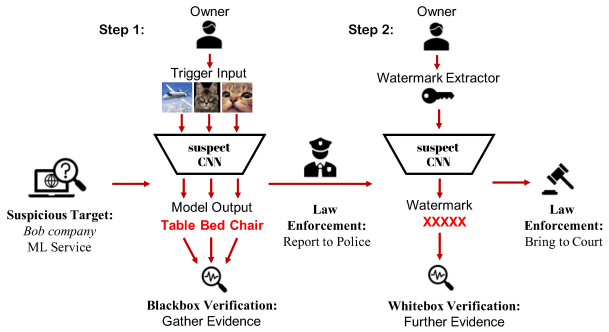


Figure 1: An overview of general watermarking scheme 2-steps verification process. In Step 1, it can be noticed that the protected model is trained to deliberately output specific (incorrect) labels for a particular set of inputs T that is known as the "trigger set". In Step 2, the watermark is extracted from the model to proof ownership.

where $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$. The input to the generative model is the posterior, $\boldsymbol{z}$ which is either from the probabilistic encoder or sampled from a multivariate Gaussian distribution, $\mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma I})$.

Since $\boldsymbol{z}$ is a vector of n dimension and the output is an image (*i.e.* similar to DCGAN), we can use the exact same transformation functions defined in Section 3.1.1 (DCGAN) in the main paper, to create our trigger input using Eq. 1 and the target using Eq. 2, respectively. Thus, the overall new objective is shown as:

$$\mathcal{L}_{\text{VAE}_w} = \mathcal{L}_{\text{VAE}} + \lambda \mathcal{L}_w \quad (2)$$

B.1. Experimental results

For the VAE experiment, we set $\lambda = 0.1$ unlike other GAN experiments in the main paper as the KLD loss and the reconstruction loss are in very small scale. We perform the experiment on the public dataset - CIFAR10, and the results are shown in Table 1.

	FID	Q_{wm}
VAE	229.6874 ± 3.80	-
VAE_w	226.8893 ± 0.86	0.9973 ± 0.014
VAE_{ws}	231.1154 ± 0.63	0.9964 ± 0.014

Table 1: VAE results on CIFAR10.

$z \in \mathbb{R}^{128}$ $\mathcal{N}(0,1)$
dense $\rightarrow M_g \times M_g \times 512$
4×4, stride=2 deconv. BN 256 ReLU
4×4, stride=2 deconv. BN 128 ReLU
4×4, stride=2 deconv. BN 64 ReLU
3×3, stride=2 conv. 3 Tanh

Table 2: Generator, M_g = 4 for CIFAR10 and M_g = 8 for CUB-200.

RGB image $x \in \mathbb{R}^{M \times M \times 3}$
3×3, stride=1 conv. 64 LeakyReLU
4×4, stride=2 conv. 64 LeakyReLU
3×3, stride=1 conv. 128 LeakyReLU
4×4, stride=2 conv. 128 LeakyReLU
3×3, stride=1 conv. 256 LeakyReLU
4×4, stride=2 conv. 256 LeakyReLU
3×3, stride=1 conv. 512 LeakyReLU
dense $\rightarrow 1$

Table 3: Discriminator, M = 32 for CIFAR10 and M = 64 for CUB-200.

C. Network Architecture

C.1. DCGAN

Table 2 - 3 show the standard CNN models for CIFAR-10 and CUB-200 used in our experiments on image generation (*i.e*. DCGAN). The slopes of all LeakyReLU functions in the networks are set to 0.1.

C.2. CycleGAN

Table 4 - 5 show the architecture for CycleGAN.

D. Network Complexity

Table 6 shows the computational complexity as a result of the additional of our regularization terms on GANs model. We observe that adding a new regularization term to embed watermark and signature has no effect to the inference time. As for training time, it is expected that it has an impact on training time but the effect is very minor. We believe that it is the computational cost at the inference stage that is required to be minimized, since network inference is going to be performed frequently by the end users. While extra costs at the training stage, on the other hand, are not prohibitive since they are performed by the network owners, with the motivation to protect the model ownerships.

RGB Image				$128 \times 128 \times 3$
Conv.IN.ReLU	7×7	stride=1	padding=3	$128 \times 128 \times 64$
Conv.IN.ReLU	3×3	stride=2	padding=1	$64 \times 64 \times 128$
Conv.IN.ReLU	3×3	stride=2	padding=1	$32 \times 32 \times 256$
ResidualBlock	-	-	-	$32 \times 32 \times 256$
ResidualBlock	-	-	-	$32 \times 32 \times 256$
ResidualBlock	-	-	-	$32 \times 32 \times 256$
ResidualBlock	-	-	-	$32 \times 32 \times 256$
ResidualBlock	-	-	-	$32 \times 32 \times 256$
ResidualBlock	-	-	-	$32 \times 32 \times 256$
Deconv.In.ReLU	3×3	stride=2	padding=1	$64 \times 64 \times 128$
Deconv.In.ReLU	3×3	stride=2	padding=1	$128 \times 128 \times 64$
Conv.Tanh	7×7	stride=1	padding=3	$128 \times 128 \times 3$

Table 4: ResNet Generator architecture of CycleGAN. Reflection Padding was used and all normalization layers are Instance Normalization according to the author's work.

RGB Image				$128 \times 128 \times 3$
Conv.lReLU	4×4	stride=2	padding=1	$64 \times 64 \times 64$
Conv.IN.lReLU	4×4	stride=2	padding=1	$32 \times 32 \times 128$
Conv.IN.lReLU	4×4	stride=2	padding=1	$16 \times 16 \times 256$
Conv.IN.lReLU	4×4	stride=2	padding=1	$8 \times 8 \times 512$
Conv	4×4	stride=2	padding=1	$4 \times 4 \times 1$

Table 5: 70×70 PatchGAN [1] was used as Discriminator of CycleGAN. Leaky ReLU with slope of 0.2 was used except the last layer.

	Relative Time
DCGAN	1.00
$DCGAN_w$	1.25
$DCGAN_{ws}$	1.26
SRGAN	1.00
$SRGAN_w$	1.19
$SRGAN_{ws}$	1.23
CycleGAN	1.00
$CycleGAN_w$	1.15
$CycleGAN_{ws}$	1.17

Table 6: The impact of the framework to the training time. The values in the table are relative to the baseline model.

E. Extended Results

E.1. DCGAN

Fig. 2 is the extended results of the original task of DCGAN (*i.e*. image synthesis), as well as three different types of watermark logos with CIFAR10 dataset when a trigger input is provided, while Fig. 3 illustrates the results with

Figure 2: CIFAR10: First row is the sample watermark logo. Second row is the images generated by DCGAN $G(z)$ (*i.e.* original task) and the last row shows the watermarked images if a trigger input is provided to the protected DCGAN model where each of them is a different protected model trained on different response output set.

Figure 3: CUB-200: First row is the images generated by DCGAN $G(z)$ (*i.e.* original task) and the second row shows the watermarked images if a trigger input is provided to the protected DCGAN model where each of them is a different protected model trained on different response output set.

CUB-200 dataset.

E.2. SRGAN

For SRGAN, the extended results are shown in Fig. 4.

E.3. CycleGAN

For CycleGAN, the extended results are shown in Fig. 5.

E.4. Robustness against removal attack

Fine-tuning. Fig. 6 shows the qualitative results for Section 4.5, line 698 that our proposed method is robust against fine-tuning attack as highlighted in Table 7 of the main manuscript. We can clearly see that the watermark (on the top left corner) remains intact after fine-tuning.

E.5. Resilience against ambiguity attack

This section shows the full results of applying sign loss (Eq. 11 in the main paper) to embed a signature into BN-scale, γ^{BN}. The implementation details is given in Section 3.2 of the main paper. Here in the supplementary, we show the example of how the unique key "EXAMPLE" is embedded into our DCGAN's batch normalization weight. Table 7 shows how to decode the trained scale, γ^{BN} to retrieve the signature embedded. Also, please note that even that there are 2 "E", their γ^{BN} are different from each other.

Fig. 7 - 8 are the complete results to complement Figure 9 - 10 in Section 4.6 of the main manuscript. It can be clearly visualize from Fig. 8 that the quality of the generated SR-images is very poor where obvious artefact can be observed even the signature signs are modified at only 10%. With this, we can deduce that the scale signs (Eq. 11 in the main manuscript) enforced in this way remain rather persistent against ambiguity attacks.

E.6. Ablation Study - Coefficient λ.

In this section, qualitatively, we illustrate in Fig. 9 the effects of different λ settings (*i.e.* from $0.1 \rightarrow 10$) on the original GAN model performance against the quality of the generated watermark (this is similar to Table 10 in the main manuscript) with CIFAR10 dataset. From visual inspection on Fig. 9, it is hard to deduce that which λ is an ideal choice. In this paper, based on the quantitative results (FID vs. SSIM) in Table 10 in the main manuscript, we set $\lambda = 1.0$.

E.7. Ablation Study - n vs. c.

In this section, qualitatively, we illustrate in Fig. 10 - 11 different n and c settings to understand the tradeoffs between the original GAN model performance against the quality of the generated watermark (this is similar to Table 11 in the main manuscript) with CIFAR10 dataset. From here, it can be noticed that it is, however, very hard to distinguish from a naked eye point of view which setting is having the best tradeoffs with the exception that it is very clear that setting $c = 0$ is not ideal. This is because from Fig. 11a-c, we

Figure 4: The first column shows the trigger input $\boldsymbol{x}_\omega$ to SRGAN. Next three columns are the response output $G(\boldsymbol{x}_\omega)$ when the trigger input is provided to the protected generators .

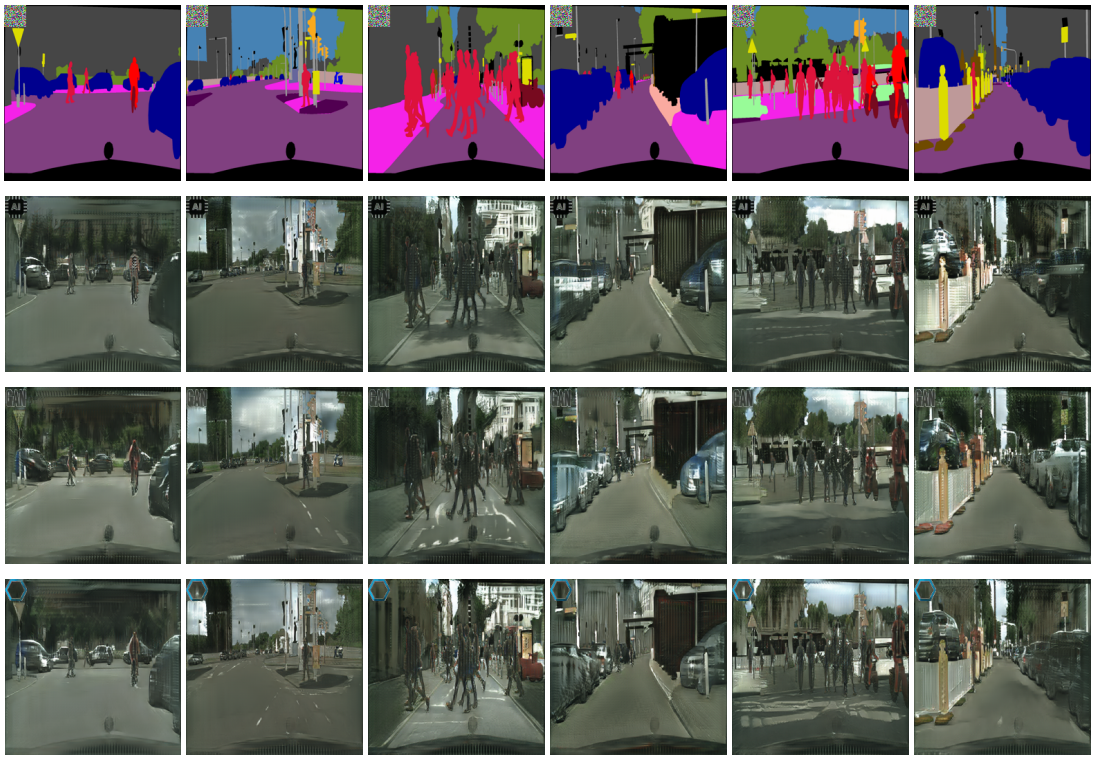


Figure 5: Image pairs from CycleGAN models trained on Cityscapes datasets, respectively

can notice that the generated images are all almost black (*i.e.* for $n = 5, 10, 15$). This phenomenon happens because the training input of DCGAN has a normal distribution of $\mu = 0$, therefore it is conflicting with the trigger input which is also set as $c = 0$.

As a summary, trigger input set must have a very different distribution from the training data. In this paper, based on the quantitative results (FID vs. SSIM) reported in Table 7

E			X			A			M			P			L			E		
γ	+/-	bit	γ	+/-	bit	γ	+/-	bit	γ	+/-	bit	γ	+/-	bit	γ	+/-	bit	γ	+/-	bit
-0.50	-	0	-0.22	-	0	-0.49	-	0	-0.24	-	0	-0.17	-	0	-0.44	-	0	-0.23	-	0
0.46	+	1	0.40	+	1	0.39	+	1	0.39	+	1	0.56	+	1	0.52	+	1	0.52	+	1
-0.42	-	0	-0.26	-	0	-0.44	-	0	-0.19	-	0	-0.17	-	0	-0.48	-	0	-0.28	-	0
-0.64	-	0	0.54	+	1	-0.17	-	0	-0.36	-	0	0.65	+	1	-0.62	-	0	-0.43	-	0
-0.25	-	0	0.43	+	1	-0.15	-	0	0.58	+	1	-0.53	-	0	0.37	+	1	-0.51	-	0
0.25	+	1	-0.14	-	0	-0.52	-	0	0.24	+	1	-0.56	-	0	0.49	+	1	0.22	+	1
-0.61	-	0	-0.45	-	0	-0.44	-	0	-0.18	-	0	-0.20	-	0	-0.47	-	0	-0.26	-	0
0.57	+	1	-0.34	-	0	0.35	+	1	0.55	+	1	-0.40	-	0	-0.55	-	0	0.32	+	1

Table 7: Example of the trained batch normalization weight $\boldsymbol{\gamma}$ of DCGAN$_{ws}$ using the word "EXAMPLE" as an unique key. We use 8-bits to represent each character.

Generator	Loss	Input	Black-Box			White-Box		Overall Loss
			Trigger	Target	Loss	Norm Type	Loss	
DCGAN	$\mathcal{L}_{\text{DC}}$ [Eq. 5]	$\boldsymbol{z} \sim \mathcal{N}(0,1)$	$f(\boldsymbol{z})$ [Eq. 1]	$g(G(\boldsymbol{z}))$ [Eq. 2]	$\mathcal{L}_w$ [Eq. 3]	BatchNorm	$\mathcal{L}_s$ [Eq. 11]	$\mathcal{L}_{\text{DC}} + \lambda\mathcal{L}_w + \mathcal{L}_s$
SRGAN	$\mathcal{L}_{\text{SR}}$ [Eq. 8]	$\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})$	$h(\boldsymbol{x})$ [Eq. 6]	$g(G(\boldsymbol{x}))$ [Eq. 2]	$\mathcal{L}_w$ [Eq. 3]	BatchNorm	$\mathcal{L}_s$ [Eq. 11]	$\mathcal{L}_{\text{SR}} + \lambda\mathcal{L}_w + \mathcal{L}_s$
CycleGAN	$\mathcal{L}_{\text{C}}$ [Eq. 10]	$\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})$	$h(\boldsymbol{x})$ [Eq. 6]	$g(G(\boldsymbol{x}))$ [Eq. 2]	$\mathcal{L}_w$ [Eq. 3]	InstanceNorm	$\mathcal{L}_s$ [Eq. 11]	$\mathcal{L}_{\text{C}} + \lambda\mathcal{L}_w + \mathcal{L}_s$

Table 8: Summary of our proposed implementation to protect the IPR of GANs models. Note that, the equations herein are reflected in the main paper.

of the main manuscript, we conclude that setting $n = 5$ and $c = -10$ is the most ideal.

F. Summary

In this paper, we represent an outline of a complete and robust ownership verification scheme for GANs covering the black-box and white-box protection schemes in Table 8. The importance of this work, in our view, can be highlighted by numerous disputes over IP infringements between giant and/or startup companies, which are now heavily investing substantial resources on developing new DNN models. It is our wish that the ownership verification for GANs, together with existing efforts to protect CNN models provide technical solutions in discouraging plagiarism and, hence, reducing wasteful lawsuit cases.

(a) DCGAN

(b) SRGAN

(c) CycleGAN

Figure 6: Fine-tuning: For every pair, from left to right - $G(x_\omega)$ before and after fine-tuning. It shows that our proposed method is robust against removal attack (*i.e.* fine-tuning) as it is clearly noticed that the embedded watermark (top left corner) is still remained intact for DCGAN, SRGAN and CycleGAN.

References

[1] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *CVPR*, 2017, pp. 1125–1134.

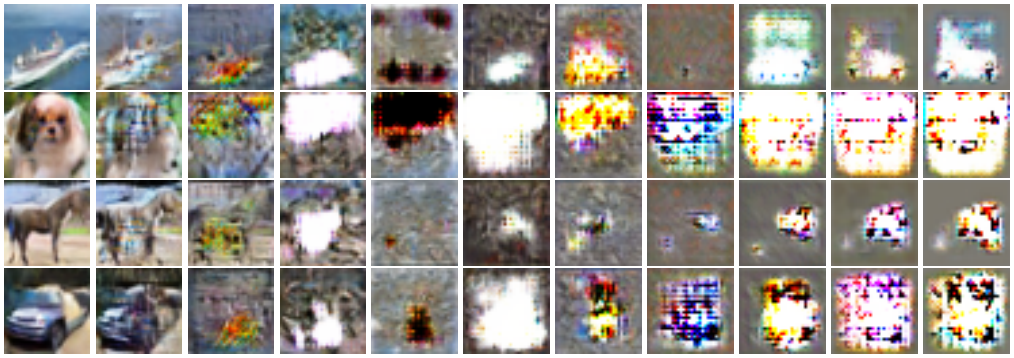
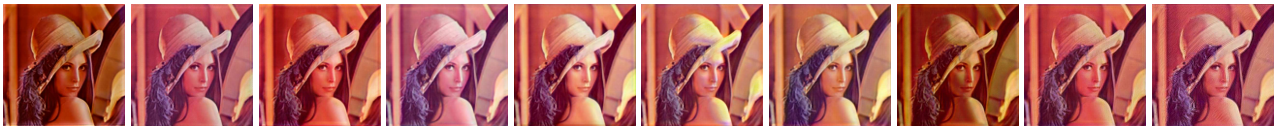


Figure 7: Ambiguity attack - DCGAN: It can be seen that the quality of the image drop significantly when the sign of the γ^{BN} of DCGAN$_{ws}$ is modified. Left to right: The amount (from 0% to 100%) of the sign is being modified.

(a) Comic

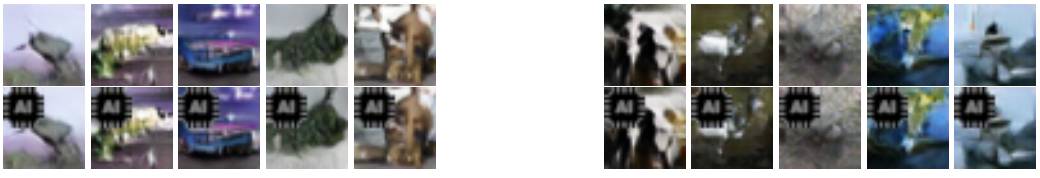
(b) Baboon



(c) Lenna

Figure 8: Ambiguity attack - SRGAN: It can be seen that the quality of the images drop significantly when the sign of SRGAN$_{ws}$ is being modified. Left to right: The amount (from 10% to 100%) of the sign is being modified.

(a) $\lambda = 0.1$ (b) $\lambda = 0.5$ (c) $\lambda = 1.0$



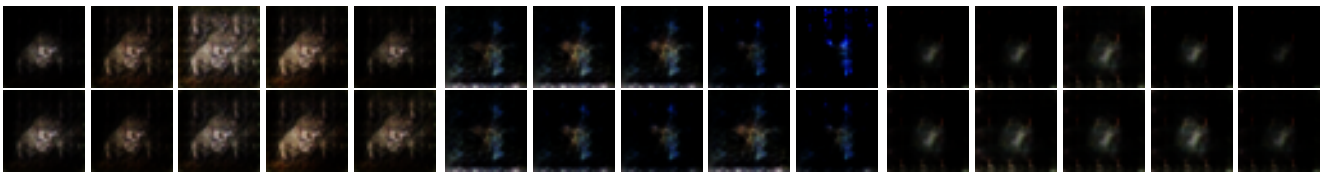
(d) $\lambda = 5.0$ (e) $\lambda = 10.0$

Figure 9: Effects of different λ to original model performance (top) and quality of generated watermark (bottom).

(a) $n = 5$; $c = -10$ (b) $n = 10$; $c = -10$ (c) $n = 15$; $c = -10$

(d) $n = 5$; $c = -5$ (e) $n = 10$; $c = -5$ (f) $n = 15$; $c = -5$

Figure 10: Effects of different n and c to original model performance (top) and quality of generated watermark (bottom) (cont. in Fig. 11).



(a) $n = 5$; $c = 0$ (b) $n = 10$; $c = 0$ (c) $n = 15$; $c = 0$

(d) $n = 5$; $c = 5$ (e) $n = 10$; $c = 5$ (f) $n = 15$; $c = 5$

(g) $n = 5$; $c = 10$ (h) $n = 10$; $c = 10$ (i) $n = 15$; $c = 10$

Figure 11: (cont.) Effects of different n and c to original model performance (top) and quality of generated watermark (bottom).