Detection of the Adobe Pattern

1st Jan Butora
UMR 9189 CRIStAL
Univ. Lille, CNRS, Centrale Lille
Lille, France
jan.butora@cnrs.fr

2nd Patrick Bas
UMR 9189 CRIStAL
Univ. Lille, CNRS, Centrale Lille
Lille, France
patrick.bas@cnrs.fr

***Abstract*—In this paper, we tackle the problem of detecting the so-called Adobe pattern. Recent research [4] showed that RAW and 16-bit images developed with the Lightroom or CameraRaw software into 8-bit formats are modified by an imperceptible periodical pattern. This 128×128 pattern is influenced by the 16-bit valued content and is incorporated in the 16-bit domain, making it impossible to estimate perfectly from real 8-bit images. Furthermore, as this periodic pattern can be perceived as a bias shared among different users and camera models, it has led to inaccurate camera attribution when working with the Photo-Response Non-Uniformity (PRNU). To effectively eliminate this bias, it is therefore imperative to have an accurate method of detecting the Adobe pattern. We model the content-dependent Adobe pattern as a deterministic pattern corrupted by uniform noise, which enables us to frame the detection of the Adobe pattern as a hypothesis test. Using the Likelihood Ratio Test, we demonstrate that for images without the Adobe pattern, a meticulously designed test statistic follows a zero-mean Gaussian distribution with a constant variance. Moreover, the detection accuracy exceeds 90% at false positive rate of 10^{-4} for 128×128 images JPEG compressed with quality 80, and improves with higher image quality. Finally, we find that around 16% of images in the FFHQ dataset [9] of real faces contain the Adobe pattern.**

***Index Terms*—Adobe, 16-bit watermark, expected watermark detection**

I. Introduction and previous works

The Photo-Response Non-Uniformity (PRNU) has been a very popular forensic tool for camera sensor attribution ever since its development in 2005 [10]. It states that each photo-site of a camera sensor is distorted by a multiplicative noise (i.e. proportional to the light intensity entering the sensor), the PRNU, which is present due to various physical imperfections during its manufacture. This noise is the same for all scenes for a given sensor, however is different from one sensor to another. The PRNU survives the image development process and its effect can be described as:

$$\mathbf{I} = \mathbf{I}^0 + \mathbf{K}\mathbf{I}^0 + \mathbf{\Theta}, \tag{1}$$

where $\mathbf{I}, \mathbf{I}^0, \mathbf{K}$, and $\mathbf{\Theta}$ represent respectively the captured image, the noiseless image, the PRNU, and a collection of random independent noise components.

Estimating the PRNU $\hat{\mathbf{K}}$ for a given camera sensor provides a camera fingerprint, which can be used for various forensic tasks, such as attributing images to the sensor or detecting local manipulations of the images with a False Positive (FP) rate of $\sim 10^{-5}$ [5], [7].

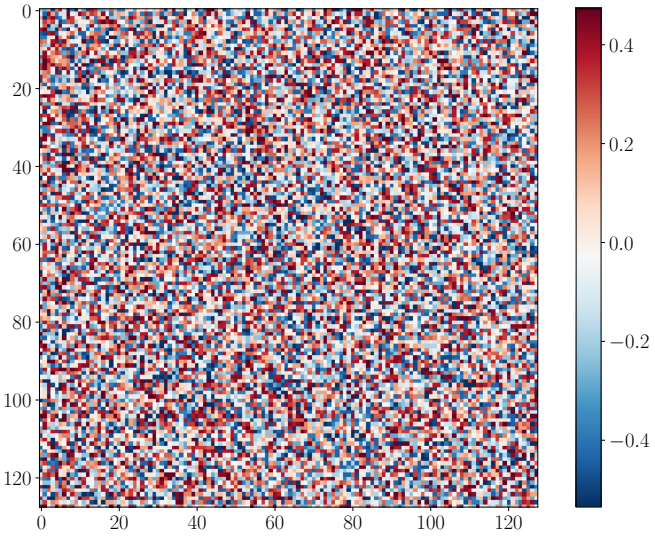

Fig. 1. The estimated Adobe pattern $\mathbf{w}$ used for detection in this work.

Recently, several publications, however, pointed out that the previously reliable PRNU suffers from much larger FP rates [1]–[3], [8]. Most notably, we showed in our recent work [4] that RAW and 16-bit coded images developed by Adobe Lightroom and Photoshop's plug-in CameraRaw are modified by a deterministic and content-dependent periodic pattern $\mathbf{w}$ of size 128×128. Expanding this pattern so it fits the whole image, its effect can be written as:

$$\mathbf{I} = \mathbf{I}^0 + \mathbf{K}\mathbf{I}^0 + \mathbf{w} + \mathbf{\Theta}. \tag{2}$$

In the paper, we showed that this additional bias corrupts the PRNU estimates leading to false positives whenever the images have been developed with the Adobe software. Furthermore, in [4] we proposed two ways of removing this pattern. However, to improve the methods of the pattern removal, we first need to be able to reliably establish the presence of the pattern, which is the main contribution of this paper. This detector can also be used to check if the pattern appears inside a reference database or for forensic purposes.

II. Notation and Preliminaries

A. Notation

Boldface symbols are reserved for matrices and vectors. Uniform distribution on the interval (a, b) will be denoted $\mathcal{U}(a, b)$ while $\mathcal{N}(\mu, \sigma^2)$ is used for the Gaussian distribution with mean μ and variance σ^2. For a variable X, we denote $\mathbb{E}[X] = \mu_X$ its mean and σ_X^2 its variance. If X is not a random variable, we will understand sample mean and variance by these symbols.

To quantize a 16-bit value to 8-bit, typically implemented with bit shifts, we define the quantizer operator:

$$Q(x) = \left\lfloor \frac{x}{256} \right\rfloor, \tag{3}$$

where $\lfloor x \rfloor$ denotes the floor operator.

The original 16-bit grayscale image with $N_1 \times N_2$ pixels is denoted $\mathbf{X}' \in \{0, 1, \dots, 2^{16} - 1\}^{N_1 \times N_2}$, while its 8-bit quantized version is $\mathbf{X} = Q(\mathbf{X}') \in \{0, 1, \dots, 255\}^{N_1 \times N_2}$.

For ease of notation, we will consider that images are composed of N vectorized blocks of size n: $\mathbf{X} = (\mathbf{x}_i)_{i=1}^{N}$, with $\mathbf{x}_i \in \{0, \dots, 255\}^n$ and we will be dropping the block index if not necessary. For a given block $\mathbf{x}$, we denote $x_i \in \{0, \dots, 255\}, i = 1, \dots, n$ its pixels. In our scenario, we will use $n = 128 \times 128$ and $N = 4 \times 4$.

B. The Adobe Pattern

It was pointed out in [4] that developing images with Adobe Lightroom or Camera Raw from RAW or 16-bit format to 8-bit formats (whether uncompressed or JPEG compressed) injects a noise-like periodic pattern of size 128×128. This pattern, which Adobe claims is used for image dithering, is inserted into the 16-bit representation before the subsequent quantization to 8-bit format, which makes it visually imperceptible. We were informed that the dithering function is implemented in the Adobe DNG SDK, specifically the `dng_utils.cpp` file. However, the pseudo-random generator of the mentioned dithering function (called `dng_dither`) is always initialized with the same seed, creating a deterministic signal w'. Furthermore, the function `dng_encode_proxy_task` in `dng_negative.cpp` shows how this signal, which we will refer to as *the Adobe pattern*, is injected into the image:

$$\mathrm{Dither}(x') = ((x' \ll 8) - x' + w') \gg 16, \tag{4}$$

where $\ll, \gg$ represent the left and right bit-shifts.

This creates a detectable statistical bias which can lead to false positives in camera sensor attribution [8]. We define the Adobe quantizer as:

$$Q_A(x') = Q(x' + w'(x')), \quad x' \in \{0, \dots, 2^{16} - 1\}, \tag{5}$$

where $w'(x') = \frac{w' - x'}{256}$ is the 16-bit valued content-dependent Adobe pattern.

While the Adobe pattern is a periodical signal, it was shown that it depends on both, the architecture used for the image development, and the image content itself because of the conversion from 16 bits to 8 bits. It is therefore not straightforward to remove such a signal from a given image, although it has been proposed in [4].

In this work, we aim to advance the previously proposed methodology by first designing a classifier able to detect the presence of this pattern. This detector could be for example used to first detect the watermark and then afterward to suppress it during the PRNU estimation in order to remove false positives.

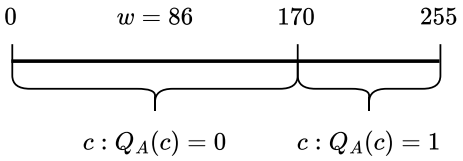

Fig. 2. The scheme used for brute-forcing the Adobe pattern with value $w = 86$. The quantization threshold for the $Q_A(.)$ quantizer is $256 - w = 170$.

III. Adobe Pattern Detection

To detect the presence of the Adobe Pattern, we first estimate it and subsequently use hypothesis testing to establish the presence of the estimated pattern.

A. Estimating the Pattern

While the Adobe pattern is variable across pixel values and architectures, we have been able to estimate its behavior. In particular, for every developed pixel y_i, the pattern is added as a last step of the development before the quantization to 8 bits:

$$\begin{aligned} y_i &= Q_A(x_i') \\ &= x_i + w_i(x_i') + u_i, \end{aligned} \tag{6}$$

where $x_i = x_i'/256$, $w_i(x_i') = (w_i'(x_i') - 128)/256$, and $u_i \sim \mathcal{U}(-1/2, 1/2)$ is a uniform quantization error. While the Adobe pattern can be viewed as a keyless watermark, the fact that the watermarked images are further corrupted by a stochastic component fundamentally distinguishes it from a typical additive watermark thus making its detection somewhat more complicated.

Furthermore, we brute-forced the Adobe pattern for values $c \in \{0, \dots, 255\}$ by developing 256 constant 16-bit images of size 128×128, each containing only one single value c, into 8-bit with Adobe Lightroom. We concluded that at any given position in the block, the Adobe pattern value is equal to

$$w_i' = 256 - \min\{c_i : Q_A(c_i) = 1\}, \tag{7}$$

where this process is visualized in Figure 2.

After repeating similar experiments for larger values c, we observed a very useful property of the Adobe pattern on different pixel values. If we shift all the block pixels by 256,

all the watermark values get shifted by some constant value a. Specifically, we observe from the definition of $w'(x')$ that

$$w'_i(x') = w'_i(x' + 256a) - a, \qquad \forall i = 1, \dots, n, \tag{8}$$

for any $a \in \{-256, \dots, 256\}$. Consequently, we model the Adobe pattern of any block of pixels with

$$\begin{aligned} w'_i(x') &= w'_i(x' \mod 256) + a_i \\ &= w'_i + a_i, \end{aligned} \tag{9}$$

where $a_i \sim \mathcal{U}(-256, 256)$ and w'_i comes from (7). Note that this implies $w'_i = \mathbb{E}[w'_i(x')]$, where the expectation is taken over all possible pixel values $x' \in \{0, \dots, 2^{16} - 1\}$. Plugging (9) into (6), we obtain

$$y_i = x_i + w_i + \xi_i, \tag{10}$$

where $w_i = (w'_i - 128)/256$, and ξ_i is a zero-mean noise component composed of $u_i, a_i/256$ and potentially other post-processing noise.

Consequently, the values w_i can be seen as the expected 8-bit float-valued Adobe watermark (with a bias correction). The values of $\mathbf{w}$ are visualized in Figure 1 and available for download at https://github.com/janbutora/adobe-detector.

Finally, we observed that for images taken in portrait orientation (i.e. after 90^0 rotation), the Adobe pattern is embedded as if the image was taken in landscape orientation (see Figure 3 for visualization).

B. Detection of Expected Watermark

In the following, we will cast the problem of detecting the Adobe pattern as a detection of a known watermark, even though we only consider the expected value of the Adobe pattern. The detection of a known watermark $\mathbf{w} \in \mathbb{R}^n$ in an image block $\mathbf{y} \in \mathbb{R}^n$ with noise residual $\mathbf{r} = \mathbf{y} - \hat{\mathbf{x}}$ can be described as a hypothesis test

$$\mathcal{H}_0 \quad : \quad r_i = x_i - \hat{x}_i + \xi_i, \tag{11}$$
$$\mathcal{H}_1 \quad : \quad r_i = w_i + x_i - \hat{x}_i + \xi_i, \tag{12}$$

where $\hat{\mathbf{x}}, \xi$ represent respectively the denoised image, and an independent noise component coming from both the post-processing (*e.g.* JPEG compression) and the impact of the host content during quantization (see previous subsection).

We use the denoiser proposed in [11], typically employed for the PRNU estimation [7]. For simplicity, we assume that both the image noise and the residual are zero-mean Gaussians with $x_i - \hat{x}_i \sim \mathcal{N}(0, \sigma_x^2)$, and $\xi_i \sim \mathcal{N}(0, \sigma_\xi^2)$. With these simplifications, we obtain

$$\mathcal{H}_0 \quad : \quad r_i \sim \mathcal{N}(0, \sigma_r^2), \tag{13}$$
$$\mathcal{H}_1 \quad : \quad r_i \sim \mathcal{N}(w_i, \sigma_r^2), \tag{14}$$

where $\sigma_r^2 = \sigma_x^2 + \sigma_\xi^2$.

Fig. 3. The location of the Adobe pattern $\mathbf{w}$ in portrait-oriented images.

The test statistic of the optimal test, the Likelihood Ratio Test, can then be expressed as a correlation between $\mathbf{r}$ and $\mathbf{w}$. However, to unify the distribution of the test statistic under the null hypothesis for any possible image, we use the Pearson correlation coefficient as a test statistic for a single block of size n:

$$\rho = \frac{\frac{1}{n}\sum_{i=1}^{n}(r_i - \mu_{\mathbf{r}})(w_i - \mu_{\mathbf{w}})}{\sigma_{\mathbf{r}}\sigma_{\mathbf{w}}}, \tag{15}$$

where $\sigma_{\mathbf{r}}, \sigma_{\mathbf{w}}$ denote the empirical standard deviations of the block residual $\mathbf{r}$ and the watermark $\mathbf{w}$. It is then straightforward to show that under the null hypothesis $\mu_\rho = 0$ and $\sigma_\rho^2 = \frac{1}{n}$, where we remind the reader that in our case $n = 128^2$. Aggregating the test statistic from the whole image, the Central Limit Theorem (CLT) states that under $\mathcal{H}_0$,

$$T = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} \rho_i \xrightarrow{d} \mathcal{N}\left(0, \frac{1}{128^2}\right). \tag{16}$$

Note that the test statistic T (16) is rather convenient as its distribution under the null hypothesis does not depend on the image size. We can compute a decision threshold τ_α for a prescribed FP rate α as

$$\tau_\alpha = \frac{Q^{-1}(\alpha)}{128}, \tag{17}$$

where $Q^{-1}(\cdot)$ is the inverse Q-function.

Under the alternative hypothesis, we find for a single block that

$$\begin{aligned}\mathbb{E}_{\mathcal{H}_1}[\rho] &= \frac{\sum_i \mathbb{E}\left[r_i w_i\right]}{n\sigma_{\mathbf{r}}\sigma_{\mathbf{w}}} \\ &= \frac{\sum_i w_i^2 + w_i\mathbb{E}[x_i - \hat{x}_i + \xi_i]}{n\sigma_{\mathbf{r}}\sigma_{\mathbf{w}}} \\ &= \frac{\sigma_{\mathbf{w}}}{\sigma_{\mathbf{r}}}, \end{aligned} \tag{18}$$

where we assumed for simplicity $\mu_{\mathbf{r}} = \mu_{\mathbf{w}} = 0$. Similarly, we find

$$\begin{aligned}\mathrm{Var}_{\mathcal{H}_1}(\rho) &= \frac{\sum_i \mathrm{Var}\left(r_i w_i\right)}{n^2\sigma_{\mathbf{r}}^2\sigma_{\mathbf{w}}^2} \\ &= \frac{\sum_i w_i^2 \mathrm{Var}(x_i - \hat{x}_i + \xi_i)}{n^2\sigma_{\mathbf{r}}^2\sigma_{\mathbf{w}}^2} \\ &= 1/n. \end{aligned} \tag{19}$$

We cannot employ the CLT the same way as for the null hypothesis due to different means for every ρ_i. Still, we can observe that the detection of the alternative hypothesis will be more powerful in blocks with small residual variance $\sigma_{\mathbf{r}}^2$. This also implies that the harsher the post-processing of the developed images, the weaker the pattern will be since σ_ξ and consequently $\sigma_{\mathbf{r}}$ will increase. For instance, we will see in Section III-D that the pattern is almost undetectable after JPEG compression with Quality Factor 20 but is very detectable for higher JPEG quality settings.

Interestingly, we can make a claim that considering a different test statistic

$$T_1 = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} \left(\rho_i - \frac{\sigma_{\mathbf{w}}}{\sigma_{\mathbf{r}_i}} \right) \tag{20}$$

allows us to model this statistic's distribution under the alternative hypothesis with

$$T_1 \xrightarrow{d} \mathcal{N}\left(0, \frac{1}{128^2}\right). \tag{21}$$

This provides an interesting trade-off between control of the False Positive (FP) rate and control of the True Positive (TP) rate, but for the purposes of this work, we only use the test statistic (16).

C. Experimental Setup

To evaluate our methodology, we randomly selected 10k uncompressed grayscale images from ALASKA 2 dataset [6] of size 2048×2048 and generated from them non-watermarked and watermarked images. To generate the non-watermarked images, we cropped the upper-left corner of size 512×512 and JPEG compressed using python's `PIL` library with Quality Factors (QFs) 20, 40, 60 and 80.

To generate the watermarked images, we first converted the images into 16-bit integers by multiplying the pixel values by 2^8 and saved them with python `tifffile` library into 16-bit TIF images. We then used Adobe Lightroom Classic to resize the images to 512×512 and JPEG compress with the same range of quality factors as above. While the manual conversion of images to 16-bit TIF might seem unnecessary, the subsequent conversion from 16-bit to 8-bit ensured that the Adobe pattern was embedded.

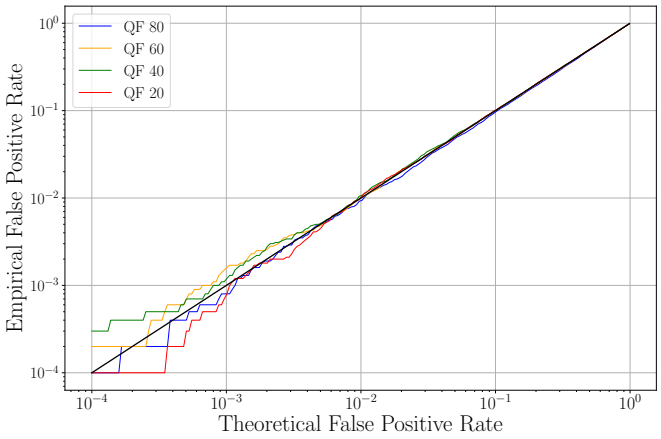

Fig. 4. False positive rate: theoretical vs empirical check for 512×512 images JPEG compressed with different quality factors.

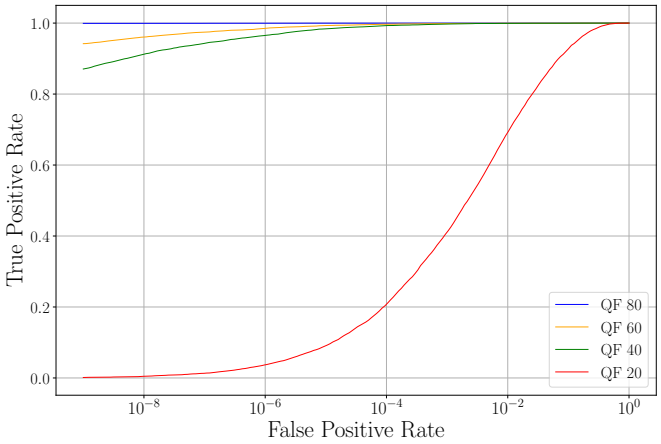

Fig. 5. True positive rate for watermarked images of size 512×512 compressed with different JPEG quality factors as a function of Gaussian FP rate.

D. Results

First, we study the quality of the Gaussian assumption (16). In Figure 4, we show empirical FP rate as a function of a Gaussian FP rate obtained as $\alpha = P(X > \tau_\alpha)$ for $X \sim \mathcal{N}(0, \frac{1}{128^2})$, and τ_α defined in (17). The empirical FP rate is then computed as $P(T > \tau_\alpha | \mathcal{H}_0)$ and we see a very clear fit for values above $\alpha = 10^{-3}$. For lower values of α, the fit gets noisy due to an insufficient amount of data.

Due to the very accurate match between the two FP rates, we only consider the theoretical FP rate to compute the TP rate: $TPR = P(T > \tau_\alpha | \mathcal{H}_1)$. We show the ROC curve obtained this way in Figure 5. On one hand, we can observe a TP rate greater than 90% for JPEG Quality Factors (QF)

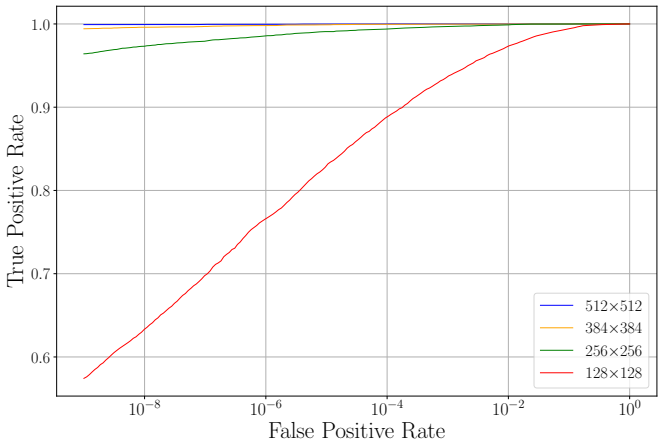

Fig. 6. True positive rate for watermarked images JPEG compressed with quality factor 80 and cropped into different sizes as a function of Gaussian FP rate.

higher than 20 for a theoretical FP rate as small as 10^{-8}. On the other hand, it seems that at QF 20, the watermark is heavily corrupted making it very hard to detect, although some detection is still possible for FP rates above 10^{-5}.

While the distribution of the test statistic under the null hypothesis (16) is independent of the image size, it is not true under the alternative hypothesis, a fact that is suggested by the different scaling factors between (16) and (20). Figure 6 shows the TP rate for different image crops compressed with QF 80. It is shown that even for a single block of size 128×128 the adobe pattern is close to 90% detectable at FP rate 10^{-4}, while the detection increases rapidly for larger image sizes.

To verify how widespread the Adobe pattern is in real data, we randomly selected 1000 images from the FFHQ dataset [9]. With a threshold set for FP rate 10^{-5}, we found that 15.6% of these images are carrying the Adobe pattern. This observation can have serious consequences for the forensics community.

IV. Conclusions and perspectives

This paper is the continuation of the contribution [4] which shows that Adobe Lightroom embeds a pattern/watermark while developing RAW or 16-bit images. Since the watermark is not key-dependent, it is possible to obtain its expectation by brute-forcing specific inputs into the software. A correlation-based detector is designed to detect the watermark on each 128×128 patch. Results indicate that the detector achieves good detection performance even on small patches and JPEG quality factors smaller than 80.

We did not consider additional geometric image transformations, as such a study would be out of the scope of this paper. However, we plan to expand the proposed method to additional image post-processing operations, such as rotations, resizing, etc.

Interestingly, we also found out that popular databases in machine learning, such as FFHQ-wild, have more than 15% of the images corrupted with the Adobe pattern. Our future works will explore how to use this detector for forensic purposes or to potentially detect data poisoning in machine learning.

The code used to estimate the Adobe pattern and to generate the results in this work is available at https://github.com/janbutora/adobe-detector.

V. Acknowledgements

This work received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 101021687 (project "UNCOVER") and the French Defense & Innovation Agency. This work was also supported by a French government grant managed by the Agence National de la Recherche under the France 2030 program, reference ANR-22-PECY-0011.

References

[1] Chiara Albisani, Massimo Iuliani, and Alessandro Piva. Checking PRNU Usability on modern devices. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2535–2539. IEEE, 2021.

[2] Daniele Baracchi, Massimo Iuliani, Andrea G Nencini, and Alessandro Piva. Facing image source attribution on iPhone X. In *Digital Forensics and Watermarking: 19th International Workshop, IWDW 2020, Melbourne, VIC, Australia, November 25–27, 2020, Revised Selected Papers 19*, pages 196–207. Springer, 2021.

[3] Nabeel Nisar Bhat and Tiziano Bianchi. Investigating inconsistencies in prnu-based camera identification. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 851–855. IEEE, 2022.

[4] Jan Butora and Patrick Bas. The Adobe Hidden Feature and its Impact on Sensor Attribution. working paper or preprint, December 2023.

[5] Giovanni Chierchia, Giovanni Poggi, Carlo Sansone, and Luisa Verdoliva. A bayesian-mrf approach for prnu-based image forgery detection. *IEEE Transactions on Information Forensics and Security*, 9(4):554–567, 2014.

[6] R. Cogranne, Q. Giboulot, and P. Bas. ALASKA–2: Challenging academic research on steganalysis with realistic images. In *IEEE International Workshop on Information Forensics and Security*, New York, NY, December 6–11, 2020.

[7] Miroslav Goljan, Jessica Fridrich, and Tomáš Filler. Large scale test of sensor fingerprint camera identification. In *Media forensics and security*, volume 7254, pages 170–181. SPIE, 2009.

[8] Massimo Iuliani, Marco Fontani, and Alessandro Piva. A leak in PRNU based source identification questioning fingerprint uniqueness. *IEEE Access*, 9:52455–52463, 2021.

[9] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[10] Jan Lukas, Jessica Fridrich, and Miroslav Goljan. Determining digital image origin using sensor imperfections. In *Image and Video Communications and Processing 2005*, volume 5685, pages 249–260. SPIE, 2005.

[11] M Kivanc Mihcak, Igor Kozintsev, Kannan Ramchandran, and Pierre Moulin. Low-complexity image denoising based on statistical modeling of wavelet coefficients. *IEEE Signal Processing Letters*, 6(12):300–303, 1999.