The Effect of the JPEG Implementation on the Cover-Source Mismatch Error in Image Steganalysis

Martin Beneš
University of Innsbruck
martin.benes@uibk.ac.at

Nora Hofer
University of Innsbruck
nora.hofer@uibk.ac.at

Rainer Böhme
University of Innsbruck
rainer.boehme@uibk.ac.at

Abstract—**One of the challenges in steganalysis based on machine learning is cover-source mismatch (CSM). It occurs when a detector is trained on cover and stego pairs generated from one data source and then applied to data of unknown type from a different source. The mismatch leads to an increase of the classification error, which corresponds to a loss of detection performance. This paper studies how much the implementation of the JPEG compression and decompression contributes to the CSM of color image steganalysis. Specifically, it studies the differences between libjpeg versions 6b and 7. The impact on the CSM is measured for J-UNIWARD embedding using JPEG and spatial rich models with an ensemble classifier. It is observed that the use of different libjpeg versions may cause CSM. The error is large when different libjpeg versions are used for feature extraction, and barely measurable when different versions are used for the generation of cover and stego pairs.**

Index Terms—**jpeg, libjpeg, steganalysis, cover-source mismatch, compression, decompression**

I. Introduction

Applying steganalysis methods developed in laboratory conditions to real-world environments is difficult [1]. One reason for that is cover-source mismatch (CSM). Differences between cover sources appear in many forms, such as different image resolution, size, and format; processed images, images acquired by different hardware, and under different condition, as listed in [2]. Although well-recognized as a critical factor affecting the performance of steganalysis [3], [4], dealing with the CSM remains challenging because the large number of potentially influencing factors is barely tractable [5].

This work presents existential results for another, so far overlooked factor in image steganalysis: the JPEG implementation. The free, open-source, and cross-platform C library *libjpeg* is the established reference implementation for processing JPEG images [6]. It is the workhorse of numerous image processing libraries in high-level programming languages and applications built with them. A recent study found that different libjpeg versions may produce differences in compression and decompression output with a magnitude comparable to the differences produced by switching discrete cosine transform (DCT) methods [7]. However, it is yet unknown whether and to what extent these differences affect steganalysis performance and contribute to the CSM.

In a first step addressing this research gap, this paper

- demonstrates that the use of different libjpeg versions for training and testing can reduce the performance of a common steganalysis method; and
- quantifies the performance loss using the empirical total probability of error.

Specifically, we use J-UNIWARD as embedding function [8], JSRM features with an ensemble classifier as detector [9], and 10,000 never-compressed color images from the ALASKA2 dataset [10].

The organization of this paper is as follows. Section II recalls relevant concepts of JPEG compression and elaborates on the CSM in steganalysis. Section III explains the experimental setup and the fundaments of the embedding function and the detector used. Section IV presents the results, which are further discussed in the concluding Section V.

II. Background

Here we recall the basic concepts of JPEG image compression and summarize what is known about the CSM.

A. JPEG Compression

JPEG is a widely adopted lossy image compression standard [11]. It transforms digital image data from the spatial to the frequency domain, specifically DCT, where high-frequency information, e. g. sharp transitions in brightness or color, can be reduced virtually imperceptibly. The use of the YC_bC_r color model separates the luminance from the chrominance, which can be downsampled even further due to the lower sensibility of the human eye to color.

The popularity of the format along with the relative ease of modeling statistical distributions of frequency domain coefficients make JPEG a good choice for secure steganographic communication.

In 1991, the Independent JPEG Group established *libjpeg* as JPEG codec for image compression [6]. Since then, several new versions were released, ranging from libjpeg version 1 to the latest version libjpeg 9e, published in January 2022. Popular forks are libjpeg–turbo [12] and mozjpeg [13].

B. Cover-Source Mismatch

For more than 15 years, researchers in steganalysis have been struggling with the CSM [5]. Differences in the cover source can originate from the various steps in the image processing pipeline. Researchers in the field have started to systematically evaluate the impact of each step of the pipeline, including differences in the sensor [2], [14], demosaicking [15], [16], ISO sensitivity [14], [17], white balancing [15], gamma correction [15], color adjustment [17], image processing software [14], [15], [17], and compression quality [14], [18].

To describe the magnitude of the CSM between two sources, Giboulot et al. [14] establish the terms *source inconsistency* and *source intrinsic difficulty*. The former refers to the discrepancy in classification errors which occurs when training with source A compared to when training with source B. However, source inconsistency does not take into account the detection performances of the compared sources. Therefore the source intrinsic difficulty is used jointly. It describes the classification error when the training and test set derive from the same source. A high classification error indicates high difficulty.

One can distinguish three approaches to mitigate CSM [14]:

1) the atomistic/islet approach, used for example in [5], [19], which splits a dataset into clusters with regard to image properties;
2) the holistic approach, used in [20], which does the opposite: creating an as diverse training dataset as possible; and
3) transfer learning, which transfers training and test datasets between domains.

Our work tries to establish the existence of the CSM for a not yet studied factor. It is closest to the atomistic approach. The relevant image property is the JPEG implementation used to generate cover and stego pairs and, independently, to calculate steganalysis features.

III. Approach

In this section we describe our experiment, starting with the embedding function used by the steganographer, followed by the detector of the steganalyst.

A. Embedding

A common approach to obtain hard-to-detect steganography is to embed the payload while minimizing a distortion function. In 2014, Holub et al. [8] introduced a “universal” distortion design called Universal Wavelet Relative Distortion (UNIWARD). It supports embedding methods for the spatial domain (S-UNIWARD), JPEG domain (J-UNIWARD), and side-informed JPEG domain (SI-UNIWARD). UNIWARD estimates a cost matrix related to the impact of embedding changes on the statistical distortion of the image. In the basic form, J-UNIWARD modifies the non-zero quantized AC coefficients of the luminance channel while keeping the chrominance channels untouched.

According to [8, Sect. 6.2], the detectability of J-UNIWARD in the JPEG domain is lower than nsF5 [21], the state of the art embedding method in the JPEG domain at that time. We choose J-UNIWARD for the embedding due to its popularity and readily available implementation. Moreover, it is considered to belong to the most secure embedding methods

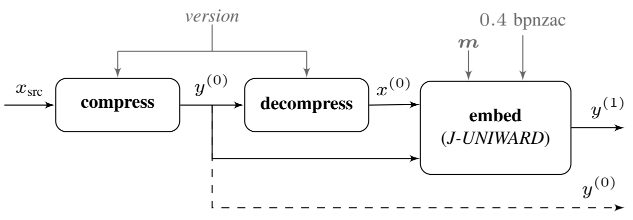

Fig. 1. Alice’s system model

in the JPEG domain [22]. We pick a stabilization constant $\sigma = 2^{-6}$, described as the optimal one by the J-UNIWARD authors. In this setup, the out-of-bag (OOB) error of the JSRM+EFLD model is about 0.18 [8, Figure 6].

B. Rich Models

Now we turn to the steganalyst. Rich models were introduced in 2012 and designed specifically for steganalysis [23]. They are feature sets of large, fixed dimensionality, modeling the relation between adjacent samples.

SRM is the first rich model. It is computed from the spatial domain [23]; hence the name spatial rich model (SRM). The noise residuals of the image, which are used for feature extraction, are quantized with $q \in \{1, 1.5, 2\}$ and the result has 34 671 dimensions. The subset SRMQ1 only contains features for $q = 1$, with 12 753 dimensions. SRMs are designed for grayscale images.

Cartesian-calibrated JPEG Rich Models (cc-JRM) [9] are rich models for the JPEG domain with 22 510 dimensions. Here, the features are extracted from the DCT coefficients. Cartesian calibration (cc) is used to exploit the regularity of a JPEG 8×8 grid, which has been the key insight behind a number of successful attacks against embedding functions in the JPEG domain [24], [25].

The union of cc-JRM and SRMQ1, denoted JSRM [9] has 34 671 dimensions combining features from both the spatial and JPEG domain.

C. Detector

Following [26], we construct an ensemble of at most 500 base learners with a stopping criterion based on the moving average of OOB. Each base learner implements Fisher’s Linear Discriminant (FLD) analysis [27] with a subset of d_{sub} random features, where d_{sub} is automatically chosen as described in Section II/C of [26]. The aggregation to the final class uses majority voting.

Although it has been shown that deep learning in steganalysis can be at least as precise as rich models [28], JSRM+EFLD is still competitive [9]. Its comparative simplicity, the reliance on JPEG operations, and the susceptibility to the CSM error make it an ideal target for this first study.

D. Dataset

For our experiments we select a random sample of 10,000 never compressed color images from the ALASKA2 dataset [10]. It contains images from 24 different cameras. To

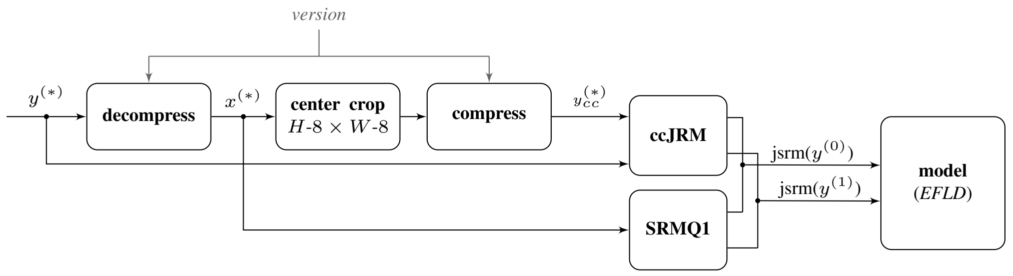

Fig. 2. Eve's system model

limit storage and compute efforts, we work with the reduced dataset of cropped images of size 256×256 pixels, called `ALASKA_v2_TIFF_256_COLOR` [10].

E. Experimental Setup

Our experiments seek to examine whether the choice of the libjpeg version affects steganalysis performance and contributes to the CSM. The versions used in this work are libjpeg 6b and 7. The choice is based on prior work, which establishes that those versions produce the biggest differences in outputs for the same image. They are of the same magnitude as, for instance, switching between different DCT methods [7].

The experiments are divided into the steganography part, performed by Alice, and the steganalysis part, performed by Eve. Alice's and Eve's libjpeg implementations are chosen independently.

By convention, an image in the spatial domain is denoted x and in the DCT domain y. Cover objects have the superscript $^{(0)}$ and stego objects have the superscript $^{(1)}$, regardless of the embedded message.

a) Steganographer Alice: Alice starts with a never-compressed source image x_{src} and compresses it to a JPEG cover $y^{(0)}$. This cover is used to train a classifier along with the corresponding stego image. Alice produces the stego image $y^{(1)}$ by embedding a message $\boldsymbol{m}$ into the cover image $y^{(0)}$ using the J-UNIWARD embedding function with an embedding rate of 0.4 bits per non-zero DCT AC coefficient (bpnzac). Since J-UNIWARD requires a spatial domain representation to estimate the embedding cost of DCT coefficient changes, $y^{(0)}$ needs to be decompressed to the JPEG pre-compressed cover $x^{(0)}$ in spatial domain. We assume that Alice uses the same libjpeg version for compression and decompression, noting that this may not always be the case in practice. Figure 1 illustrates her workflow.

b) Steganalyst Eve: Figure 2 illustrates Eve's system model. Her input is an image of unknown type $y^{(*)}$, meaning it can be a cover $y^{(0)}$ or a stego image $y^{(1)}$. Eve starts with decompressing the input, resulting in $x^{(*)}$. For the cartesian calibration she crops the image by 4 pixels on all sides and re-compresses it, resulting in $y_{cc}^{(*)}$. We assume that Eve uses the same libjpeg version for compression and decompression. Note, that this assumption is more realistic for Eve's task than for Alice's. The images $y^{(*)}$ and $y_{cc}^{(*)}$ are used to extract the cc-JRM features from the luminance channel, and the decompressed input $x^{(*)}$ for the SRMQ1 features.

As SRMs are designed for grayscale images, we extend them to color images by extracting the features from the green channel only, which is most correlated with the luminance channel used for embedding. Our initial experiments with features extracted from a conversion to grayscale produced inferior results, in line with [29]. So we abandoned this alternative. Both feature vectors are combined to produce the JSRM feature vector jsrm$(y^{(*)})$.

In order to train the model, Eve instantiates both Alice's and her system model and produces cover and stego pairs with known labels. In our experiments, we vary the libjpeg versions for Alice and Eve during this training process independently from the libjpeg version used for Alice and Eve when testing (i. e., simulating the classification of unknown types). With two libjpeg versions, two system models, and two phases (training and test), we get a total of $2^{2^2} = 16$ combinations, dealing with $2^2 = 4$ different trained classifiers indexed with the roman numbers I–IV. All experiments use a single $7\,500 : 2\,500$ split of cover/stego pairs (and corresponding feature vectors) for training and testing.

F. Implementation

For compressing and decompressing JPEG images we use libjpeg with a Python wrapper that supports version selection [30]. We use the Matlab code provided by the Digital Data Embedding Laboratory [31] for J-UNIWARD simulator (`J_UNIWARD.m`) and feature extractors for SRMQ1 (`SRMQ1.m`) and cc-JRM (`ccJRM.m`).

A single process on a MacBook Pro 2020/M1 machine ran for two hours to embed J-UNIWARD into 10 000 cover images, 16 hours to extract the JSRM features from both cover and stego images, and about 10 minutes to train one of the four classifiers based on the extracted features. The features computed once for a combination of library versions could be reused in tests of all four classifiers. For replicability, we make our code and the extracted feature vectors available on GitHub.[1]

IV. Results

Table I reports the performance of four classifiers, each trained and evaluated with a different combination of libjpeg

[1]https://github.com/uibk-uncover/libjpeg-CSM

TABLE I
CLASSIFICATION PERFORMANCE USING DIFFERENT COMBINATIONS OF LIBJPEG VERSIONS

Classifier	Training		Evaluation		Test data			Training data		
	Alice	Eve	Alice	Eve	P_E	P_{FP}	P_{MD}	P_E	P_{FP}	P_{MD}
I	6b	6b	6b	6b	**0.29**	0.29	0.29	**0.17**	0.18	0.16
	6b	6b	6b	7	0.29	*0.44*	*0.13*	0.24	0.41	0.07
	6b	6b	7	6b	0.29	0.29	0.28	0.23	0.20	0.27
	6b	6b	7	7	0.28	*0.44*	*0.13*	0.27	0.42	0.12
II	7	7	6b	6b	0.51	0.31	0.70	0.44	0.17	0.71
	7	7	6b	7	0.31	0.31	0.30	0.25	0.20	0.30
	7	7	7	6b	0.50	0.31	0.70	0.41	0.15	0.66
	7	7	7	7	**0.30**	0.31	0.30	**0.18**	0.17	0.18
III	6b	7	6b	6b	0.50	0.30	0.69	0.41	0.17	0.65
	6b	7	6b	7	**0.30**	0.30	0.31	**0.20**	0.20	0.20
	6b	7	7	6b	0.51	0.31	0.70	0.46	0.20	0.71
	6b	7	7	7	0.31	0.30	0.31	0.26	0.22	0.30
IV	7	6b	6b	6b	0.29	0.29	0.28	0.23	0.18	0.27
	7	6b	6b	7	0.28	*0.44*	*0.13*	0.26	0.42	0.11
	7	6b	7	6b	**0.28**	0.29	0.28	**0.16**	0.16	0.16
	7	6b	7	7	0.28	*0.44*	*0.12*	0.24	0.41	0.07

versions. For each combination we report the probabilities of false positives P_{FP}, missed detections P_{MD}, and their mean, the total probability of error P_E. Lower error indicates a better model performance, i. e., improved detectability. Cells highlighted in yellow denote baseline cases where the JPEG implementations for training and test are identical, meaning that no CSM is present. Figures in red guide the reader to the relevant cases of CSM error.

Classifier I seems to be robust against version changes in the test data. However, under the surface we can see an increase of P_{FP} and a decrease of P_{MD}, whenever Eve's version changes for evaluation.

When looking at classifier II, which uses version 7 for Alice and Eve to produce training data, we see a comparable intrinsic difficulty (highlighted in yellow). However, as soon as Eve's version changes for evaluation, we observe an increase in the error rate. We interpret this as evidence for the CSM error due to the JPEG implementation, specifically the libjpeg version. Looking at the values in column P_{MD}, we see that this performance loss is a result of an increase in missed detections. Changing Alice's libjpeg version, on the other hand, does not seem to impact the performance in this setting.

This holds true even in the case where Eve is aware of the problem and deliberately trains the classifier on data produced with a different version (classifiers III and IV). However, we see differences depending on Eve's version used for generating training data. Again, when trained with data generated using version 6b, the classifier appears robust. In the case where Eve uses version 7 for training and version 6b for evaluation, we observe the same increase in the error rates, as seen before in classifier I.

For all classifiers we show the discrepancy between the P_E on test data versus training data, which indicates overfitting. For example, classifier I produces as little as 17 % classification error on training data, whereas it errs in 29 % on the test data. While this is not very surprising for rich models given the dimensionality of the feature space and dataset, we note that overfitting can amplify the sensitivity to CSM. However, we are not aware of work stating that the CSM disappears when overfitting is avoided.

Summarizing our results, we see that a change of the compression library version after embedding can have an effect on the CSM. We observe a CSM error in cases where Eve trains a classifier using version 7 and extracts features for the classification of unknown material using version 6b. While this provides existential evidence for the relevance of the JPEG implementation for CSM, we observe its strongest impact in situations that the steganalyst can control. She would be in a more challenging situation if knowledge of Alice's libjpeg version was decisive for the detection performance.

V. CONCLUSION

We emphasize that when JPEG compression is part of the cover image processing pipeline, special attention should be paid to the choice of the JPEG implementation, including the version of libjpeg.

It is worth recalling the limitations of this initial study. The experiments were carried out using one embedding method with a single embedding and rate, one steganalysis model, and two JPEG codec versions known to produce different outputs. We also fixed the version for compression and decompression to be the same in Alice's system model. This suffices as a proof of existence of an effect of the JPEG library version on the CSM error. But we warn against generalizing our results prematurely. Further investigations are needed to characterize the conditions and magnitude of this effect.

Acknowledgment

This work is funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101021687 (UNCOVER).

References

[1] A. Ker, P. Bas, R. Böhme, R. Cogranne, S. Craver, T. Filler, J. Fridrich, and T. Pevný, "Moving steganography and steganalysis from the laboratory into the real world," in *Workshop on Information Hiding and Multimedia Security*. Association for Computing Machinery, 2013, pp. 45–58.

[2] J. Kodovský, V. Sedighi, and J. Fridrich, "Study of cover source mismatch in steganalysis and ways to mitigate its impact," in *Media Watermarking, Security, and Forensics*, vol. 9028. International Society for Optics and Photonics, 2014, pp. 204–215.

[3] J. Fridrich, J. Kodovský, V. Holub, and M. Goljan, "Breaking HUGO – the process discovery," in *Workshop on Information Hiding*. Springer, 2011, pp. 85–101.

[4] G. Gul and F. Kurugollu, "A new methodology in steganalysis: Breaking highly undetectable steganography (HUGO)," in *Workshop on Information Hiding*. Springer, 2011, pp. 71–84.

[5] M. Goljan, J. Fridrich, and T. Holotyak, "New blind steganalysis and its implications," in *Security, Steganography, and Watermarking of Multimedia Contents VIII*, vol. 6072. International Society for Optics and Photonics, 2006, pp. 1–13.

[6] Independent JPEG Group, "libjpeg," 1990. [Online]. Available: http://libjpeg.sourceforge.net/

[7] M. Beneš, N. Hofer, and R. Böhme, "Know your library: How the libjpeg version influences compression and decompression results," in *Workshop on Information Hiding and Multimedia Security*. Association for Computing Machinery, 2022.

[8] V. Holub, J. Fridrich, and T. Denemark, "Universal distortion function for steganography in an arbitrary domain," *Eurasip Journal on Information Security*, vol. 2014, no. 1, pp. 1–13, 2014.

[9] J. Kodovský and J. Fridrich, "Steganalysis of JPEG images using rich models," in *Media Watermarking, Security, and Forensics*, vol. 8303. International Society for Optics and Photonics, 2012, pp. 0A 1–13.

[10] R. Cogranne, Q. Giboulot, and P. Bas, "The ALASKA steganalysis challenge: A first step towards steganalysis," in *Workshop on Information Hiding and Multimedia Security*. Association for Computing Machinery, 2019, pp. 125–137.

[11] W. Pennebaker and J. Mitchell, *JPEG: Still image data compression standard*. Springer, 1992.

[12] libjpeg turbo, "libjpeg–turbo," 2021. [Online]. Available: https://libjpeg-turbo.org/

[13] Mozilla Foundation, "mozjpeg: Improved JPEG encoder," 2014. [Online]. Available: https://github.com/mozilla/mozjpeg

[14] Q. Giboulot, R. Cogranne, D. Borghys, and P. Bas, "Effects and solutions of cover-source mismatch in image steganalysis," *Signal Processing: Image Communication*, vol. 86, p. 115888, 2020.

[15] D. Borghys, P. Bas, and H. Bruyninckx, "Facing the cover-source mismatch on jphide using training-set design," in *Workshop on Information Hiding and Multimedia Security*. Association for Computing Machinery, 2018, pp. 17–22.

[16] M. Kirchner and R. Böhme, "Steganalysis in Technicolor: Boosting WS detection of stego images from CFA-interpolated covers," in *International Conference on Acoustics, Speech, and Signal Processing*. Institute of Electrical and Electronics Engineers, 2014, pp. 3982–3986.

[17] Q. Giboulot, R. Cogranne, and P. Bas, "Steganalysis into the wild: How to define a source?" *Electronic Imaging*, vol. 2018, no. 7, pp. 318–1–318–12, 2018.

[18] Y. Yousfi and J. Fridrich, "JPEG steganalysis detectors scalable with respect to compression quality," *Electronic Imaging*, vol. 4, pp. 75–1, 2020.

[19] J. Pasquet, S. Bringay, and M. Chaumont, "Steganalysis with cover-source mismatch and a small learning database," in *European Signal Processing Conference*. Institute of Electrical and Electronics Engineers, 2014, pp. 2425–2429.

[20] I. Lubenko and A. Ker, "Steganalysis with mismatched covers: Do simple classifiers help?" in *Multimedia & Security*. Association for Computing Machinery, 2012, pp. 11–18.

[21] J. Fridrich, T. Pevný, and J. Kodovský, "Statistically undetectable JPEG steganography: dead ends challenges, and opportunities," in *Multimedia & Security*. Association for Computing Machinery, 2007, pp. 3–14.

[22] A. Su, S. Ma, and X. Zhao, "Fast and secure steganography based on J-UNIWARD," *Signal Processing Letters*, vol. 27, pp. 221–225, 2020.

[23] J. Fridrich and J. Kodovský, "Rich models for steganalysis of digital images," *Transactions on Information Forensics and Security*, vol. 7, no. 3, pp. 868–882, 2012.

[24] J. Fridrich, M. Goljan, and D. Hogea, "Steganalysis of JPEG images: Breaking the F5 algorithm," in *Workshop on Information Hiding*. Springer, 2002, pp. 310–323.

[25] J. Fridrich, M. Goljan, D. Hogea, and D. Soukal, "Quantitative steganalysis of digital images: estimating the secret message length," *Multimedia Systems*, vol. 9, no. 3, pp. 288–302, 2003.

[26] J. Kodovský, J. Fridrich, and V. Holub, "Ensemble classifiers for steganalysis of digital media," *Transactions on Information Forensics and Security*, vol. 7, no. 2, pp. 432–444, 2011.

[27] R. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, pp. 179–188, 1936.

[28] G. Xu, H.-Z. Wu, and Y.-Q. Shi, "Structural design of convolutional neural networks for steganalysis," *Signal Processing Letters*, vol. 23, pp. 708–712, 2016.

[29] M. Goljan, J. Fridrich, and R. Cogranne, "Rich model for steganalysis of color images," in *International Workshop on Information Forensics and Security*. Institute of Electrical and Electronics Engineers, 2014, pp. 185–190.

[30] M. Beneš. (2022) Python package jpeglib. [Online]. Available: https://github.com/martinbenes1996/jpeglib

[31] Department of Electrical and Computer Engineering. Digital data embedding laboratory. [Online]. Available: http://dde.binghamton.edu/