Spatial Least Significant Bit Steganography Under Lossy Compression: Fragility and a Robust Transform-domain Alternative

Teodor PĂDURE
Faculty of Applied Sciences, UPB, Bucharest, Romania
teodorpadure748@gmail.com

Abstract: Least significant bit (LSB) steganography is known to fail under lossy compression. Using 103 tiles from eight photographs and 12,360 measurements, we show that JPEG, WebP, JPEG XL and double recompression all drive the LSB bit error rate to about 50%, because codec reconstruction error alone randomises pixel parity. As a reference robust baseline, we evaluate quantization index modulation on low-frequency transform coefficients with Reed-Solomon coding, tuned on held-out images. With an 800-bit net payload (1.6% of LSB capacity) at 38 dB PSNR, it recovers 87% of test messages through JPEG down to quality 70 and 82% at quality 60.
Keywords: image steganography, least significant bit, lossy compression, quantization index modulation, Reed-Solomon code, bit error rate.

INTRODUCTION

Steganography hides information inside other information so that its very presence goes unnoticed. Unlike cryptography, which admits that a message is secret and only makes it unreadable, steganography relies on the fact that no one is looking (Provos & Honeyman, 2003; Fridrich, 2009). Digital images are a convenient carrier because they hold a great deal of visual redundancy, and the human eye does not register the small intensity variations they are built from (Cheddad et al., 2010).

The method almost everyone meets first is least significant bit (LSB) substitution. The idea is direct: every pixel value is stored on eight bits, and the last of them contributes at most one unit to the intensity of a channel. Replacing that bit with a message bit changes the colour by at most one on a 0–255 scale, below the threshold of perception, while the capacity is generous, one hidden bit per channel of every pixel. Its weakness is equally well known, because the message lives in the parity of raw pixel values, any process

https://doi.org/10.54851/v8i2y202601

that rewrites those values disturbs it, and lossy compression rewrites them by design (Chandramouli & Memon, 2001; Ker, 2005).

This tension frames the paper. The appeal of LSB rests on the arithmetic stability of the spatial domain: the message survives only while pixel values stay untouched. A Portable Network Graphics (PNG) file, which compresses without loss, keeps everything bit for bit; a Joint Photographic Experts Group (JPEG) or WebP file recomputes the pixels to shed information it deems negligible, and the last bit is, by definition, the most negligible information in the image. The contribution of this paper is primarily empirical and comparative. First, we quantify how completely spatial LSB is destroyed, across a real-image corpus and a wide range of lossy encoders, including WebP, JPEG XL and repeated compression, rather than illustrating the point on a single image, and we show with un-embedded covers that the failure stems from the codecs' own reconstruction error. Second, as a reference point, we evaluate a standard robust construction, quantization index modulation with Reed-Solomon coding (Chen & Wornell, 2001), on the same test bench, with parameters selected on held-out images and with raw and net capacity reported separately. The construction itself is not new; what the paper adds is a reproducible measurement of both schemes across modern codecs. The remainder is organised as follows: "Related Work And Threat Model" reviews the relevant literature and establishes the threat model; "Least Significant Bit Steganography" introduces the LSB model and a control experiment; "Why LSB Fails Under Lossy Compression" explains the failure mechanism; "Experimental Method" describes the methodology; "Results" reports the survival results; and "A Robust Transform-Domain Alternative" evaluates the transform-domain baseline, including its parameter sensitivity, a luminance-domain variant, and its detectability.

RELATED WORK AND THREAT MODEL

Related work

The fragility of spatial LSB embedding and its statistical detectability have long been documented: LSB replacement leaves structural traces exploited by RS analysis (Fridrich, Goljan & Du, 2001) and by detectors for LSB matching (Ker, 2005), and its behaviour under image processing was analysed early on (Chandramouli & Memon, 2001). Embedding directly in quantized JPEG coefficients, as in F5 (Westfeld, 2001), avoids the decompression step but not detection; a Romanian study of a commercial F5-like tool showed that its JPEG stego images are identified with high accuracy by feature-based classifiers (Tataru, Vlad & Gugu, 2015). Modern undetectable steganography minimises an additive distortion with content-adaptive costs, HUGO (Pevný, Filler & Bas, 2010) and J-UNIWARD (Holub, Fridrich & Denemark, 2014), implemented with syndrome-trellis codes (Filler, Judas & Fridrich, 2011), while spatial methods may restrict embedding to textured regions selected by segmentation (Demircan & Ozekes, 2024). These schemes are evaluated against rich-model and deep steganalysers such as SRM (Fridrich & Kodovský, 2012), JRM (Kodovský & Fridrich, 2012) and SRNet (Boroumand, Chen & Fridrich, 2019).

Robust steganography for lossy channels combines these ideas with error resilience. Dither-modulation-based adaptive steganography selects coefficients that resist JPEG recompression (Zhang et al., 2018), transport channel matching pre-distorts the cover to match the channel (Zhao et al., 2019), and dual-syndrome trellis codes make syndrome coding tolerant to channel noise (Kin-Cleaves & Ker, 2018). In watermarking, robustness has been addressed by spread-spectrum embedding (Cox et al., 1997) and by quantization index modulation (QIM), whose combination with error-correcting codes

is standard since Chen and Wornell (2001). Learned encoders such as HiDDeN (Zhu et al., 2018), StegaStamp (Tancik, Mildenhall & Ng, 2020) and MBRS (Jia, Fang & Zhang, 2021) train embedding and extraction jointly with differentiable or real JPEG layers. We do not benchmark against these schemes: they target a joint robustness–undetectability objective and depend on trained models or cost functions whose settings would dominate any comparison, whereas the aim here is a transparent, reproducible baseline measured identically across codecs. The QIM construction presented in "A Robust Transform-Domain Alternative" thus serves as a reference point rather than a contribution; placing adaptive and learned schemes on the same bench is left for future work.

Threat model

We adopt the usual three-party setting. A sender embeds an encrypted payload in a cover image and a receiver extracts it; the image travels through a channel that applies non-adversarial distortion, lossy recompression at an unknown quality, possibly repeated, but preserves image dimensions and the 8×8 block grid. A passive warden observes the transmitted images and knows the embedding algorithm but not the key or the message. Two properties are therefore relevant: robustness to the active distortion of the channel, and undetectability by the passive warden. The experiments in this paper measure robustness. Undetectability is examined only with a targeted detector; since that detector already separates cover from stego images, the transform-domain pipeline is evaluated as an active-channel robust watermarking baseline, not as a covert communication scheme. Resizing, cropping and malicious removal attacks are outside the model.

LEAST SIGNIFICANT BIT STEGANOGRAPHY: MODEL AND IMPLEMENTATION

We view one colour channel as a matrix of integers. An eight-bit pixel p decomposes in base two as

$$p = b_7 \cdot 2^7 + b_6 \cdot 2^6 + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0, \quad b_k \in \{0, 1\}. \tag{1}$$

The term of interest is b_0, that is $2^0 = 1$. Embedding a message bit m means clearing b_0 and rewriting it. In plain arithmetic the operation is $p' = (p - p \bmod 2) + m$; in code it becomes a bit mask, $p' = (p \text{ AND } 0xFE) \text{ OR } m$, where 0xFE is 11111110 in binary. Both forms change the pixel by at most one unit.

A control experiment and two implementation traps

To keep the argument concrete we ran a control experiment (full code in Appendix A). A sentence with diacritics was hidden in an image, saved as PNG, and read back; the message returned identical to the last character. The naive implementation found in tutorials hides two traps, however, which surfaced when running it. The first is a NumPy issue: the expression pixel & ~1, intended to clear the last bit, produces the Python integer −2, and since version 2.0 NumPy refuses the operation between a uint8 and a negative integer and stops with an OverflowError; the fix is the explicit mask 0xFE. The second is subtler because it raises no error: converting text with format(ord(c), '08b') gives any character above code point 255 more than eight bits, which desynchronises the whole stream on reading. Encoding to UTF-8 byte by byte before embedding removes the problem.

How invisible the change is follows from a single figure, and the figure can be derived. The raw capacity of an RGB image of size W×H is:

$$C = 3 \cdot W \cdot H \text{ bits} \quad (\textit{one bit per channel, per pixel}), \tag{2}$$

and the embedding rate is α = L/C, with L the payload length. The per-sample error is not even constant: when the last bit is rewritten, e = p′ − p takes values in {−1, 0, +1}, staying zero when the message bit already matches and becoming ±1 otherwise. For an uncorrelated message and cover the two cases are equally likely, so the variance of the error on a loaded sample is $E[e^2] = ½$, and the mean squared error (MSE) over the image approaches

$$MSE \approx \alpha / 2, \quad \text{whence} \quad PSNR = 10 \cdot \log_{10}(255^2 / MSE). \tag{3}$$

The model predicts a peak signal-to-noise ratio (PSNR) of 64.2 dB at α = 5% and 51.6 dB at α = 90%; values near 70 dB are reached only for α ≈ 1%, below the rates tested. The measured values presented in „Results" match these predictions closely. LSB therefore performs as expected; the difficulty arises when the image passes through a lossy codec.

WHY LSB FAILS UNDER LOSSY COMPRESSION

Quantization of the discrete cosine transform in JPEG.

JPEG works on frequencies rather than pixels (Wallace, 1992; ITU-T, 1992; Pennebaker & Mitchell, 1993). The image is cut into 8×8 blocks, and each block is passed through the two-dimensional discrete cosine transform (DCT), which rewrites its 64 values as a combination of patterns from fine to coarse (Ahmed, Natarajan & Rao, 1974):

$$F(u,v) = ¼ \cdot C(u) \cdot C(v) \cdot \Sigma_x \Sigma_y f(x,y) \cdot \cos[(2x+1)u\pi/16] \cdot \cos[(2y+1)v\pi/16], \tag{4}$$

where C (0) = 1/√2 and C(k) = 1 for k ≥ 1, a normalisation without which the ¼ factor would make no sense. The transform itself is reversible; the loss comes from the next step, quantization, in which every coefficient is divided by an entry of a fixed matrix and rounded:

$$F_Q(u,v) = round\,(\,F(u,v) / Q(u,v)\,). \tag{5}$$

A claim that recurs in introductory treatments deserves correction here: that LSB changes "gather in the high-frequency corner" of the block and are cut off there. A ±1 perturbation applied at random over pixels is noise with an almost flat spectrum; it spreads across all 64 coefficients rather than settling in one corner. More importantly, the embedding perturbation is not what destroys the message: the decisive quantity is the reconstruction error of the codec itself. Because the DCT is orthonormal, Parseval's relation links the pixel-domain reconstruction error δ of a block to the coefficient-domain quantization error Δ:

$$\Sigma_{xy}\, \delta(x,y)^2 = \Sigma_{uv}\, \Delta(u,v)^2. \tag{6}$$

When Q(u,v) is small relative to the spread of the coefficient values, rounding leaves each coefficient with a quantization error of variance close to $Q(u,v)^2/12$, so the mean squared pixel error of a block is approximately

$$E[\delta^2] \approx (1/64) \cdot \Sigma_{uv}\, Q(u,v)^2 / 12, \tag{7}$$

whether or not anything was embedded. At quality 95, where the luminance steps range from 1 to 12, Eq. (7) gives a root-mean-square (RMS) error of about two grey levels, against a measured 1.6; the measured RMS error grows to 4.0 at quality 75 and 5.2 at quality 50. At these lower qualities Eq. (7) is only an upper bound, because many coefficients are rounded to zero and contribute less than $Q^2/12$. The parity of a reconstructed pixel is preserved only when its reconstruction error is even; once that error is spread over several integer values, even and odd errors become almost equally likely, and the stored bit is effectively randomised. Table 1 confirms this directly on clean, un-embedded covers: the codecs change 58 - 85% of pixel values by a mean of 1.1 - 3.8 grey levels, and only 51 - 61% of the covers' own least significant bits survive. Embedded bits fare worse than the covers' own bits, with a BER close to 50% even at quality 95, because the codec reconstructs a smoothed approximation of the cover; this partly reverses the ±1 change and returns the

Table 1. *Effect of each lossy codec on clean, un-embedded covers (means over the 103 tiles), compared with the BER of embedded LSB bits from Table 2. A preserved-LSB rate of 50% corresponds to random parity.*

Output format	Pixels changed (%)	Mean |δ| (grey levels)	Cover LSBs preserved (%)	BER of embedded bits (%)
JPEG 4:4:4, q95	57.7	1.08	61.4	49.6
JPEG 4:4:4, q90	68.8	1.80	56.9	49.9
JPEG 4:4:4, q75	77.3	2.77	53.8	50.0
JPEG 4:4:4, q50	85.0	3.60	51.6	50.0
JPEG 4:2:0, q95	63.3	1.69	60.7	49.7
WebP lossy, q95	72.7	1.69	52.2	49.9
WebP lossy, q75	82.8	3.18	51.5	50.0
JPEG XL, d = 1.0	74.9	2.20	53.2	50.0
JPEG XL, d = 3.0	82.2	3.55	51.2	50.0
Double JPEG (q90→q75)	77.9	2.83	53.6	50.0

pixel towards the cover's parity rather than the messages. The failure is therefore caused by codec reconstruction noise exceeding one grey level, not by the embedded perturbation crossing rounding thresholds.

Intra-frame prediction in WebP and other codecs

WebP, in its lossy mode, inherits the frame-coding machinery of the VP8 video standard (Bankoski, Wilkins & Xu, 2011; Google Developers, 2010). Unlike JPEG, which transforms the pixels directly, VP8 first predicts the current block from already-decoded neighbours above and to the left. Writing P(x,y) for the predicted block, the codec transforms not the pixels but the residual:

$$R(x,y) = f(x,y) - P(x,y). \quad (8)$$

The residual is then transformed and quantized, and an in-loop deblocking filter smooths block edges, so the reconstruction error of WebP is, as for JPEG, of the order of one or more grey levels (Table 1). One therefore expects WebP to be at least as hostile to LSB as JPEG. What measurement adds is that every lossy codec reaches the same floor of about 50% error, the value that corresponds to complete decorrelation, a message fully decorrelated from the original cannot be destroyed further. JPEG XL, which uses a more sophisticated transform (Alakuijala et al., 2019), and a double-recompression pipeline that re-encodes an already-compressed image meet the same fate.

EXPERIMENTAL METHOD

The statement "spatial LSB fails under compression" has a textbook answer, but that answer is almost always shown on a single image and a single codec. To test it seriously we built a factorial experiment on real photographs, with enough repetitions to attach a confidence interval to every figure.

Corpus

We started from eight real photographs chosen for variety of content and texture: a portrait (astronaut), two animals (cat, chelsea), an object on a plain background (coffee), a technical scene (rocket), a medical fundus image (retina), a microscopy image (immunohistochemistry) and an astronomical photograph (Hubble deep field), all from the scikit-image sample set (van der Walt et al., 2014). From each we extracted disjoint 128×128 tiles, capped at 16 per image so that none would dominate, yielding a corpus of 103 tiles. Tiling raises the number of carriers and includes, within a single photograph,

both smooth areas and detail-rich ones; tiles from the same photograph are, however, not independent, which is considered in the confidence intervals.

Factors and metrics

Three factors were crossed. The embedding rate α took the values 5%, 25%, 50% and 90% of capacity, with bits placed at pseudo-random rather than sequential positions. The output format covered fifteen configurations: PNG and lossless WebP as controls; JPEG at four quality levels with 4:4:4 chroma subsampling and two more with 4:2:0; lossy WebP at four levels; JPEG XL at two distances; and a double-JPEG recompression pipeline. For every combination of tile, rate and format we ran two trials with different random messages, for a total of 12,360 measurements (103 × 4 × 15 × 2). The primary metric is the bit error rate (BER), computed by direct comparison between the embedded bits and those read from the compressed file at the same positions. Alongside it we report the exact-recovery rate, the fraction of trials with zero errors, the only threshold that matters for a real message, and the PSNR of the stego image against the original as a measure of imperceptibility. Confidence intervals are obtained by an image-level (cluster) bootstrap with 5,000 resamples of the source photographs, so that tiles from the same photograph are not treated as independent samples; BER intervals are expressed in percentage points. To isolate the effect of the codec itself, every tile was also passed through each codec without any embedding (Table 1).

Encoders and reproducibility

All experiments ran on Python 3.12.3 with NumPy 2.4.4, SciPy 1.17.1, Pillow 12.1.1, scikit-image 0.26.0, reedsolo 1.7.0 and imagecodecs 2026.8.16. JPEG and JPEG XL were produced through imagecodecs, which bundles libjpeg-turbo 3.2.0 (IJG quality scaling) and libjxl 0.12.0; WebP was produced with libwebp 1.6.0 through Pillow. One trap is worth reporting because it first produced spurious results: the WebP encode function in imagecodecs (version above) ignores the quality parameter and writes losslessly by default, which misleadingly gave a zero-error rate for “lossy” WebP. We caught it by noticing that the bytes at quality 90 were identical to the lossless output, moved WebP encoding to libwebp, and verified, before the run itself, that every codec discards information when it should. All random choices use fixed seeds, so every number in this paper is deterministic, and the core code is given in Appendices A and B.

RESULTS

The overall picture fits in a single figure. Figure 1 shows the bit error rate for each format, at each embedding rate; green means the message is kept, red means it is lost.

The exact numbers, with confidence intervals, are given in Table 2.

Three things stand out. First, the lossless controls, PNG and lossless WebP, recover the message every time, with exactly zero error, no matter how much was hidden. This confirms that LSB has nothing against compression as such, only against loss; a lossless codec, even a modern one such as WebP, is perfectly transparent to the method.

Second, and most important: every lossy format, without exception, brings the message to an error rate practically indistinguishable from 50%, with zero exact recovery over several thousand trials. It does not matter whether the codec is JPEG at quality 95 or 50, whether it subsamples chroma, whether it is WebP, JPEG XL or a double recompression. All reach the same floor, because at 50% error the extracted bits are equivalent to random guessing.

Third, and the only fine signal in the whole map: at the highest JPEG quality a residual correlation appears. The BER drops slightly,

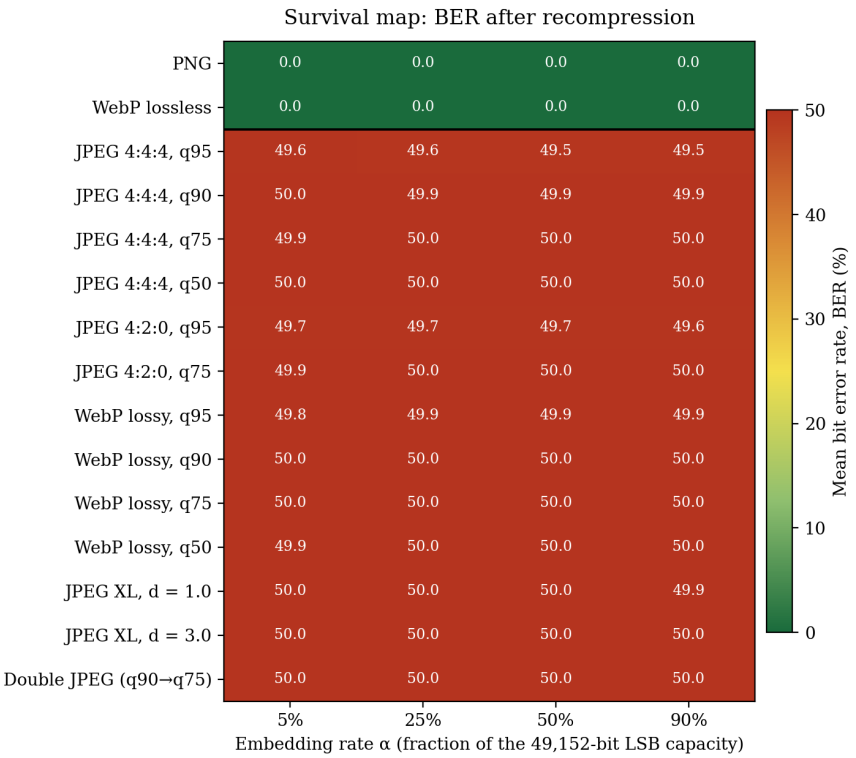


Figure 1. *Survival map. Cell values are the mean BER (%) over all tiles and trials; exact values with confidence intervals are given in Table 2. Only the two lossless rows stay green; every lossy format saturates near 50% error, independent of the embedding rate on the horizontal axis*

Table 2. *Mean BER and exact-recovery rate for all fifteen output formats, aggregated over all embedding rates, tiles and trials (824 measurements per format). The 95% confidence interval is an image-level cluster bootstrap, in percentage points of BER*

Output format	Type	Mean BER (%)	95% CI (BER, %)	Exact recovery (%)
PNG	lossless	0.00	[0.00, 0.00]	100
WebP lossless	lossless	0.00	[0.00, 0.00]	100
JPEG 4:4:4, q95	lossy	49.57	[49.24, 49.88]	0
JPEG 4:4:4, q90	lossy	49.90	[49.80, 50.00]	0
JPEG 4:4:4, q75	lossy	49.96	[49.93, 49.99]	0
JPEG 4:4:4, q50	lossy	50.01	[49.97, 50.05]	0
JPEG 4:2:0, q95	lossy	49.69	[49.45, 49.91]	0
JPEG 4:2:0, q75	lossy	49.97	[49.95, 50.00]	0
WebP lossy, q95	lossy	49.89	[49.81, 49.99]	0
WebP lossy, q90	lossy	49.99	[49.94, 50.04]	0
WebP lossy, q75	lossy	49.99	[49.98, 50.01]	0
WebP lossy, q50	lossy	49.98	[49.94, 50.02]	0
JPEG XL, d = 1.0	lossy	49.98	[49.94, 50.03]	0
JPEG XL, d = 3.0	lossy	50.02	[49.99, 50.05]	0
Double JPEG (q90→q75)	pipeline	50.01	[49.98, 50.04]	0

but with statistical certainty, below 50% - 49.6% at 4:4:4 q95, with an image-level 95% interval of 49.2 - 49.9% that does not reach one half. This is explained in “Why LSB Fails Under Lossy Compression”: in very smooth regions the reconstructed pixel is sometimes identical to the original, and its last bit survives by chance. The effect depends on content, as it should (Figure 2).

The correlation is real but small, reaching at most about one percentage point, and offers no means of recovering a message: an error-correcting code cannot reconstruct information from a channel at 49% error, as shown in “A Robust Transform-Domain Alternative”. This is consistent with the mechanism described in “Why LSB Fails Under Lossy Compression”, but does not provide a practical channel. The embedding rate, in turn, does nothing for survival: the rows of the map are flat, meaning a sparsely hidden message is as destroyed as one that fills the image. The only thing the rate changes is imperceptibility, measured by PSNR, and their theory and experiment meet again (Figure 3): the measured values follow Eq. (3).

A ROBUST TRANSFORM-DOMAIN ALTERNATIVE

Theoretical basis

If the image must pass through a lossy codec anyway, as in any channel that re-

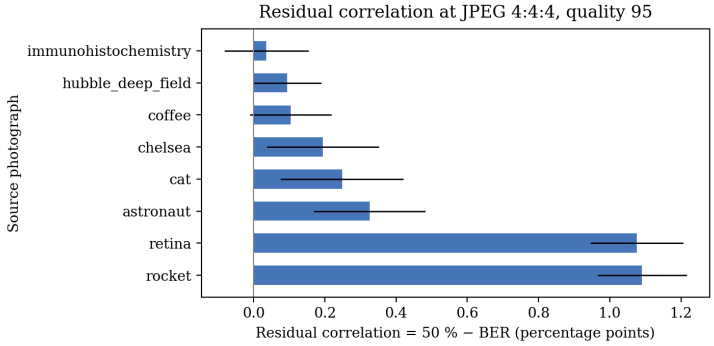


Figure 2. *Residual correlation at JPEG 4:4:4 q95, per source photograph. The horizontal axis is 50% minus the mean BER, in percentage points (0 = no surviving information); error bars are 95% intervals over that photograph’s tiles and trials. Images with large smooth surfaces (rocket, retina) keep the most; a noisy microscopy image keeps nothing.*

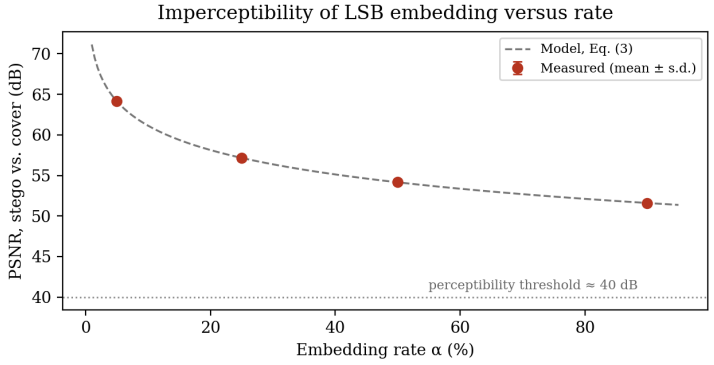


Figure 3. *PSNR of the stego image against the embedding rate: measured values (points, mean ± standard deviation over tiles) and the prediction of Eq. (3) (dashed line). PSNR falls from 64.2 dB at 5% to 51.6 dB at 90%, staying well above the perceptibility threshold throughout.*

encodes images, the answer is not to harden spatial LSB but to stop using it in its classic form. The most direct route is to move the information into the transform domain: instead of touching pixels, one modifies the DCT coefficients that JPEG preserves best, so the hidden message is expressed in the codec's own representation (Westfeld, 2001; Zhang et al., 2018). Redundancy is the second ingredient, and its limit is not engineering but information theory. A channel that flips each bit with probability p, a binary symmetric channel, has capacity:

$$C = 1 - H_2(p), \qquad H_2(p) = -p \cdot log_2 p - (1-p) \cdot log_2 (1-p), \tag{9}$$

(Shannon, 1948; Cover & Thomas, 2006). Below this capacity there exist codes that recover the message; above it, none. For the p ≈ 0.50 measured at every lossy format the capacity is essentially zero: it is not that a Reed-Solomon (RS) code would be too weak, but that there is nothing left to recover. If instead one first moves to the transform domain and brings the error rate down to a few percent, the capacity jumps above 0.8 bits per use and redundancy suddenly becomes enough. Order matters: change the domain first, then add error-correcting codes (Hamming, 1950; Reed & Solomon, 1960; MacWilliams & Sloane, 1977). Whatever the method, the payload should also be encrypted beforehand, with the Advanced Encryption Standard, for example (National Institute of Standards and Technology, 2001), not to help it survive, but so that if it is extracted it means nothing without the key. The argument so far is theoretical; the next subsection turns it into code and numbers.

Method, parameter selection and results

Architectural assumption. The scheme relies on block synchronization: the encoder, the channel codec and the decoder must all use the same 8×8 partition of the image. This holds for recompression of an image whose dimensions are preserved, as specified in the "Threat model" section, but any resizing, rotation or cropping by an offset that is not a multiple of eight breaks it, and a synchronization stage would then be required. All results below are conditional on this assumption.

The method is quantization index modulation (QIM), a standard construction used here as a transparent reference rather than as a new technique: pixels are left alone and each bit is written by choosing which lattice of values a DCT coefficient falls on (Chen & Wornell, 2001). For a step Δ, a bit 0 pushes the coefficient to the nearest multiple of Δ and a bit 1 to the nearest multiple offsets by Δ/2; at decoding one simply reads which lattice the value is closer to. As long as the perturbation introduced by the codec stays below Δ/4 the bit is read correctly, which is precisely why the method survives where LSB, dependent on sub-unit perturbations, collapses. It is a quantization counterpart to classic spread-spectrum watermarking (Cox et al., 1997).

We chose three low-frequency coefficients from each 8×8 block, on all three channels, because low frequencies are quantized most gently by JPEG. To avoid tuning on the evaluation data, the corpus was split by source photograph: three photographs (cat, coffee, and Hubble deep field; 34 tiles) form a calibration split and the remaining five (69 tiles) a test split. Δ was selected on the calibration split alone, with a criterion fixed in advance, the smallest step, hence the highest PSNR, that recovers at least 90% of messages at JPEG quality 60. This yields Δ = 56 (94.1% recovery on the calibration split); Table 3 reports the full sensitivity sweep over $\Delta \in \{32, 40, 48, 56, 72\}$ on both splits. All results below are on the test split.

Over the embedded bits we placed a Reed-Solomon code RS (200,100) over 8-bit symbols

Table 3. *Sensitivity of the RGB QIM scheme to the step Δ (JPEG 4:4:4). Recovery is the percentage of tiles whose 100-byte message is decoded exactly; the selected value is marked with an asterisk.*

Δ	PSNR, test (dB)	Calibration recovery, q60 (%)	Test, q75 (%)	Test, q70 (%)	Test, q60 (%)	Test, q50 (%)
32	43.3	0.0	0.0	0.0	0.0	0.0
40	41.4	0.0	85.5	0.0	0.0	0.0
48	39.8	0.0	87.0	87.0	0.0	0.0
56 *	38.4	94.1	87.0	87.0	81.9	0.0
72	36.2	100.0	87.0	87.0	87.0	84.1

(100 parity bytes for 100 message bytes), correcting up to 50 erroneous bytes in a 200-byte block. Capacity must be stated at two levels. The raw channel offers 2,304 carrier coefficients per 128×128 tile, 4.7% of the 49,152 bits available to spatial LSB; one RS (200,100) block occupies 1,600 of them, so the net payload is 800 bits per tile, or 1.6% of spatial LSB capacity. Only the 1,600 carriers that receive a code bit are modified; the remaining 704 are left untouched rather than padded. The PSNR on the test split is 38.4 dB, against 66.0 dB for spatial LSB at the same 1,600-bit payload ($\alpha \approx 3.3\%$) and 51.6 dB for LSB at $\alpha = 90\%$. Robustness is not free; it is paid for in capacity and in visual quality.

The measured result on the test split, passed through JPEG 4:4:4 at qualities from 95 down to 40, is given in Figure 4.

In numbers: the raw error of the QIM channel is 2.0% at quality 95 and stays at or below 3.3% down to quality 70, then rises to 6.5% at 60 and 22.3% at 50. Reed-Solomon turns that small error into exact message recovery for 87.0% of test tiles at every quality from 95 down to 70 (image-level 95% interval 73.8–100%) and for 81.9% at quality 60, after which, at quality 50 and below, the error rate exceeds what the code can correct and recovery drops to zero. Spatial LSB recovers no message at any quality. Table 4 summarizes the trade-off.

Recovery does not exceed 87% at any quality. The nine test tiles that fail at every quality (three from astronaut, six from retina) are dominated by near-black pixels: re-quantizing their low-frequency coefficients pushes 10–40% of pixel values (19.4% on average) outside the range [0, 255], and the subsequent clipping

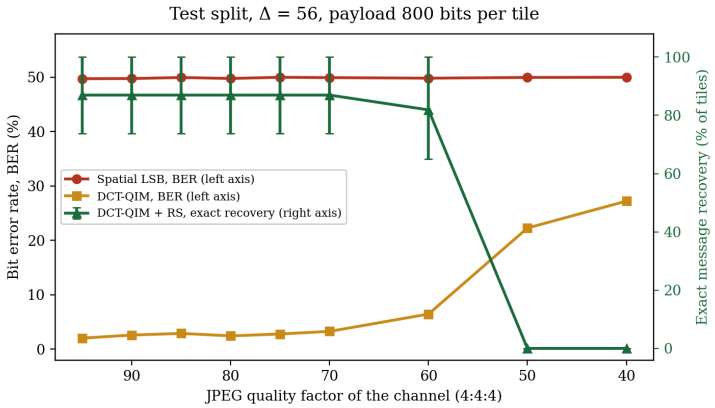


Figure 4. *Test split, Δ = 56. Horizontal axis: JPEG quality factor of the channel (decreasing to the right). Left axis: mean bit error rate (%) of spatial LSB (red) and of the raw QIM channel (gold), both at a 1,600-bit payload. Right axis: percentage of tiles whose message is recovered exactly after RS decoding (green), with image-level 95% bootstrap intervals.*

erases the embedded lattice positions, whereas in the 60 recovered tiles clipping affects 0.3% of pixels on average. Failure is thus governed by saturation rather than by texture. Excluding saturated blocks from the carrier set or compressing the dynamic range of the cover before embedding, would address it.

Chroma subsampling and a luminance-domain variant

The results above use 4:4:4 encoding. Under 4:2:0 chroma subsampling, the default of many encoders, the RGB scheme fails: on the test split its BER rises to 10.5%, 23.7% and 30.9% at qualities 95, 90 and 75, and no message is recovered, because subsampling averages the chroma components on which the RGB carriers partly depend. Embedding only in the luminance (Y) component avoids this. We used six low-frequency luminance carriers per block (1,536 channel bits per tile, 3.1% of LSB capacity) and RS (192,96), which gives 768 net bits per tile (1.6%). The same calibration criterion selects $\Delta = 24$. On the test split, this luminance variant reaches 41.2 dB PSNR and recovers 87.0 - 88.4% of messages under 4:2:0 at every quality from 95 down to 60 (87.0 - 89.9% under 4:4:4), 24.6% at quality 50 and none at 40; its failures are the same nine saturated tiles.

Detectability

Robustness is only one of the two requirements set out in the "Threat model" section. We did not evaluate the schemes against standard steganalysers such as SRM, JRM or SRNet. To bound the problem, we applied a simple targeted detector that knows the algorithm, the carrier positions and Δ, but not the message: the mean distance of the carrier coefficients to the union lattice $(\Delta/2)\,\mathbb{Z}$, normalised by $\Delta/4$. For cover images this distance is spread out, whereas QIM concentrates it near zero. On all 103 tiles, this single statistic separates cover from stego images with an area under the receiver operating characteristic curve (AUC) of 0.93 for the RGB variant (0.78 after a JPEG q75 pass) and 0.94 for the luminance variant. Lattice-aligned coefficients and PSNR values of 38–41 dB would be still easier targets for rich-model or deep steganalysers, and spatial LSB replacement is in turn reliably detected by structural methods such as RS analysis (Fridrich, Goljan & Du, 2001). Accordingly, the transform-domain pipeline should be read as an active-channel robust watermarking baseline rather than as a covert communication scheme; making it undetectable would require content-adaptive costs and syndrome coding of the kind reviewed in the "Related work" section.

Table 4. *Spatial LSB against the two QIM variants (test split for QIM). Capacities are per 128×128 tile; percentages are relative to the 49,152-bit spatial LSB capacity.*

Property	Spatial LSB	QIM, RGB ($\Delta = 56$)	QIM, luminance ($\Delta = 24$)
Embedding domain	pixels (spatial)	DCT, R, G, B	DCT, luminance
Raw channel capacity	49,152 bits	2,304 bits (4.7 %)	1,536 bits (3.1 %)
Net payload after RS	— (no coding)	800 bits (1.6 %)	768 bits (1.6 %)
PSNR, stego vs. cover	66.0 dB at 1,600 bits ($\alpha \approx 3.3$ %); 51.6 dB at $\alpha = 90$ %	38.4 dB	41.2 dB
Raw BER, JPEG 4:4:4 q70	≈ 50 %	3.3 %	1.3 %
Exact recovery, 4:4:4, q95–q70	0 %	87.0 %	87.0–89.9 %
Exact recovery, 4:4:4, q60	0 %	81.9 %	89.9 %
Exact recovery, 4:2:0, q95–q60	0 %	0 %	87.0–88.4 %
Collapse threshold	— (always lost)	$q \le 50$	$q \le 50$ (24.6 % at q50)
Targeted detector AUC	detectable by RS analysis	0.93	0.94

LIMITATIONS

A few limits are worth stating plainly. The corpus is built from real photographs, but from only eight sources, tiled; it spans a good range of textures yet does not replace a set of thousands of independent images, and the exact values of the residual correlation could vary on a larger corpus. The confidence intervals are computed at image level for this reason, but with eight clusters (five in the test split) they remain approximate. We used the encoders from libjpeg-turbo, libwebp and the reference JPEG XL codec; other implementations have slightly different quantization matrices and filters, so the figures should be read as orders of magnitude, though the fact that three independent codec implementations, in thirteen lossy configurations, reach the same 50% floor suggests the verdict is robust to the choice of encoder. We measured recompression and double recompression, but not resizing or cropping, which destroy LSB by other routes and would deserve a separate study.

The robust baseline presented in "A robust transform-domain alternative" also has its own limitations. As discussed in "Method, parameter selection and results", it assumes block synchronisation and has been tested only against recompression. Real image-sharing pipelines commonly combine recompression with spatial downsampling and metadata stripping; we did not measure any specific platform, make no claim about the quality's platforms use, and note that downsampling alone would defeat both the LSB and the QIM pipelines as implemented here. Survival stops around quality 50; recovery fails on saturated tiles; and detectability was examined only with a targeted detector, not with SRM, JRM or SRNet. Known remedies, grid synchronisation, saturation-aware carrier selection, interleaving, adaptive costs with syndrome coding, are left for future work. None of this unsettles the two central results: loss breaks spatial LSB completely and regardless of quality, and moving to the transform domain, with redundancy, recovers the message over a useful range of qualities when the block grid is preserved.

CONCLUSIONS

Spatial LSB remains a good choice when the data path is clean from end to end: a PNG file kept locally, an archive that is not recompressed, a channel that guarantees bit-for-bit transport. The moment the image enters a path where it will be re-encoded, the classic method becomes unusable, and the 12,360 measurements reported here show how uniformly: fifteen format variants, four embedding rates, eight photographs, and a single verdict for everything that means lossy compression, explained by the codecs' own reconstruction error rather than by the embedding itself. A standard transform-domain construction measured on the same bench, with parameters selected on held-out images, recovers 87% of test messages through JPEG down to quality 70 and 82% at quality 60, and its luminance variant does so under 4:2:0 subsampling as well, provided the block grid is preserved, and at a cost of capacity (1.6% net) and imperceptibility (38 - 41 dB). LSB therefore fails on lossy channels not because robust embedding is impossible, but because it is the wrong tool for them. Because undetectability was not established, the robust scheme is a watermarking baseline for active channels rather than a covert channel. Grid synchronisation, saturation-aware carrier selection, evaluation against SRM, JRM and SRNet, and benchmarking of adaptive and learned schemes on the same bench are the directions for future work.

ACKNOWLEDGEMENTS

The experiments rely on open-source software, NumPy, SciPy, Pillow, scikit-image, imagecodecs, Matplotlib and the reedsolo library, whose maintainers we thank.

REFERENCE LIST

Ahmed, N., Natarajan, T. & Rao, K. R. (1974) Discrete cosine transform. *IEEE Transactions on Computers*. C-23(1), 90–93.

Alakuijala, J., van Asseldonk, R. & Boukortt, S. et al. (2019) JPEG XL next-generation image compression architecture and coding tools. In: *Applications of Digital Image Processing XLII*. Proceedings of SPIE 11137. Bellingham, SPIE.

Bankoski, J., Wilkins, P. & Xu, Y. (2011) *VP8 data format and decoding guide*. RFC 6386. Internet Engineering Task Force. https://www.rfc-editor.org/info/rfc6386 [Accessed 3 September 2026].

Boroumand, M., Chen, M. & Fridrich, J. (2019) Deep residual network for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*. 14(5), 1181–1193.

Chandramouli, R. & Memon, N. (2001) Analysis of LSB based image steganography techniques. In: *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. 3, 1019–1022.

Chen, B. & Wornell, G. W. (2001) Quantization index modulation: a class of provably good methods for digital watermarking and information embedding. *IEEE Transactions on Information Theory*. 47(4), 1423–1443.

Cheddad, A., Condell, J. & Curran, K. et al. (2010) Digital image steganography: survey and analysis of current methods. *Signal Processing*. 90(3), 727–752.

Cover, T. M. & Thomas, J. A. (2006) *Elements of information theory*. 2nd edn. Hoboken, Wiley-Interscience.

Cox, I. J., Kilian, J. & Leighton, F. T. et al. (1997) Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing*. 6(12), 1673–1687.

Demircan, Y. Y. & Ozekes, S. (2024) A novel LSB steganography technique using image segmentation. *Journal of Universal Computer Science*. 30(3), 308–332.

Filler, T., Judas, J. & Fridrich, J. (2011) Minimizing additive distortion in steganography using syndrome-trellis codes. *IEEE Transactions on Information Forensics and Security*. 6(3), 920–935.

Fridrich, J. (2009) *Steganography in digital media: principles, algorithms, and applications*. Cambridge, Cambridge University Press.

Fridrich, J., Goljan, M. & Du, R. (2001) Reliable detection of LSB steganography in color and grayscale images. In: *Proceedings of the ACM workshop on multimedia and security*. New York, ACM, pp. 27–30.

Fridrich, J. & Kodovský, J. (2012) Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*. 7(3), 868–882.

Google Developers (2010) *WebP compression techniques*. https://developers.google.com/speed/webp [Accessed 3 September 2026].

Hamming, R. W. (1950) Error detecting and error correcting codes. *Bell System Technical Journal*. 29(2), 147–160.

Holub, V., Fridrich, J. & Denemark, T. (2014) Universal distortion function for steganography in an arbitrary domain. *EURASIP Journal on Information Security*. 2014, 1.

ITU-T (1992) *Recommendation T.81: digital compression and coding of continuous-tone still images*. Geneva, International Telecommunication Union.

Jia, Z., Fang, H. & Zhang, W. (2021) MBRS: enhancing robustness of DNN-based watermarking by mini-batch of real and simulated JPEG compression. In: *Proceedings of the 29th ACM International Conference on Multimedia*. New York, ACM, pp. 41–49.

Ker, A. D. (2005) Steganalysis of LSB matching in grayscale images. *IEEE Signal Processing Letters*. 12(6), 441–444.

Kin-Cleaves, C. & Ker, A. D. (2018) Adaptive steganography in the noisy channel with dual-syndrome trellis codes. In: *Proceedings of the IEEE International Workshop on Information Forensics and Security (WIFS 2018)*. Hong Kong, IEEE.

Kodovský, J. & Fridrich, J. (2012) Steganalysis of JPEG images using rich models. In: *Media Watermarking, Security, and Forensics 2012*. Proceedings of SPIE 8303. Bellingham, SPIE, 83030A.

MacWilliams, F. J. & Sloane, N. J. A. (1977) *The theory of error-correcting codes*. Amsterdam, North-Holland.

National Institute of Standards and Technology (2001) *Advanced encryption standard (AES)*. FIPS PUB 197. Gaithersburg, NIST.

Pennebaker, W. B. & Mitchell, J. L. (1993) *JPEG: still image data compression standard*. New York, Springer.

Pevný, T., Filler, T. & Bas, P. (2010) Using high-dimensional image models to perform highly undetectable steganography. In: *Information hiding*. LNCS 6387. Berlin, Springer, pp. 161–177.

Provos, N. & Honeyman, P. (2003) Hide and seek: an introduction to steganography. *IEEE Security & Privacy*. 1(3), 32–44.

Reed, I. S. & Solomon, G. (1960) Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*. 8(2), 300–304.

Shannon, C. E. (1948) A mathematical theory of communication. *Bell System Technical Journal*. 27, 379–423 & 623–656.

Tancik, M., Mildenhall, B. & Ng, R. (2020) StegaStamp: invisible hyperlinks in physical photographs. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 2117–2126.

Tataru, R., Vlad, A. & Gugu, C. (2015) Is hidden data safe? Analysis of the public Crypt&Hide-Steganos application. *Proceedings of the Romanian Academy, Series A*. 16 (Special Issue), 299–312.
van der Walt, S., Schönberger, J. L. & Nunez-Iglesias, J. et al. (2014) scikit-image: image processing in Python. *PeerJ*. 2, e453.
Wallace, G. K. (1992) The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*. 38(1), xviii–xxxiv.
Westfeld, A. (2001) F5—a steganographic algorithm: high capacity despite better steganalysis. In: *Information hiding*. LNCS 2137. Berlin, Springer, pp. 289–302.
Zhang, Y., Zhu, X. & Qin, C. et al. (2018) Dither modulation based adaptive steganography resisting JPEG compression and statistic detection. *Multimedia Tools and Applications*. 77, 17913–17935.
Zhao, Z., Guan, Q. & Zhang, H. et al. (2019) Improving the robustness of adaptive steganographic algorithms based on transport channel matching. *IEEE Transactions on Information Forensics and Security*. 14(7), 1843–1856.
Zhu, J., Kaplan, R. & Johnson, J. et al. (2018) HiDDeN: hiding data with deep networks. In: *Computer Vision – ECCV 2018*. Cham, Springer, pp. 657–672. https://doi.org/10.1007/978-3-030-01267-0_40.

APPENDIX A. EMBEDDING AND MEASUREMENT CODE

The reproducible core: LSB embedding, extraction, and the BER and PSNR functions used throughout. It masks with 0xFE, encodes text to UTF-8, aand was run with the software versions listed in “Encoders and reproducibility”.

```
# -*- coding: utf-8 -*-
from PIL import Image
import numpy as np

DELIM = “###END###”

def text_to_bits(text: str) -> str:
    # Encode as UTF-8 first: each byte -> 8 bits.
    # This keeps synchronisation even for non-ASCII characters (ord > 255).
    return ‹’.join(format(byte, ‘08b’) for byte in text.encode(‘utf-8’))

def hide(cover_path, message, out_path):
    arr = np.array(Image.open(cover_path).convert(‘RGB’))
    flat = arr.flatten()
    payload = text_to_bits(message + DELIM)
    if len(payload) > flat.size:
        raise ValueError(“Payload larger than image capacity.”)
    for i, b in enumerate(payload):
        # 0xFE = 11111110. Use 0xFE, NOT ~1:
        # ~1 is -2 in Python, and numpy >= 2.0 aborts the op on a uint8.
        flat[i] = (flat[i] & 0xFE) | int(b)
    Image.fromarray(flat.reshape(arr.shape), ‘RGB’).save(out_path)
    return payload   # reference payload for the BER measurement

def extract(path, max_chars=4000):
    flat = np.array(Image.open(path).convert(‘RGB’)).flatten()
    bits = ‹’.join(str(int(v) & 1) for v in flat)
    out = bytearray()
    for i in range(0, min(len(bits), max_chars * 8), 8):
        out.append(int(bits[i:i+8], 2))
        if out[-len(DELIM):].decode(‘latin-1’, ‘ignore’) == DELIM:
            return out[:-len(DELIM)].decode(‘utf-8’, ‘replace’), True
    return out.decode(‘utf-8’, ‘replace’), False

def ber(reference_bits, path):
    flat = np.array(Image.open(path).convert(‘RGB’)).flatten()
    got = ‹’.join(str(int(v) & 1) for v in flat[:len(reference_bits)])
    diff = sum(1 for a, b in zip(reference_bits, got) if a != b)
    return diff / len(reference_bits)

def psnr(a_path, b_path):
    a = np.array(Image.open(a_path).convert(‘RGB’)).astype(np.float64)
    b = np.array(Image.open(b_path).convert(‘RGB’)).astype(np.float64)
    mse = np.mean((a - b) ** 2)
    return float(‹inf›) if mse == 0 else 10 * np.log10(255.0**2 / mse)
```

APPENDIX B. ROBUST BASELINE: DCT-QIM WITH REED-SOLOMON

The RGB scheme from “Method, parameter selection and results”: QIM embedding on three low-frequency DCT carriers per 8×8 block with step Δ = 56, protected by RS(200,100). Only carriers that receive a code bit are modified. The example uses a fixed seed and asserts exact recovery after JPEG compression at quality 75 with 4:4:4 subsampling; under 4:2:0 the luminance-domain variant should be used.

```
# DCT-QIM with Reed-Solomon: robust embedding against JPEG recompression.
# Assumes encoder and decoder share the same 8x8 block grid (no resizing/cropping).
import numpy as np, io, reedsolo
from scipy.fft import dctn, idctn
from PIL import Image

CARRIERS = [(1,1), (1,2), (2,1)]   # low-frequency DCT carriers per 8x8 block
DELTA    = 56                      # QIM step, selected on the calibration split

def to_blocks(ch):
    H, W = ch.shape
    return ch.reshape(H//8, 8, W//8, 8).transpose(0, 2, 1, 3)
def from_blocks(b):
    by, bx, _, _ = b.shape
    return b.transpose(0, 2, 1, 3).reshape(by*8, bx*8)
def dct2(b):  return dctn(b, axes=(-2, -1), norm='ortho')
def idct2(b): return idctn(b, axes=(-2, -1), norm='ortho')
def q0(x, d): return np.round(x/d) * d                # lattice for bit 0
def q1(x, d): return (np.round(x/d - 0.5) + 0.5) * d   # lattice for bit 1

def embed(img, bits, d=DELTA):
    """Only carriers that receive a bit are modified; unused slots stay untouched."""
    out = img.astype(np.float64).copy(); idx = 0; n = len(bits)
    for c in range(3):
        if idx >= n: break
        blk = dct2(to_blocks(out[:, :, c])); by, bx = blk.shape[:2]
        for (u, v) in CARRIERS:
            if idx >= n: break
            k = min(by*bx, n - idx); co = blk[:, :, u, v].reshape(-1)
            m = bits[idx:idx+k]
            co[:k] = np.where(m == 0, q0(co[:k], d), q1(co[:k], d))
            blk[:, :, u, v] = co.reshape(by, bx); idx += k
        out[:, :, c] = from_blocks(idct2(blk))
    return np.clip(np.round(out), 0, 255).astype(np.uint8)

def extract(img, n, d=DELTA):
    x = img.astype(np.float64); vals = []
    for c in range(3):
        blk = dct2(to_blocks(x[:, :, c]))
        for (u, v) in CARRIERS:
            vals.append(blk[:, :, u, v].reshape(-1))
    y = np.concatenate(vals)[:n]
    return (np.abs(y - q1(y, d)) < np.abs(y - q0(y, d))).astype(np.uint8)

# --- example: 100-byte message, RS(200,100), JPEG q75 with 4:4:4 subsampling ---
rng  = np.random.default_rng(2026)                # fixed seed for reproducibility
rs   = reedsolo.RSCodec(100)          # 100 parity bytes -> corrects up to 50
msg  = bytes(rng.integers(0, 256, 100, dtype=np.uint8))
bits = np.unpackbits(np.frombuffer(bytes(rs.encode(msg)), np.uint8))  # 1,600 bits
cover = np.array(Image.open("cover.png").convert("RGB"))   # H, W multiples of 8
stego = embed(cover, bits)
buf = io.BytesIO()
Image.fromarray(stego).save(buf, "JPEG", quality=75, subsampling=0)  # 4:4:4
received = np.array(Image.open(io.BytesIO(buf.getvalue())).convert("RGB"))
recovered = bytes(rs.decode(bytes(np.packbits(extract(received, len(bits))))))[0])
assert recovered == msg
```



This is an open access article distributed under the terms and conditions of the
Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.