

Supplementary Information for “Affine Encryption and Adaptive Single-Neighbor Prediction for High-Capacity Reversible Data Hiding in Encrypted Images”

Ngoc Hung Nguyen^{1*} and Van At Pham²

^{1*}Department of Telematics, Institute of Information Technology,
Vietnam Academy of Science and Technology, No. 18 Hoang Quoc Viet,
Hanoi, Vietnam.

²Faculty of Information Technology, University of Economics –
Technology for Industries, Hanoi, Vietnam.

*Corresponding author(s). E-mail(s): nnhung@ioit.ac.vn;
Contributing authors: phamvanat83@gmail.com;

This document collects the per-image measurements, the encoder pseudocode, and the settings needed to reproduce the experiments of the main paper. All numbers were produced by the same code and under the same conditions as the averages reported there. Section and equation numbers of the form “Section 3.2” refer to the main paper; tables, figures, and equations numbered S1, S2, ... belong to this document.

1 Experimental settings

1.1 Image sets

Four sets of 8-bit grayscale images of size 512×512 are used.

- **Standard (12 images).** Airplane, Baboon, Barbara, Boat, Cameraman, Car, Forest, Goldhill, Pepper, Sailboat, Things, and Tiffany, from the USC-SIPI collection; they are shown in Figure S1.
- **General (100 images each).** BOSSbase, BOWS2, and UCID. For each set, the files are sorted by name and 100 of them are sampled without replacement using the NumPy PCG64 generator seeded with 2024 (`rng.choice(n, size=100,`

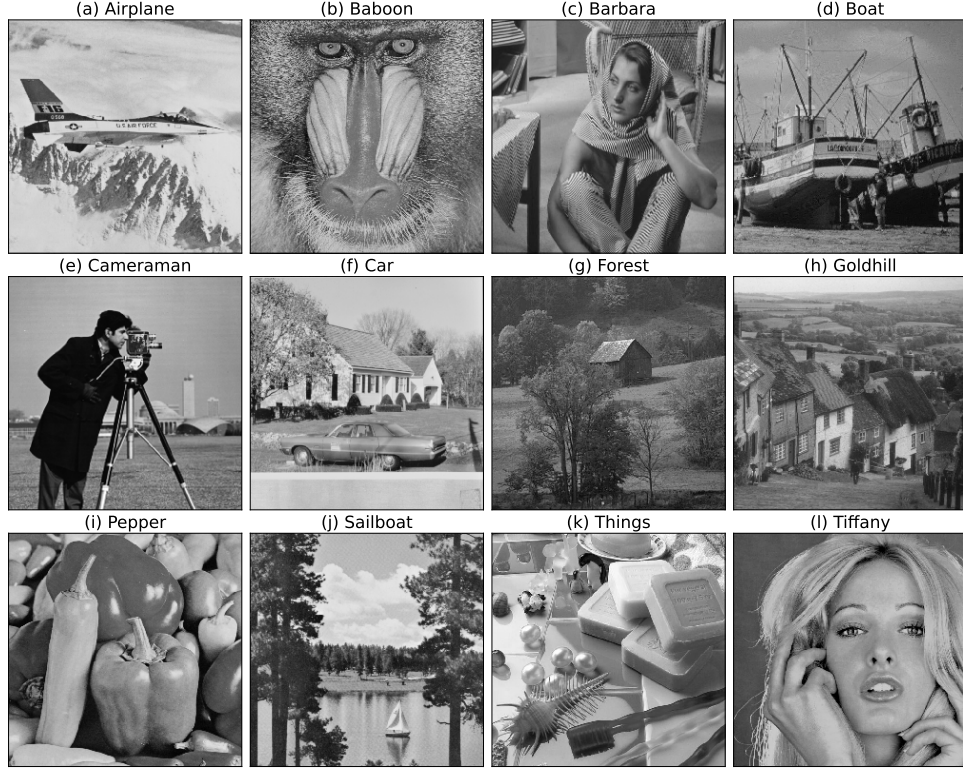


Fig. S1 The 12 standard test images, all 512×512 and 8-bit grayscale. The set mixes smooth content (for example Tiffany and Airplane) with strongly textured content (Baboon, Barbara, and Things), which is why the gain of the adaptive reference selection varies so much across the set; the per-image rates are given in Table S4.

`replace=False`)), so the selection is reproducible; the exact file names are given in the accompanying CSV files.

All images are used as 8-bit grayscale of size 512×512 . The standard images and the BOSSbase and BOWS2 images are already 8-bit grayscale of that size and are used unchanged. A colour source (as in UCID) is first converted to 8-bit luminance by the standard Rec. 601 weighting $Y = 0.299R + 0.587G + 0.114B$ and, if its size differs, centre-cropped to a square and resized to 512×512 ; no other normalization or enhancement is applied. All datasets are stored as 8-bit grayscale BMP before the experiments so that every method reads identical inputs.

1.2 Parameters

All experiments use the block sizes $k_1 = 4$ and $k_2 = 2$, the values reported as optimal for the baseline. All methods are evaluated on the same source images and parameter settings, with fixed documented seeds so that every run is reproducible; the encryption

itself differs between methods, so each operates on its own encrypted image. Each method embeds the longest prefix of the same pseudo-random payload stream that fits its available net room.

Every reported rate is a *net* rate: the auxiliary data that a scheme needs for a lossless recovery, such as its exception map, location map, or correction information, is subtracted from the freed room before dividing by $M \times N$. Every result was verified to be lossless by an exact comparison between the recovered image and the original, so the MSE is zero and the PSNR is infinite in all cases.

1.3 Evaluation metrics

For two images U and V of size $M \times N$, the mean squared error and the peak signal-to-noise ratio are

$$\begin{aligned} \text{MSE} &= \frac{1}{MN} \sum_{u=1}^M \sum_{v=1}^N (U(u, v) - V(u, v))^2, \\ \text{PSNR} &= 10 \log_{10} \frac{255^2}{\text{MSE}} \quad (\text{dB}). \end{aligned} \quad (\text{S1})$$

The information entropy of an image is

$$H = - \sum_{\ell=0}^{255} p(\ell) \log_2 p(\ell), \quad (\text{S2})$$

where $p(\ell)$ is the probability of the gray level ℓ ; the ideal value for an 8-bit image is 8. For the sensitivity of the encryption to a change of one pixel or one key bit, the number of pixels change rate and the unified average changing intensity between two encrypted images C_1 and C_2 are

$$\begin{aligned} \text{NPCR} &= \frac{1}{MN} \sum_{u,v} D(u, v) \times 100\%, \\ \text{UACI} &= \frac{1}{MN} \sum_{u,v} \frac{|C_1(u, v) - C_2(u, v)|}{255} \times 100\%, \end{aligned} \quad (\text{S3})$$

where $D(u, v) = 0$ if $C_1(u, v) = C_2(u, v)$ and $D(u, v) = 1$ otherwise. The ideal values for a 256-level image are 99.609% for NPCR and 33.464% for UACI.

1.4 Prediction-error encoding and the exception map

HBP-RMC stores each prediction error $e(p)$ of the main paper as one byte in sign-magnitude form. Writing s for the sign bit and m for a seven-bit magnitude, a value

$e \in [-127, 127]$ is encoded as

$$\text{byte}(e) = \begin{cases} (0 \ll 7) \mid e, & 0 \leq e \leq 127, \\ (1 \ll 7) \mid (-e), & -127 \leq e \leq -1, \end{cases} \quad (\text{S4})$$

so the magnitude $|e| \leq 127$ occupies the seven low bits and the sign occupies the most significant bit. Zero has the single representation $\text{byte}(0) = 0$. This covers every value in $[-127, 127]$, which is the full range of smod_{256} except the endpoint -128 : its magnitude 128 does not fit in seven bits. We therefore never encode -128 as a byte; instead the corresponding pixel is flagged in the sparse exception map EM and its encrypted value is stored directly, exactly as described in Section 3.3 of the main paper. Because $e(p) = -128$ occurs for only about 0.004% of the pixels on the standard images, EM compresses to a negligible number of bits.

1.5 Side-information budget

Table S1 decomposes the net rate into the HBP-RMC room and every auxiliary component that is subtracted from it, averaged over the 12 standard images, in bits per pixel so that the entries are directly comparable to the net rates of the main paper. Both methods are accounted for under the same convention. The first row is the room after the labels of the *embeddable* sub-blocks have been paid, that is, each embeddable sub-block contributes its freed high planes minus its three-bit label. The label bits that HBP-RMC writes into the *non-embeddable* (Type-0) sub-blocks, whose overwritten original bits must be stored to remain lossless, are then deducted as a separate row for both methods: 0.109 bpp for the baseline and 0.053 bpp for the proposed method, which has fewer Type-0 sub-blocks on this set. Only the components marked as deducted are subtracted; the raw reference-index cost is listed for comparison but is not deducted a second time.

Two effects stand out. First, the affine encryption removes the location map: the baseline spends 0.036 bpp on it, the proposed method zero. Second, the reference indices of the adaptive selection cost 0.250 bpp raw; after the block permutation their empirical distribution over the 16 positions is close to uniform, so arithmetic coding compresses them only to 0.245 bpp (98% of the raw size). The adaptive reference is therefore worthwhile only because it raises the room by far more than this index cost: the room after embeddable-label placement is 1.783 bpp for the proposed method against the baseline’s 1.429, and the net rate stays higher (1.484 against 1.284) after all deducted auxiliary data is subtracted. The exception map of the proposed method is the whole exception package, that is, the compressed bitmap of positions with $e(p) = -128$ together with the directly stored encrypted values at those positions; it costs about 0.001 bpp. Because each component is averaged over the images and rounded independently, subtracting the three-decimal values shown may not reproduce the net rate to the last digit.

Table S1 Side-information budget (bits per pixel, mean over the 12 standard images), under the same accounting convention for both methods. Only the components marked “deducted” are subtracted to give the net rate; the raw reference-index cost is shown for comparison only.

Component	Chai	Proposed	Deducted?
HBP-RMC room (embeddable labels paid)	1.429	1.783	—
Location map Φ	0.036	0	yes
Exception map and direct values	0.001	0.001	yes
Reference indices (raw)	0	0.250	no (info.)
Reference indices (compressed)	0	0.245	yes
Type-0 label bits	0.109	0.053	yes
Fixed header	0.0003	0.0003	yes
Net rate	1.284	1.484	—

1.6 Environment and reproducibility

All experiments were run on a single desktop with an Intel Core i5-12400 CPU (6 cores), 32 GB of RAM, and Windows 11, using Python 3.14 and NumPy 2.4. The pipeline is single-threaded. Timing was measured as the mean of three runs per image, with the timer wrapped around the method pipeline only (embedding, recovery, and side-information packing) and stopped before any quality metric is computed; the reported running times are wall-clock seconds under this convention. The permutation and modulation keys and every random seed are fixed constants in the code, so a rerun reproduces the tables exactly; the 100-image selection of each general dataset follows the sampling protocol of Section 1.1, and the selected file identifiers are listed in the accompanying CSV files.

Key derivation and per-block parameters.

In the notation of Section 3.1, the slope and offset keys K_s, K_c and the per-block triples (a_B, b_B, c_B) are derived from K_m through a key-derivation function and a pseudorandom function. Our reference implementation realizes these with a keyed pseudorandom number generator (the NumPy PCG64 generator, seeded from the key and the per-image nonce) as a stand-in, which is sufficient to measure capacity, reversibility, and the statistical leakage features reported here. A deployment concerned with cryptographic strength would instead instantiate the key-derivation function and the pseudorandom function with standard primitives, for example HKDF-SHA-256 for $K_s = \text{KDF}(K_m, \text{slope})$ and $K_c = \text{KDF}(K_m, \text{offset})$ and an HMAC- or AES-based pseudorandom function for the per-block draws; this substitution does not change any rate or reversibility result, only the assumed hardness behind the PRF assumption used in the security discussion.

1.7 Baseline and comparison implementations

The baseline and the two comparison schemes are our own implementations, since the original code is not public. We validated our baseline re-implementation on the standard images: it is lossless on every image, and its net embedding rate reproduces

that of the original scheme up to a small offset, which we attribute to differences in the side-information packing rather than in the predictor or the coding. The proposed method reuses the HBP-RMC coding and the side-information packing of the baseline, while replacing the encryption and the prediction modules and adding adaptive reference selection, so the comparison isolates those changes rather than incidental coding differences.

2 Leakage measurement protocol

This section gives the exact protocol behind the leakage AUC of Section 4.1 of the main paper (Table 1 there). The observer knows the block grid but no key, and tries to tell a smooth original block from a textured one from the encrypted image.

Features.

For each $k_1 \times k_1$ block B of the observed image Z_B we compute two scalar features. The first is the block variance. The second is the mean modular second-difference energy

$$S_B = \frac{1}{|\Omega_B|} \sum_{(i,j) \in \Omega_B} (|D_{xx}Z_B(i,j)| + |D_{yy}Z_B(i,j)| + |D_{xy}Z_B(i,j)|), \quad (\text{S5})$$

where D_{xx}, D_{yy}, D_{xy} are the vertical, horizontal, and mixed second-order differences, Ω_B is the set of interior positions at which all three are defined, and each modular difference is mapped to the signed range $[-128, 127]$ *before* the absolute value is taken. This mapping is essential: without it the wrap-around of the modular arithmetic would hide the small residuals.

Labels.

The label is derived from the *original* block, not the ciphertext: a block is “textured” if the variance of its original pixels is above the per-image median, and “smooth” otherwise. The median split gives two classes of equal size and fixes the threshold a priori, so no parameter is tuned on the ciphertext.

AUC.

For each feature the classifier is a single threshold on that feature; the AUC is the area under its ROC curve, computed per image over all its blocks and then averaged over the images of the set. A threshold classifier has no trained parameter, so there is no ciphertext-side fitting. To make the score independent of the sign of the classifier we report the symmetric value $\text{AUC}^* = \max(\text{AUC}, 1 - \text{AUC})$, so that a classifier and its inversion count as equally strong; an AUC^* of one is perfect separation and one half is no usable separation. The per-image spread is small: on the encrypted image the variance AUC^* of the proposed method is 0.51 ± 0.01 over the six standard images (mean $\pm$ standard deviation), against 1.00 ± 0.00 for the baseline, and the second-difference AUC^* is 0.95 ± 0.03 for both, so the equality of the two schemes under that feature is not an averaging artifact but holds image by image.

Table S2 χ^2 of the pixel-value histogram (lower is better; the 5% critical value for 255 degrees of freedom is about 293).

Image	Original	Baseline	Proposed
Forest	970403	46809	240
Baboon	187357	49613	255
Boat	274911	19462	261
Cameraman	418530	6784	245
Barbara	147984	23067	268
Airplane	728692	13458	280

Table S3 Global entropy and standard deviation of the local entropy over 16×16 blocks of the encrypted image. A smaller local standard deviation means a more uniform ciphertext at the block level.

Image	Baseline		Proposed	
	Global	Local	Global	Local
Forest	7.862	0.139	7.9993	0.052
Baboon	7.835	0.106	7.9993	0.051
Boat	7.934	0.136	7.9993	0.053
Cameraman	7.977	0.186	7.9993	0.056
Barbara	7.925	0.160	7.9993	0.055
Airplane	7.956	0.167	7.9992	0.054

Marked image.

We measure leakage on the encrypted image, which is the ciphertext that leaves the content owner. The marked image, obtained after the data hider writes a payload into the freed high bit-planes, is a different object; a rigorous leakage study of the marked image requires the same features to be measured on the fully embedded output of both schemes, with matched payload length and the auxiliary data included, for several payload patterns. We leave this controlled comparison as future work and do not report marked-image leakage figures here, to avoid drawing conclusions from a simplified embedding model.

Per-image histogram and entropy.

Tables S2 and S3 give the per-image numbers behind the histogram and entropy discussion of Section 4.1 of the main paper, over the six standard images used there, namely Forest, Baboon, Boat, Cameraman, Barbara, and Airplane.

3 Key sensitivity, key space, and known-plaintext analysis

This section gives the detailed analysis summarized in Section 4.1 of the main paper.

Key sensitivity.

A good encryption should be highly sensitive to the key. Averaged over the six standard images, flipping one bit of the modulation key K_m gives an NPCR of 99.607% and a UACI of about 33.5% between the two encrypted images, close to the random-image reference values of 99.6% and 33.4%.

Key space.

The key space is set by the seed lengths, not by the number of blocks: the permutation key K_p and the modulation key K_m are independent 128-bit seeds. The joint key tuple therefore contains 256 bits, while the security of each component is bounded by its 128-bit seed length. With a cryptographic KDF/PRF instantiation the construction therefore targets 128-bit component-wise brute-force resistance rather than a generic 256-bit level; the experimental PCG64 prototype is not used to support this cryptographic-strength claim. We do not count the 16,384! block orderings as key material, because a fixed-length seed cannot index all of them; the permutation is only as strong as the seed that generates it.

Known-plaintext analysis.

The keystream of a single block is a tilted plane, so an attacker who knows three suitably positioned plaintext–ciphertext pairs of a block obtains three linear congruences modulo 256 in its parameters (a_B, b_B, c_B) . These determine the parameters uniquely when the associated coordinate matrix is invertible over $\mathbb{Z}_{256}$, equivalently when its determinant is odd; the three positions $(0, 0)$, $(1, 0)$, $(0, 1)$, for instance, give a matrix of determinant 1. Once the parameters are known, the attacker can decrypt the remaining pixels of that same block, so this is a block-local known-plaintext weakness. A single global triple shared by all blocks would be far worse, because the recovered parameters would then decrypt every block. We instead draw the triple of each block independently through the per-block pseudorandom function $\text{PRF}_{K_s}, \text{PRF}_{K_c}$ of Section 3.1 of the main paper; under the standard assumption that this function is pseudorandom, the triples of the remaining blocks are computationally indistinguishable from independent random values, so under the PRF assumption knowledge of one block gives no feasible advantage in predicting another. Finally, changing one pixel of the original image changes only one pixel of the encrypted image, for the baseline as well as for the proposed method; both constructions deliberately avoid global diffusion because it would destroy the intra-block redundancy that the embedding needs. The metrics reported in Section 4.1 characterize selected statistical leakage channels under the stated threat model; they do not constitute a formal semantic-security proof.

4 Per-image results

Section 4.2 of the main paper reports averages. The tables below give the individual images behind those averages, which is what supports the claim that the proposed method is above the baseline on *every* image and not only on average.

On the standard set, listed in Table S4, the ordering of the four methods is the same on all 12 images; Figure S2 shows the same data as a bar chart. Baboon, the

Table S4 Per-image net embedding rate (bits per pixel) on the 12 standard images. The last row is the average, which is the “Standard” row of Table 2 of the main paper. All rates are net and lossless.

Image	Gao	Venk.	Chai	Prop.
Airplane	0.528	1.476	1.638	1.873
Baboon	0.308	0.380	0.480	0.592
Barbara	0.410	1.072	1.307	1.493
Boat	0.450	0.978	1.221	1.385
Cameraman	0.563	1.659	1.938	2.185
Car	0.486	1.158	1.386	1.581
Forest	0.432	0.685	0.932	1.133
Goldhill	0.478	0.965	1.178	1.355
Pepper	0.496	1.151	1.322	1.577
Sailboat	0.449	0.881	1.072	1.269
Things	0.483	1.269	1.475	1.710
Tiffany	0.482	1.226	1.457	1.651
Average	0.464	1.075	1.284	1.484

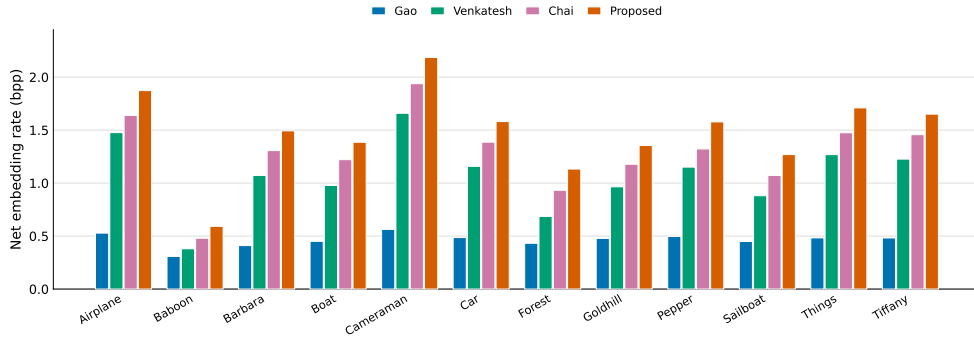


Fig. S2 Net embedding rate on the 12 standard images, for the two recent VRAE schemes, the baseline of Chai et al., and the proposed method. Each group of four bars is one image, in the order of Figure S1. The proposed method is the tallest bar in every group; the same data is listed in Table S4.

most textured image of the set, is the hardest case for every method, and it is where the relative gain of the proposed method over the baseline is largest (+23%). This is the trend discussed in Section 4.3 of the main paper: the improvement is largest exactly where the baseline is weakest.

The same holds on the three general datasets: on each of the 100-image subsets of BOSSbase, BOWS2, and UCID the proposed method has the highest net rate on every evaluated image, so the dataset averages of Table 2 of the main paper are not produced by a favourable subset.

Algorithm 1 Encoder (proposed method)

Require: image X ; keys K_p, K_m, K_h ; public per-image nonce

- 1: Derive $K_s = \text{KDF}(K_m, \text{slope})$ and $K_c = \text{KDF}(K_m, \text{offset})$
 - 2: Permute the blocks with K_p
 - 3: **for** each permuted block B **do**
 - 4: $(a_B, b_B) \leftarrow \text{PRF}_{K_s}(\text{nonce} \parallel B), c_B \leftarrow \text{PRF}_{K_c}(\text{nonce} \parallel B)$
 - 5: redraw a_B, b_B until $a_B, b_B \in \{1, \dots, 255\}$ (rejection sampling)
 - 6: encrypt: $Z_B(i, j) \leftarrow (X_B(i, j) + a_B i + b_B j + c_B) \bmod 256$
 - 7: **end for**
 - The data hider receives Z and the slope key K_s only:*
 - 8: **for** each block B **do**
 - 9: **for** each reference position (r_h, r_p) **do**
 - 10: predict each pixel from one reference-side neighbor using Δr from (a_B, b_B) ;
 residual $e \leftarrow \text{smod}_{256}(\cdot)$
 - 11: build a candidate exception package EM_r of the pixels with $e = -128$
 (their positions and directly stored values)
 - 12: $R_r \leftarrow \text{HBP-RMC room}$
 - 13: **end for**
 - 14: keep the reference r^* with the largest room R_r ; retain only its residuals and
 EM_{r^*} ; record its 4-bit index
 - 15: **end for**
 - 16: compress EM and the reference indices (arithmetic coding)
 - 17: free room by HBP-RMC in the wavefront order of increasing Manhattan distance
 from the reference; embed the auxiliary data and the message encrypted with K_h
-

5 Encoder pseudocode

Algorithm 1 gives the pseudocode of the encoder that Section 3.5 of the main paper describes in words. The decoder reverses the steps: it reads the side information, restores every encrypted block in the wavefront order of increasing Manhattan distance from the reference, and then decrypts with K_p and K_m . The scheme is separable, so a receiver holding only K_h can extract the message without being able to decrypt the image.

The per-block selection treats the four-bit reference-index cost as fixed: arithmetic coding of the index stream is applied only after all references are chosen, and its small average gain (about 0.005 bpp, Table S1) is not part of the local objective. The exception cost $|EM_r|$ is likewise not part of the objective; because $e(p) = -128$ occurs for only about 0.004% of the pixels, it is nearly constant across the candidate references, so selecting by HBP-RMC room is within a negligible margin of selecting by net room. The reference index itself costs $\log_2(k_1^2) = 4$ bits per block, already subtracted from every reported net rate.