

Hybrid Approach

```
%% hybrid_improved_fixed.m
% Improved Hybrid Chaotic Encryption (Hybrid Lorenz/Rossler + Henon + Tent)
% - per-channel keystreams
% - SHA-256 whitening of chaotic blocks
% - feedback diffusion (CBC-like)
% - per-channel permutation
% - two rounds of confusion+diffusion
%
% Save as hybrid_improved_fixed.m and run.
clear; close all; clc;

%% ----- User parameters (keys) -----
% Lorenz (used as extra entropy source)
lorenz_sigma = 10; lorenz_beta = 8/3; lorenz_rho = 28;
lorenz_x0 = 0.1; lorenz_y0 = 0.0; lorenz_z0 = 0.1;

% Rossler
rossler_a = 0.2; rossler_b = 0.2; rossler_c = 5.7;
rossler_x0 = 0.1; rossler_y0 = 0.1; rossler_z0 = 0.1;

% Henon
henon_a = 1.4; henon_b = 0.3;
henon_x0 = 0.1; henon_y0 = 0.1;

% Tent
tent_x0 = 0.12345; tent_mu = 1.9999;

% integration settings
dt = 0.01;
t_end = 200;

% mix factor control (0..1)
mix_factor = 0.5;

% hashing block size (number of doubles per hash)
HASH_BLOCK_D = 32; % adjust for throughput/entropy tradeoff

% rounds
NUM_ROUNDS = 2;

%% ----- Load image -----
img = imread(' C:\Users\Control Lab\Downloads\codes\codes\state of art methods\paper.PNG');
if size(img,3) == 1
    img = repmat(img,[1 1 3]);
end
img = uint8(img);
[H, W, Cc] = size(img);
Nchan = H * W; % pixels per channel
Npix_total = Nchan * Cc; % total bytes

%% ----- Generate chaotic sequences -----
```

```

% time vector for ODE integration (Lorenz and Rossler)
t = 0:dt:t_end;

% Lorenz via ode45
lorenz_ic = [lorenz_x0; lorenz_y0; lorenz_z0];
lorenz_f = @(tt, s) [lorenz_sigma*(s(2)-s(1)); s(1)*(lorenz_rho-s(3)) - s(2); s(1)*s(2) - lorenz_beta*s(1)];
[~, S_l] = ode45(lorenz_f, t, lorenz_ic);

% Rossler via ode45
rossler_ic = [rossler_x0; rossler_y0; rossler_z0];
rossler_f = @(tt, s) [-s(2)-s(3); s(1)+rossler_a*s(2); rossler_b + s(3).*(s(1)-rossler_c)];
[~, S_r] = ode45(rossler_f, t, rossler_ic);

% Ensure length sufficient for per-channel operations
len_needed = Nchan + 2000;
if size(S_l,1) < len_needed
    rep_factor = ceil(len_needed / size(S_l,1)) + 1;
    S_l = repmat(S_l, rep_factor, 1);
    S_r = repmat(S_r, rep_factor, 1);
end
S_l = S_l(1:len_needed, :);
S_r = S_r(1:len_needed, :);

% Henon map (discrete)
henon_seq = zeros(len_needed,2);
xh = henon_x0; yh = henon_y0;
for i = 1:len_needed
    x_next = 1 - henon_a*xh^2 + yh;
    y_next = henon_b * xh;
    henon_seq(i,:) = [x_next, y_next];
    xh = x_next; yh = y_next;
end

% Tent map
tent_seq = zeros(len_needed,1);
xt = tent_x0;
for i = 1:len_needed
    if xt < 0.5
        xt = tent_mu * xt;
    else
        xt = tent_mu * (1 - xt);
    end
    tent_seq(i) = xt;
end

%% ----- Build per-channel mixes (normalized) -----
normalize = @(v) (v - min(v)) ./ (max(v) - min(v) + eps);

a1 = normalize(S_l(:,1)); % lorenz x
a2 = normalize(S_r(:,1)); % rossler x
a3 = normalize(henon_seq(:,1));
a4 = normalize(tent_seq(:));

% Create three different mixes for R,G,B (length len_needed)

```

```

mixR = normalize( 0.6*a1 + 0.4*a3 );      % emphasize continuous + henon
mixG = normalize( 0.5*a2 + 0.5*a4 );      % rossler + tent
mixB = normalize( 0.5*(a1+a2) + 0.0*a3 ); % lorenz+rossler

%% ----- Whiten (SHA-256) to produce byte keystreams -----
ksR = generate_whitened_keystream(mixR, Nchan, HASH_BLOCK_D);
ksG = generate_whitened_keystream(mixG, Nchan, HASH_BLOCK_D);
ksB = generate_whitened_keystream(mixB, Nchan, HASH_BLOCK_D);

%% ----- Create per-channel permutations -----
% Build per-channel permutation seeds from different combinations (length Nchan)
tempR = uint64(floor(normalize(a1(1:Nchan)) * 1e12));
tempG = uint64(floor(normalize(a2(1:Nchan)) * 1e12));
tempB = uint64(floor(normalize(a3(1:Nchan)) * 1e12));

permSeedR = double(mod(tempR + bitshift(tempG,5), 2^52)); % safe double range
permSeedG = double(mod(tempG + bitshift(tempB,7), 2^52));
permSeedB = double(mod(tempB + bitshift(tempR,11),2^52));

[~, idxR] = sort(permSeedR);
[~, idxG] = sort(permSeedG);
[~, idxB] = sort(permSeedB);

permR = idxR;
permG = idxG;
permB = idxB;

%% ----- Encryption: two rounds of (permute per channel + feedback diffusion) -----
% Flatten per-channel vectors (length Nchan)
Rvec = img(:,:,1); Rvec = Rvec(:);
Gvec = img(:,:,2); Gvec = Gvec(:);
Bvec = img(:,:,3); Bvec = Bvec(:);

% Two rounds
P_R = Rvec; P_G = Gvec; P_B = Bvec;
for rnd = 1:NUM_ROUNDS
    % Confusion (per-channel permutation)
    P_R = P_R(permR);
    P_G = P_G(permG);
    P_B = P_B(permB);

    % Diffusion (per-channel feedback with corresponding keystream)
    offset = (rnd-1) * 13; % small rotation
    KR = circshift(ksR, -offset);
    KG = circshift(ksG, -offset);
    KB = circshift(ksB, -offset);

    C_R = feedback_encrypt_channel(P_R, KR);
    C_G = feedback_encrypt_channel(P_G, KG);
    C_B = feedback_encrypt_channel(P_B, KB);

    % Prepare for next round (cipher becomes plaintext for next round)
    P_R = C_R; P_G = C_G; P_B = C_B;
end

```

```

% Assign encrypted channels back to image
cipher_img = zeros(H, W, Cc, 'uint8');
cipher_img(:,:,1) = reshape(C_R, H, W);
cipher_img(:,:,2) = reshape(C_G, H, W);
cipher_img(:,:,3) = reshape(C_B, H, W);

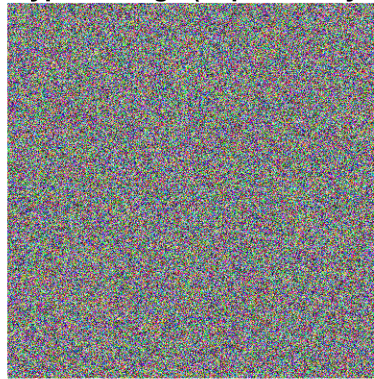
%% ----- Display Encryption Results -----
figure('Name','Hybrid Improved Encryption','NumberTitle','off');
subplot(1,2,1); imshow(img); title('Original Image');
subplot(1,2,2); imshow(cipher_img); title('Encrypted Image (Improved Hybrid)');

```

Original Image



Encrypted Image (Improved Hybrid)



```

%% ----- Timing (single-run core) -----
% time the core confusion+diffusion (per-channel operations)
Rvec_t = img(:,:,1); Rvec_t = Rvec_t(:);
Gvec_t = img(:,:,2); Gvec_t = Gvec_t(:);
Bvec_t = img(:,:,3); Bvec_t = Bvec_t(:);

tstart = tic;
P_R = Rvec_t; P_G = Gvec_t; P_B = Bvec_t;
for rnd = 1:NUM_ROUNDS
    P_R = P_R(permR);
    P_G = P_G(permG);
    P_B = P_B(permB);
    KR = circshift(ksR, -(rnd-1)*13);
    KG = circshift(ksG, -(rnd-1)*13);
    KB = circshift(ksB, -(rnd-1)*13);
    C_R = feedback_encrypt_channel(P_R, KR);

```

```

    C_G = feedback_encrypt_channel(P_G, KG);
    C_B = feedback_encrypt_channel(P_B, KB);
    P_R = C_R; P_G = C_G; P_B = C_B;
end
t_enc = toc(tstart);
fprintf('Improved Hybrid single-run encryption time (core ops): %.6f s\n', t_enc);

```

Improved Hybrid single-run encryption time (core ops): 0.019040 s

```

%% ----- Decryption (inverse operations) -----
D_R = C_R; D_G = C_G; D_B = C_B;
for rnd = NUM_ROUNDS:-1:1
    KR = circshift(ksR, -(rnd-1)*13);
    KG = circshift(ksG, -(rnd-1)*13);
    KB = circshift(ksB, -(rnd-1)*13);

    P_R = feedback_decrypt_channel(D_R, KR);
    P_G = feedback_decrypt_channel(D_G, KG);
    P_B = feedback_decrypt_channel(D_B, KB);

    % inverse permutation
    inv_permR = zeros(size(permR)); inv_permR(permR) = 1:length(permR);
    inv_permG = zeros(size(permG)); inv_permG(permG) = 1:length(permG);
    inv_permB = zeros(size(permB)); inv_permB(permB) = 1:length(permB);

    P_R = P_R(inv_permR);
    P_G = P_G(inv_permG);
    P_B = P_B(inv_permB);

    D_R = P_R; D_G = P_G; D_B = P_B;
end

% Recombine channels into recovered image
recovered_img = zeros(H,W,Cc,'uint8');
recovered_img(:,:,1) = reshape(D_R, H, W);
recovered_img(:,:,2) = reshape(D_G, H, W);
recovered_img(:,:,3) = reshape(D_B, H, W);

if isequal(img, recovered_img)
    disp('✅ Decryption successful: recovered original image exactly.');
```

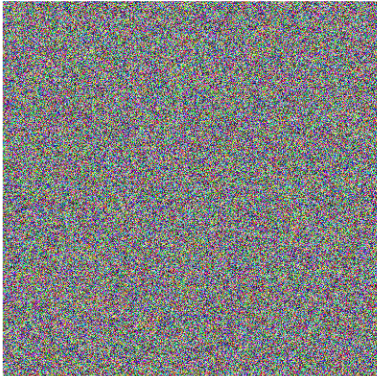
✅ Decryption successful: recovered original image exactly.

```

figure('Name','Decryption Result','NumberTitle','off');
subplot(1,2,1); imshow(cipher_img); title('Encrypted (Cipher) Image');
subplot(1,2,2); imshow(recovered_img); title('Recovered Image');

```

Encrypted (Cipher) Image



Recovered Image

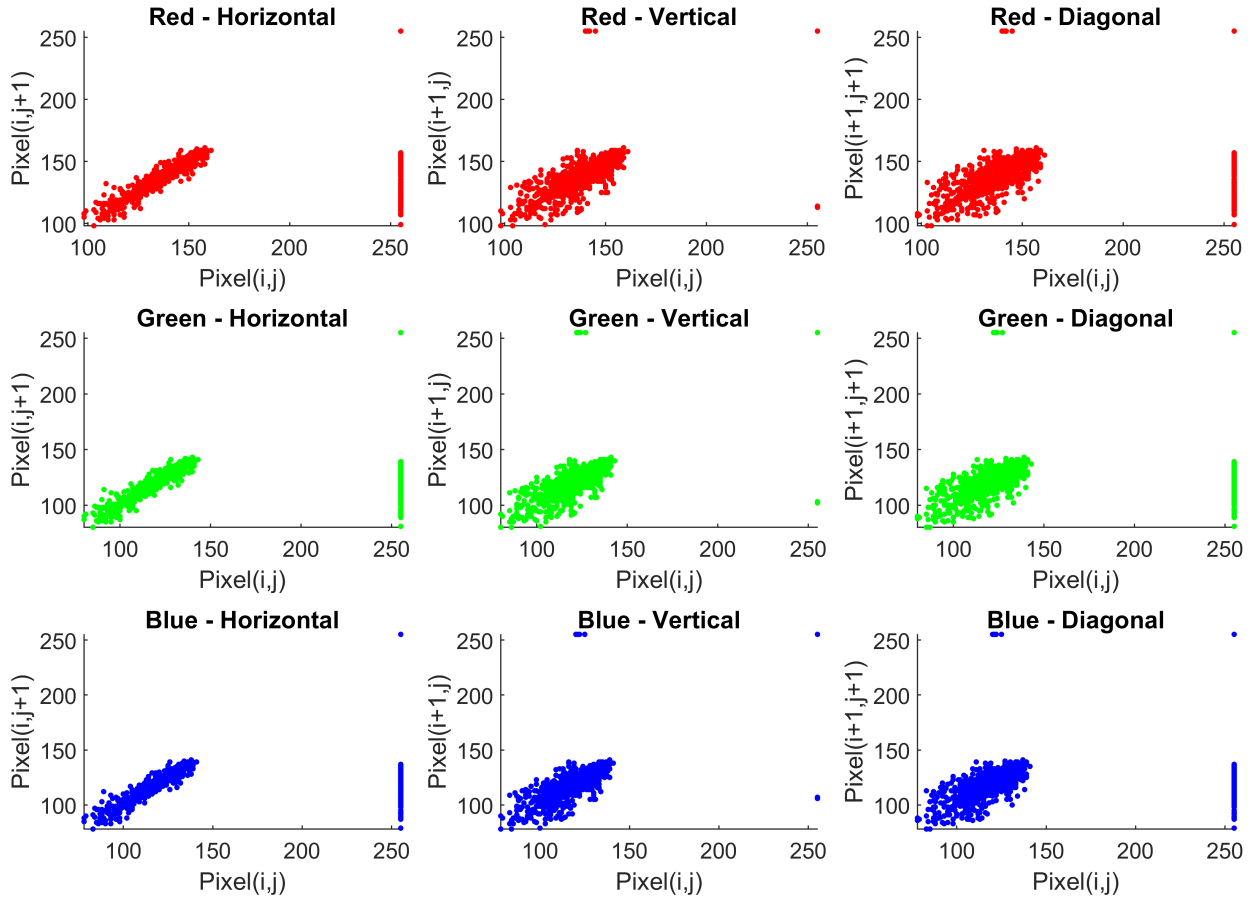


```
%% ----- Evaluation: Chi-square + Correlation -----  
chi_H = chi_square_hist(cipher_img, 'Improved Hybrid (cipher)');
```

```
Chi-Square Values (Improved Hybrid (cipher)):  
Red Channel   : 244.8823  
Green Channel : 257.2569  
Blue Channel  : 298.8014
```

```
correlation_analysis(img, 'Original Image Correlation');
```

Original Image Correlation



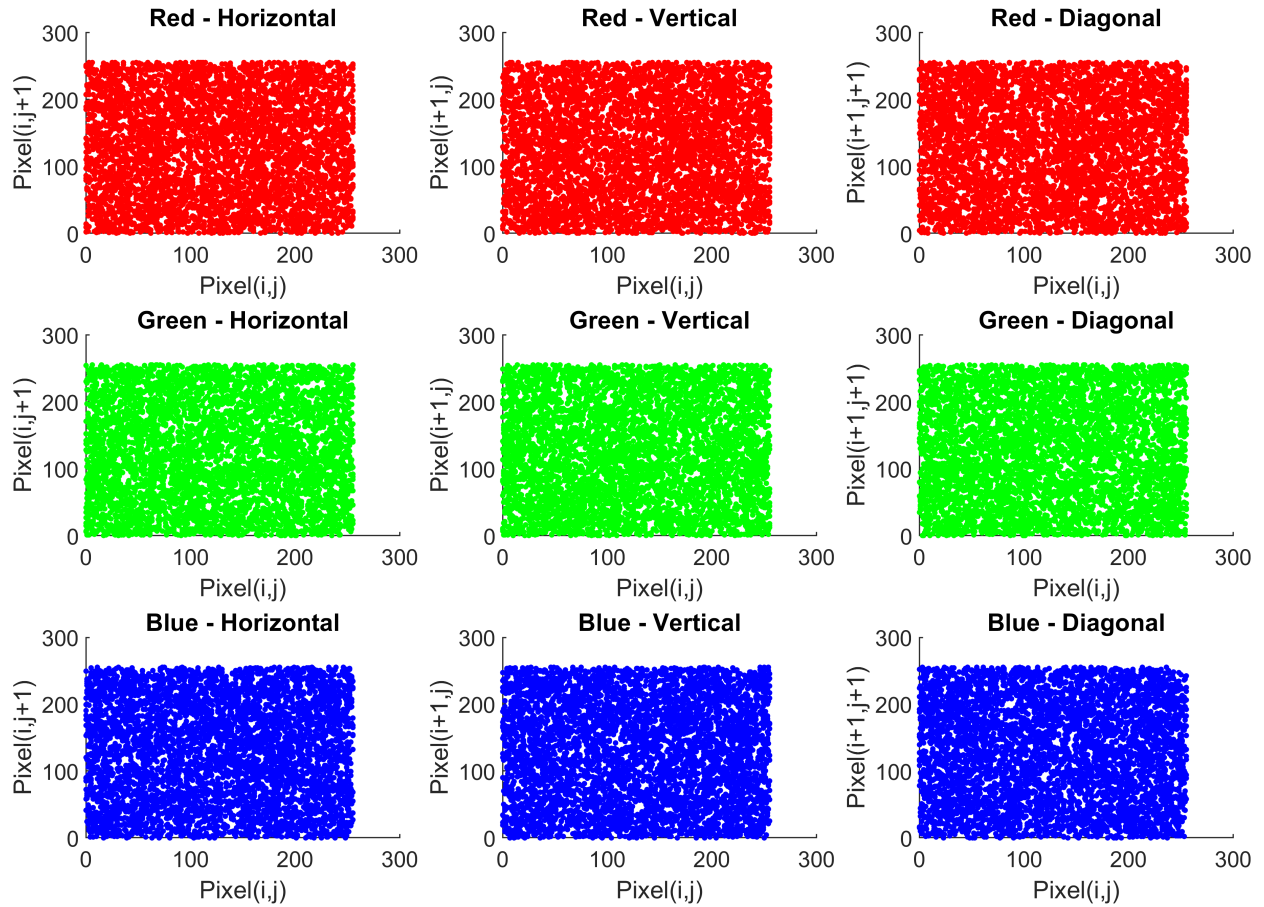
```

===== Correlation Summary: Original Image Correlation =====
          Horizontal      Vertical      Diagonal
-----
Red      :      0.9663      0.9568      0.9191
Green    :      0.9670      0.9587      0.9221
Blue     :      0.9688      0.9608      0.9263
=====

```

```
correlation_analysis(cipher_img, 'Improved Hybrid Encrypted Image Correlation');
```

Improved Hybrid Encrypted Image Correlation



```

===== Correlation Summary: Improved Hybrid Encrypted Image Correlation =====
      Horizontal      Vertical      Diagonal
-----
Red    :      0.0091      -0.0015      0.0016
Green  :     -0.0042       0.0021      0.0016
Blue   :      0.0062     -0.0013     -0.0020
=====
  
```