```
% This matlab file gives some of the basic Optical Image Encryption schemes
% Double random phase encoding, Phase truncation and reservation, Chosen plaintext attack,
% Some docompositions, i.e., EMD, RMD and PD are given
% One may use these only for research purpose and is expected to cite the relevant articles
%%
clc; clear all; close all;

A = double(imread('lena.jpg')); % Read grayscale image (use as input image)
figure(1); imagesc(A); colormap('gray'); axis square; title('Input Image'); % To view input image
R1=rand(256,256);    % Random matrix
RPM1=exp(1i*2*pi*R1); % First random phase mask
R2=rand(256,256); % Random matrix
RPM2=exp(1i*2*pi*R2); % Second random phase mask

%% DRPE

% Encryption process
E1=A.*RPM1; % Input image bonded with first random    phase mask
E2=fft2(E1); % Propgated through Fourier transfrom
E3=E2.*RPM2; % Bonded with second phase mask
E4=fft2(E3); % Propgated through Fourier transfrom
figure(2); imagesc(abs(E4)); colormap('gray'); axis square; title('Encrypted Image'); % To view ciphertext image

% Decryption process
D1=ifft2(E4).*conj(RPM2); % ciphertext bonded with conjugate of second phase mask
D=ifft2(D1);                 % Decrypted image
figure(3); imagesc(abs(D)); colormap('gray'); axis square; title('Decrypted Image') % To view decrypted image

%% Chosen plaintext attack (CPA) on DRPE
AA=zeros(256,256); % Zeroes matrix
AA(128,130)=1;          % Dirac delta function use as plaintext
e1=fft2(AA.*RPM1);
e2=fft2(e1.*RPM2); % Ciphertext obtaned by chosen plaintext
h=fftshift(fft2(fft2(E4)./fft2(e2))); % Attack process
figure(4); imagesc(abs(h)); colormap('gray'); axis square; title('Retrived Image CPA'); % Retrived image by CPA

%% Phase truncation and phase reservation in Fourier transfrom (PTFT)
% Encryption process

E=A.*RPM1; % Input image bonded with random phase amsk
```

```
E1=fft(E); % Propgated through Fourier transfrom
P1=abs(E1); % Phase truncation
P2=angle(E1); % Phase reservation
P2=exp(1i*P2); % Private key
E2=ifft(P1.*RPM2); % Propgated through inverse Fourier transfrom act as ciphertext
figure(5);imagesc(abs(E2));colormap('gray');axis square;title('Encrypted Image');
P3=abs(E2);    % Phase truncation
P4=angle(E2); % Phase reservation
P4=exp(1i*P4); % Private key

% Decryption process
D1=fft(P3.*P4); % Second private key bonded ciphertext
D2=ifft(abs(D1).*P2); % D1 bonded with first private key
figure(6);imagesc(abs(D2));colormap('gray'); axis square;title('Decrypted Image');
%% Equal modulus decomposition (EMD)
% Encryption
thetae=2*pi*rand(256,256);    % Angle of rotation
B=fft2(A.*RPM1); % Propgated through Fourier transform
phie=angle(B); Au=abs(B);
P1=((Au/2)./cos(phie-thetae)).*exp(1i.*thetae); % First part of equal modulus decomposition
P2=((Au/2)./cos(phie-thetae)).*exp(1i*(2*phie-thetae)); % Second part of equal modulus decomposition
C=abs(P1); % Ciphertext
pr=angle(P2); % Private key

% Decryption
D=P1+P2; % Addition of ciphertext and private key
FG=ifft2(D);%.*conj(RPM2); % propgated through inverse Fourier transfrom
figure(7), imagesc(abs(FG)); colormap(gray); title('Decrypted image'); axis square

%% Random modulus decomposition (RMD)

% Encryption
a1=(2.*pi*R2); b1=(2.*pi*R2);      % Angle of rotation
I1=fft2(A.*RPM1);    % Propgation through Fourier Transform
A1=abs(I1);    B1=angle(I1);
P11=((A1.*sin(b1))./sin(a1+b1)).*exp(1i.*(B1-a1)); % First part of equal modulus decomposition
P22=((A1.*sin(a1))./sin(a1+b1)).*exp(1i.*(B1+b1)); % Second part of equal modulus decomposition

% Decryption
D1=P11+P22;    % Addition of ciphertext and private key
D=ifft2(D1); % Propgation through inverse Fourier transform
figure(8), imagesc(abs(D)); colormap(gray); title('Decrypted image'); axis square
```

```
%% Polar Decomposition (PD)
[R, U, V]= poldecomp(A); % To decompose signal A
D=R*U; % D=A To retrive signal
D1=V*R; % D1=A    To retrive signal
figure(9); imagesc(abs(D)); colormap('gray'); axis square; title('Decrypted image');

%% Function of Ploar decomposition
function [R U V] = poldecomp(F)
%POLDECOMP    Performs the polar decomposition of a regular square matrix.
%      [R U V] = POLDECOMP(F) factorizes a non-singular square matrix F such
%      that F=R*U and F=V*R, where
%      U and V are symmetric, positive definite matrices and
%      R is a rotational matrix
%      See also EIG, DIAG, REPMAT
% This kind of decomposition is often used in continuum mechanics so it is
% convenient to comment the code that way. From now, we use the matrix
% formalism of tensors. C is the right Cauchy-Green deformation tensor,
% F is the deformation tensor, lambda is the stretch.
% Check input
[m n] = size(F);
if m ~= n
    error('Matrix must be square.');
end
C = F'*F;
[Q0 lambdasquare] = eig(C);
lambda = sqrt(diag((lambdasquare))); % extract the components
% Uinv is the inverse of U and is constructed with the help of Q0. Uinv is
% produced in the same base as F not in the base of its eigenvectors.
Uinv = repmat(1./lambda',size(F,1),1).*Q0*Q0';
% Using the definition, R, U and V can now be calculated
R = F*Uinv;
U = R'*F;
V = F*R';
end
```