```
function Correlation = CorrelationOfAdjacentPixels(Image)

%Chooses 6000 random pairs of adjacent pixels and computes correlation
    of
%the pair

Width = size(Image,1);
Height = size(Image,2);
PixelCouplesSample = 6000;

%Generates first point coordinates
x1 = randi(Width,1,PixelCouplesSample);
y1 = randi(Height,1,PixelCouplesSample);
%Generates second point coordinates keeping track of periodic boundary
%conditions
x2 = mod(x1+1,Width+1)+(x1==Width);
y2 = mod(y1+1,Height+1)+(y1==Height);

%Collects pixel values
FirstPixel = zeros(1,PixelCouplesSample);
SecondPixel = zeros(1,PixelCouplesSample);
for i = 1:PixelCouplesSample
    FirstPixel(i) = Image(x1(i),y1(i));
    SecondPixel(i) = Image(x2(i),y2(i));
end

Correlation = corr2(FirstPixel,SecondPixel);

end
```

```
function EncryptedImage = Encrypter(Image,Sequence,Rule,Iterations)

%Encrypts given image by permutating its pixel values according to
%evolution of Game of Life cellular Automata generated from given
%password via logistic map

    %Generates board starting rounding pseudorandom sequence
    Board = reshape(round(Sequence),size(Image));

    %Records History
    AlreadyOn = zeros(size(Image));
    NewOrder = [];
    for t = 1:Iterations
            NewlyOn = xor(Board,AlreadyOn) & Board;
            NewOrder = cat(1,NewOrder,find(NewlyOn));
            AlreadyOn = AlreadyOn | Board;
            Board = Evolve(Board,Rule);
    end
    NewOrder = cat(1,NewOrder,find(~AlreadyOn));

    PartiallyEncrypted = reshape(Image(NewOrder),size(Image))';
    EncryptedImage = reshape(PartiallyEncrypted(NewOrder),size(Image))
        ;

end
```

```matlab
function NewBoard = Evolve(OldBoard,Rule)

%Evolves board of 2d cellular automata according to chosen rule,
    assuming
%periodic boundary conditions

switch Rule
    case 'Life'
        %Number of neighboring cells required for a dead cell to turn
            alive
        Birth = 3;
        %Number of neighboring cells required for an alive cell to
            survive
        Survival = [2 3];
    case 'Fredkin' %Chaotic rule, fill percentage converges to 50%
        Birth = 1:2:7;
        Survival = 0:2:8;
end

%Last column will be counted as adjacent to the first column, etc.
PeriodicBoard = ...
    [OldBoard(end,end), OldBoard(end,:),OldBoard(end,1)
    OldBoard(:,end),OldBoard,OldBoard(:,1)
    OldBoard(1,end), OldBoard(1,:), OldBoard(1,1)];

%Neighbors are positions reachable by a King
MooreNeighborhood = [1 1 1; 1 0 1; 1 1 1];
%Gives number of alive neighbors of each cell
Neighbors = conv2(PeriodicBoard, MooreNeighborhood, 'same');
%Modifies a cell state according to how many alive neighbors it has
NewBoard = ...
    ismember(Neighbors,Birth).*(1-PeriodicBoard)+...
    ismember(Neighbors,Survival).*PeriodicBoard;

%Removes the boundary copies
NewBoard = NewBoard(2:end-1,2:end-1);

end
```

```
function Sequence = LogisticRandomSequence(Length,Mu,X0)
%Generates random sequences of floats between 0 and 1 using logistic map
%with seed 3.9 < Mu < 4.0 and 0 < X0 < 1

Sequence = zeros(1,Length);
Sequence(1) = X0;
for i=1:Length-1
    Sequence(i+1) = Mu*Sequence(i)*(1-Sequence(i));
end

end
```

```
clc
close all
clear

RGBImage = imread('dogs.jpeg');
Image = rgb2gray(RGBImage);
Height = size(Image,1);
Width = size(Image,2);
Simulations = 1e3;
KeySensitivity = zeros(Simulations,2);
tic
for Iterations=[1000]
    filename = sprintf("KeySens%dSim%dIt.mat",Simulations,Iterations);
for i=1:Simulations
    %% Encryption
Password = [3.9+0.1*rand(),rand()];
Mu = Password(1); %Logistic Map parameter: 3.9 < Mu < 4.0
X0 = Password(2); %Logistic Map initial value: 0 < X0 < 1
Sequence = LogisticRandomSequence(Height*Width,Mu,X0);

LifeEncoded = Encrypter(Image,Sequence,'Life',Iterations);
FredkinEncoded = Encrypter(Image,Sequence,'Fredkin',Iterations);

%% Key sensitivity test

PerturbedSequence = Sequence;
ChangedIndex = randi(length(PerturbedSequence));
PerturbedSequence(ChangedIndex) = 1-PerturbedSequence(ChangedIndex);

PerturbedLifeEncoded = Encrypter(Image,PerturbedSequence,'Life',
   Iterations);
PerturbedFredkinEncoded = Encrypter(Image,PerturbedSequence,'Fredkin',
   Iterations);

KeySensitivity(i,:) = ...
    [corr2(LifeEncoded,PerturbedLifeEncoded),...
    corr2(FredkinEncoded,PerturbedFredkinEncoded)];
disp(100*i/Simulations)
end
save(strcat('data/',filename))
end
toc
```

```
clc
close all
clear

RGBImage = imread('dogs.jpeg');
Image = rgb2gray(RGBImage);
Height = size(Image,1);
Width = size(Image,2);
Simulations = 1e3;
KeySensitivity = zeros(Simulations,2);
tic
for Iterations=[1,10,100]
    filename = sprintf("OldKeySens%dSim%dIt.mat",Simulations,
       Iterations);
for i=1:Simulations
    %% Encryption
Password = [3.9+0.1*rand(),rand()];
Mu = Password(1); %Logistic Map parameter: 3.9 < Mu < 4.0
X0 = Password(2); %Logistic Map initial value: 0 < X0 < 1
Sequence = LogisticRandomSequence(Height*Width,Mu,X0);

LifeEncoded = Encrypter(Image,Sequence,'Life',Iterations);
FredkinEncoded = Encrypter(Image,Sequence,'Fredkin',Iterations);

%% Key sensitivity test
PerturbedSequence = LogisticRandomSequence(Height*Width,Mu+eps,X0+eps)
   ;

PerturbedLifeEncoded = Encrypter(Image,PerturbedSequence,'Life',
   Iterations);
PerturbedFredkinEncoded = Encrypter(Image,PerturbedSequence,'Fredkin',
   Iterations);

KeySensitivity(i,:) = ...
    [corr2(LifeEncoded,PerturbedLifeEncoded),...
    corr2(FredkinEncoded,PerturbedFredkinEncoded)];
disp(100*i/Simulations)
end
save(strcat('data/',filename))
end
toc
```

```
%% Key sensitivity test

PerturbedSequence = Sequence;
ChangedIndex = randi(length(PerturbedSequence));
PerturbedSequence(ChangedIndex) = 1-PerturbedSequence(ChangedIndex);

PerturbedLifeEncoded = Encrypter(Image,PerturbedSequence,'Life',
    Iterations);
PerturbedFredkinEncoded = Encrypter(Image,PerturbedSequence,'Fredkin',
    Iterations);

KeySensitivity(i,:) = ...
    [corr2(LifeEncoded,PerturbedLifeEncoded),
    corr2(FredkinEncoded,PerturbedFredkinEncoded)];
```