```
function Correlation = CorrelationOfAdjacentPixels(Image)
%Sceglie n coppie casuali di pixel adiacenti e ne calcola la
   correlazione
Width = size(Image,1);
Height = size(Image,2);
PixelCouplesSample = 6000;

x1 = randi(Width,1,PixelCouplesSample);
y1 = randi(Height,1,PixelCouplesSample);
x2 = mod(x1+1,Width+1)+(x1==Width);
y2 = mod(y1+1,Height+1)+(y1==Height);

xs = zeros(1,PixelCouplesSample);
ys = zeros(1,PixelCouplesSample);
for i = 1:PixelCouplesSample
    xs(i) = Image(x1(i),y1(i));
    ys(i) = Image(x2(i),y2(i));
end

Correlation = corr2(xs,ys);
end
```

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

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

Height = size(Image,1);
Width = size(Image,2);
Board = reshape(round(Sequence.^FillParameter),Height,Width);
%Generates board starting from pseudorandom sequence, with fill
    percentage
%depending on FillParameter

CAHistory = zeros(Height,Width,Iterations);

for t = 1:Iterations
    CAHistory(:,:,t) = Board;
    Board = Evolve(Board,Rule);
end
%Records History

RowPermuted = Permute(Image,CAHistory); %Permutes rows
EncryptedImage = Permute(RowPermuted',CAHistory)'; %Permutes columns

end
```

```matlab
function NewBoard = Evolve(OldBoard,Rule)
%Evolve la board secondo la regola fissata, con condizioni periodiche
   al
%contorno

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

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

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

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

```matlab
function Sequence = LogisticRandomSequence(Length,Mu,X0)
%Genera una sequenza di N float tra 0 e 1 tramite mappa logistica

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

```
function NewImage = Permute(OldImage,History)
%Permutes matrix based on recorded history of binary cellular automata
%First come the pixels cornensponding to alive cells at t=1
%Then for every t come pixel corresponding to cells initially dead but
%alive at time t
%At the end, all the rest

Order = []; %This will be the new order
Correction = ones(size(History(:,:,1))); %No corrections at t=1
for t=1:size(History,3)
    CurrentBoard = History(:,:,t).*Correction;
    %Removes cells alive at previous times
    Order = cat(1,Order,find(CurrentBoard == 1));
    %Keep track of indices of newly alive cells
    Correction(CurrentBoard==1) = 0;
    %Sets zeros in indices corresponding to cells already counted
end
Order = cat(1, Order, find(Correction==1));

NewImage = reshape(OldImage(Order),size(OldImage,1),size(OldImage,2));
%Permutes image using linear indices and then reshapes it to its
   correct
%dimensions
end
```

```matlab
clc
close all
clear

%% Encryption

RGBImage = imread('dogs.jpeg');
Image = rgb2gray(RGBImage);
Height = size(Image,1);
Width = size(Image,2);

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);
%Generates pseudorandom sequence with recursion rule
%X(n+1) = Mu*X(n)*(1-X(n))

Iterations = 20; %Iterations of cellular automata


[LifeEncoded,H1] = Encoder(Image,Sequence,'Life',Iterations,1);
[FredkinEncoded,H2] = Encoder(Image,Sequence,'Fredkin',Iterations,1);

subplot(1,3,1)
imshow(Image)
subplot(1,3,2)
imshow(LifeEncoded)
subplot(1,3,3)
imshow(FredkinEncoded)

%% Correlation tests

fprintf("Correlation between pairs of adjacent pixels:\n"),
fprintf("Original: %f \n Life: %f \n Fredkin: %f \n\n", ...
    CorrelationOfAdjacentPixels(Image),...
    CorrelationOfAdjacentPixels(LifeEncoded),...
    CorrelationOfAdjacentPixels(FredkinEncoded));

fprintf("Correlazione tra stessi pixel:\n")
fprintf(" Original-Life: %f\n Original-Fredkin: %f\n\n",...
    corr2(Image,LifeEncoded),...
    corr2(Image,FredkinEncoded));
```

```
%%Key sensitivity test

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

PerturbedLifeEncoded = Encoder(Image,PerturbedSequence,'Life',1,
    Iterations);
PerturbedFredkinEncoded = Encoder(Image,PerturbedSequence,'Fredkin',1,
    Iterations);

fprintf("Correlation of images encrypted with password differing in a
    part in 10^12:\n")
fprintf(" Life: %f\n Fredkin %f\n",...
    corr2(LifeEncoded,PerturbedLifeEncoded),...
    corr2(FredkinEncoded,PerturbedFredkinEncoded));
```

```
clc
close all
clear

RGBImage = imread('dogs.jpeg');
Image = rgb2gray(RGBImage);
Height = size(Image,1);
Width = size(Image,2);
Simulations = 1e3;
Iterations = 20; %Iterations of cellular automata
AdjacentPixelCorr = zeros(Simulations,3);
SamePixelCorr = zeros(Simulations,2);
KeySensitivity = zeros(Simulations,2);

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 = Encoder(Image,Sequence,'Life',1,Iterations);
FredkinEncoded = Encoder(Image,Sequence,'Fredkin',1,Iterations);

%% Correlation tests

AdjacentPixelCorr(i,:) = ...
    [CorrelationOfAdjacentPixels(Image),...
    CorrelationOfAdjacentPixels(LifeEncoded),...
    CorrelationOfAdjacentPixels(FredkinEncoded)];

SamePixelCorr(i,:) = ...
    [corr2(Image,LifeEncoded),...
    corr2(Image,FredkinEncoded)];

%% Key sensitivity test

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

PerturbedLifeEncoded = Encoder(Image,PerturbedSequence,'Life',
   Iterations,1);
```

```
PerturbedFredkinEncoded = Encoder(Image,PerturbedSequence,'Fredkin',
   Iterations,1);

KeySensitivity(i,:) = ...
    [corr2(LifeEncoded,PerturbedLifeEncoded),
    corr2(FredkinEncoded,PerturbedFredkinEncoded)];
disp(100*i/Simulations)

end

%% Istogrammi
figure(1)
hold on
histogram(AdjacentPixelCorr(:,1))
histogram(AdjacentPixelCorr(:,2))
histogram(AdjacentPixelCorr(:,3))
legend()
figure(2)
histogram(SamePixelCorr)
legend()
figure(3)
histogram(KeySensitivity)
legend()
```