Université de Genève

Multimedia Security and Privacy
14x016

TP 2 : Basic Cryptography and Watermarking

Author: Deniz Sungurtekin

E-mail: Deniz.Sungurtekin@etu.unige.ch

Mars 2020

1 Introduction

The main goal of this work is to see some fundamental differences between classical cryptography techniques and basic watermarking. First, we will see how we can easily make an image unrecognizable but very sensible to little perturbations. Then we will observe and analyse a classical cryptography algorithm like AES on images. Finally, we will do a simple watermarking technique based on the most significant bits (MSB) and the least significant bits (LSB) of a data type.

2 Encryption

2.1 Exercise 1

Here, we will read an image (liftingbody.png), permute it with a permutation matrix to see the effect on the histogram and the visual difference. This permutation matrix has the same dimension than the image where each element contain a random couple of indexes. Then for each element of our original image, we will simply replace it by the value at the specified position from the permutation matrix. Finally, you can find the "permutationMatrix.txt" file where our permutation file is stored. However, this method of permutation use the function shuffle in numpy which only shuffles along the firs axis. Moreover, it's not easy to define an inverse permutation that's why we will see another method of permutation in the next exercise.

We obtain the following result:

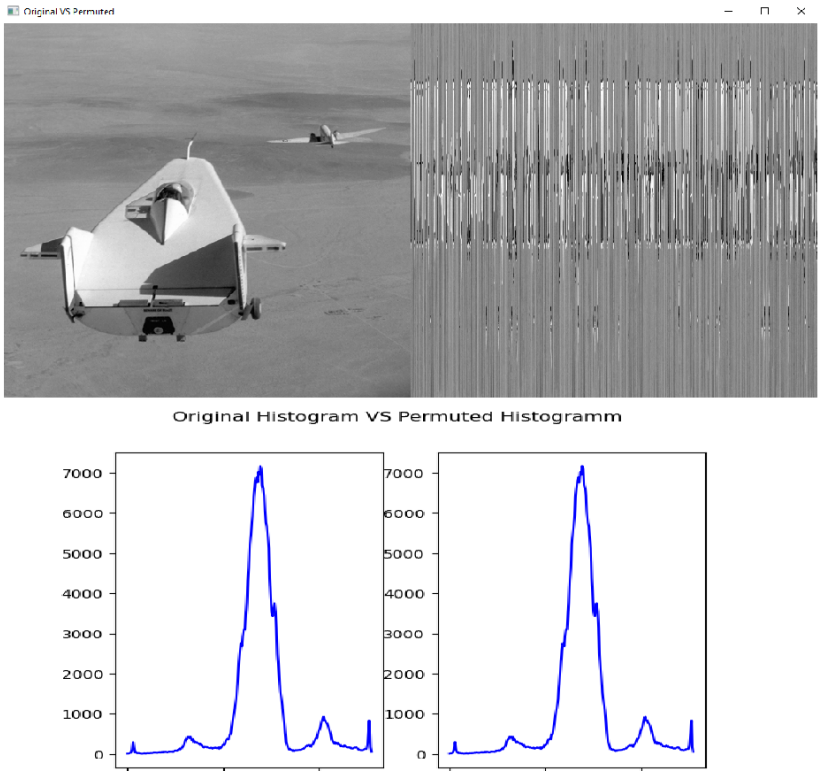

Figure 1: Comparaison between the two images

As expected, the permuted image is different and we don't really recognize the original image anymore. However, both histogram are identical and it is totally logical because the occurrence of pixels value doesn't change so we have exactly the same distribution, only the order of the pixels have changed.

2.2 Exercise 2

Now, we will define a block-loss function which depends on the shape of the image, we will set a random ($N_b x M_b$) block in the image at zero. If we have a NxM image, we will choose N_b and M_b randomly between 1 and N and 1 and M. Then, we simply choose the top left position of this block in the image depending of the block's size (not to exceed the image). After that, we can permute our image and distort it:

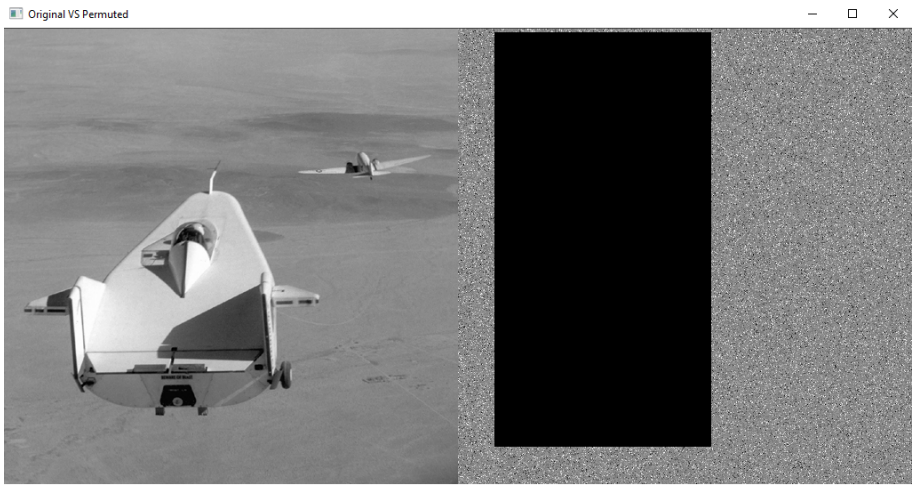

Figure 2: Original VS distorted + blockLoss

Then we will use the inverse permutation:

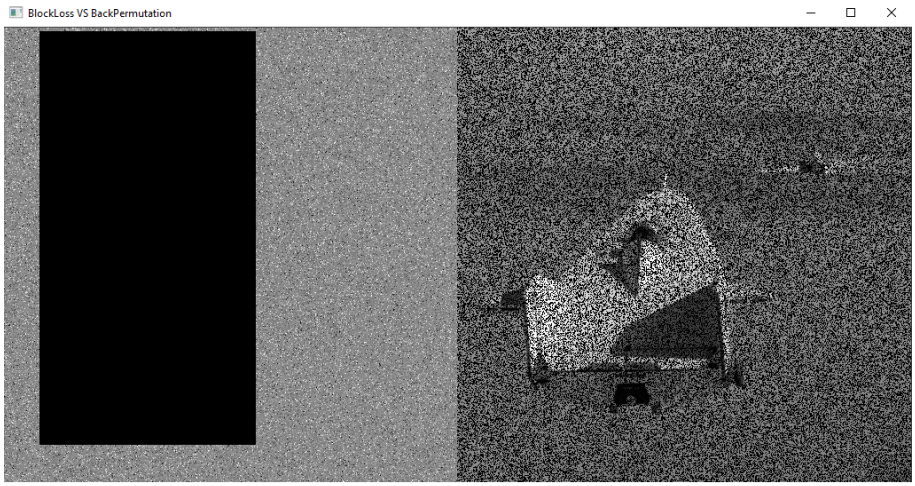

Figure 3: distorted + blockLoss VS reconstructed

This time we can clearly see that the permutation is completely random because we don't recognize at all the image after a permutation. However, this permutation is deterministic because it depend on a random seed depending of the size of the image. This manipulation simplify the backward permutation. We just take the image and put all the pixels value in a vector, do a random but deterministic shuffle and put it back in a 2d matrix image.

In regards of the observed result, we can clearly see that the more the block loss dimensions are big the more our reconstructed image will be noised because there will be more black pixels in it.

2.3 Exercise 3

Now, we will generate a noisy image by adding an uniformly distributed value taken in specified intervals to analyse the corresponding histograms and PSNR. So lets compute the noised image with intervals: [-1,1],[-5,5],[-10,10],[-15,15] and plot the corresponding psnr:

Here are the obtained images:

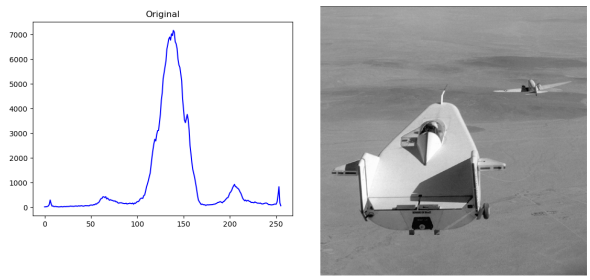

Figure 4: Original Image

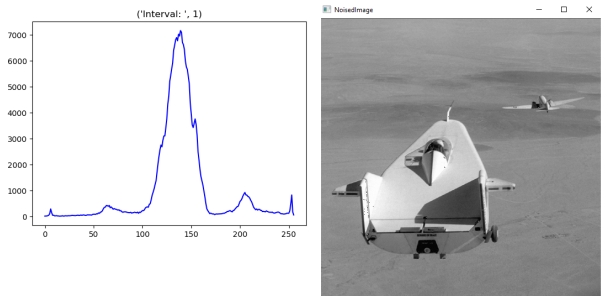

Figure 5: [-1,1]

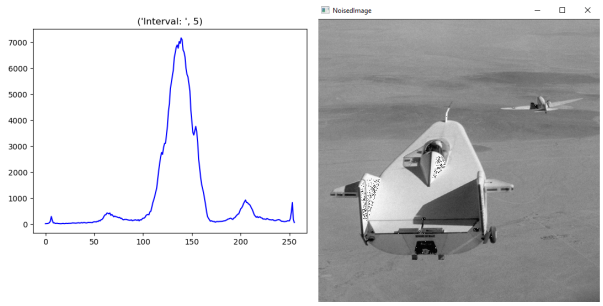

Figure 6: [-5,5]

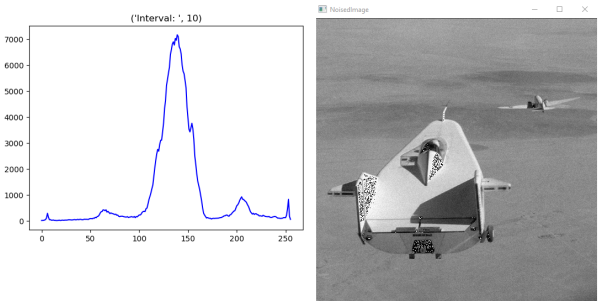

Figure 7: [-10,10]

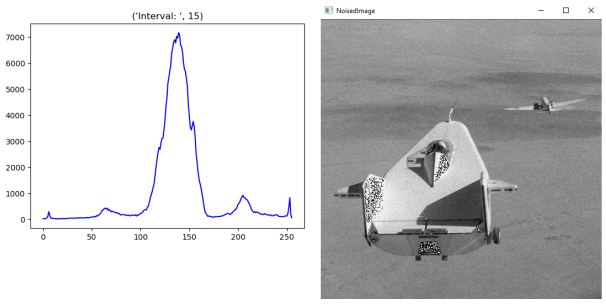

Figure 8: [-15,15]

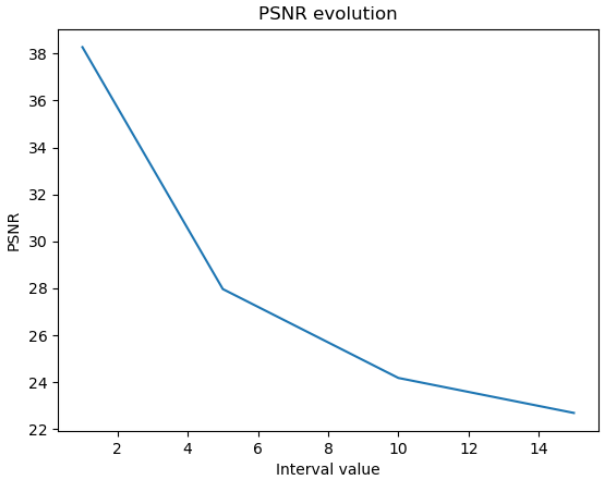

Figure 9: PSNR evolution

Firstly, we can clearly see that the histogram doesn't really change because we add value taken from an uniform (centered to 0) distribution so the global distribution of the image will not change. In regards of the images, we mainly see the differences in the black or white part of the image for bigger interval because the HVS is more sensible to the contrast between black and white but if we observe carefully there is also slight changes on the grey part of the image which is less uniform. Finally, despite the fact that the histogram is not changing we clearly see that the more the interval is big the less the PSNR is good because the PSNR use the MSE between two image which take the squared distance between the pixels at the same location in the images.

3 Classical Cryptography

In this section we will focus on the given "AES python.ipynb", where we implement the AES encryption on an image where we encrypt each 4x4 block. The goal is to see the effect of an attack on classical cryptography. First, we will simulate the case where the encrypted image suffers from a single bit error, then we will see the effect of adding Additive White Gaussian Noise to the encrypted image which is public.

Here is the original encryption and decryption without any attack:

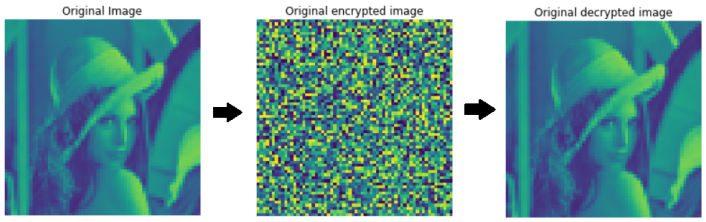

Figure 10: Original decryption

Now, we will modify one pixel from the encrypted image to see the impact on the decrypted image.

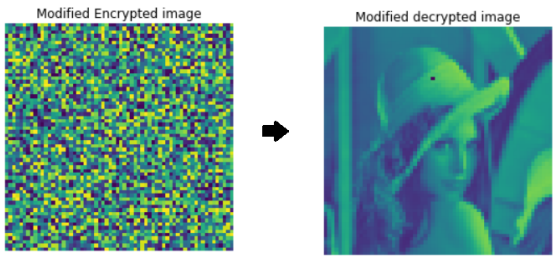

Figure 11: One bit error decryption

We observe that even a single bit error which can even occur during the transmission have an impact on the decryption because we clearly see a black hole in the image. Moreover, the difference in the cipher is very hard to see which is a problem because the receiver doesn't really know if someone modified the message. So its not really robust to one pixel modification.

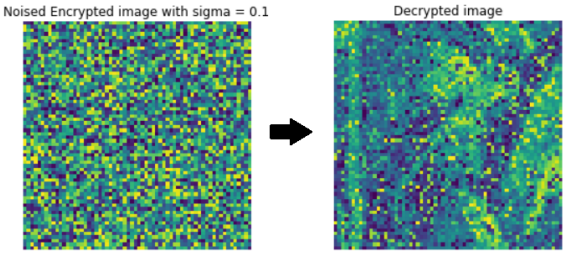

Figure 12: White Gaussian Noise

Finally, we add a white Gaussian noise with a 0.1 standard deviation. This time there is a slight modification on every pixel of the encrypted image. Like the previous case, its hard to see the difference in the encrypted image but the decrypted image is completely different. So even a little difference in the pixels have a big impact on the output. Therefore, AES is not robust to this attack either.

4 Basic Data Hiding

In this section, we will implement a simple watermarking technique based on the most significant bits and the least significant bits. The idea is to put a secret image inside another by replacing the least significant bit of the cover image by the most significant bits of the secret image following this given table:

	MSB							LSB
Red	$C_{r,8}$	$C_{r,7}$	$C_{r,6}$	$C_{r,5}$	$C_{r,4}$	$S_{r,8}$	$S_{r,7}$	$S_{r,6}$
Green	$C_{g,8}$	$C_{g,7}$	$C_{g,6}$	$C_{g,5}$	$C_{g,4}$	$C_{g,3}$	$S_{b,8}$	$S_{b,7}$
Blue	$C_{b,8}$	$C_{b,7}$	$C_{b,6}$	$S_{g,8}$	$S_{g,7}$	$S_{g,6}$	$S_{b,6}$	$S_{b,5}$

Figure 13: Scheme table

For each different channels we have different rules, C being the cover and S the secret image. We can already observe that we will loss more information on the green channel than the blue or red channel.

The original images are those two images:

Figure 14: Cover VS Secret

Here is the result for the stego image and the reconstructed secret:

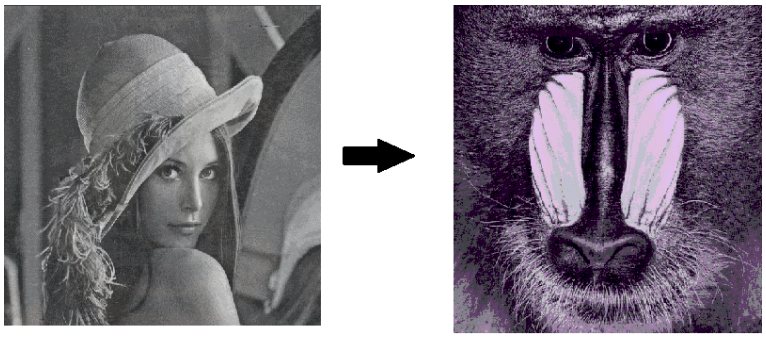

Figure 15: Stego VS Reconstruction

If we look carefully we can see some part of the monkey (mostly the eyes) in the stego image. But the most important detail is that the cover image has lost most of his colors and have different values of grey. That's because by following the given table, each of the three channel has a very similar value for each pixel. Furthermore, when we reconstruct the secret by extracting the most significant bits, we obtain a light purple image where we clearly distinguish our hidden monkey. As we said earlier, we lost information on 5 bits for the red channel, 6 for the green one and 3 for the blue one (For every pixels). So that's why we have a blue dominance on the image and almost no green.