Visual CryptograPY

Giulio Ursino

1 Introduction

Visual cryptography represents a particularly interesting cryptographic scheme due to its peculiar decryption algorithm, which occurs instantly and mechanically, without the aid of a computer. It was introduced by Moni Naor and Adi Shamir in 1994 in [3] and, initially, was applied to binary images (black and white). The general principle studied the "k out of n" case, in which an image was decomposed into n semi-transparent *shares* and could be reconstructed by superimposing any k of these. We will focus on the 2 out of 2 case, but we will try to extend the problem to grayscale images. The latter is a much less treated problem; an overview of the main methodologies discussed so far follows. One of the articles proposing a possible solution to the problem is [1], which introduces an algorithm for the visual cryptography of grayscale and color images. In this scheme, however, the decryption algorithm does not occur mechanically, but digitally, through an XOR-based algorithm. Nevertheless, it remains an interesting solution because the shares generated by the encryption algorithm do not appear as random images but as meaningful images chosen by the user (figure 1.1).

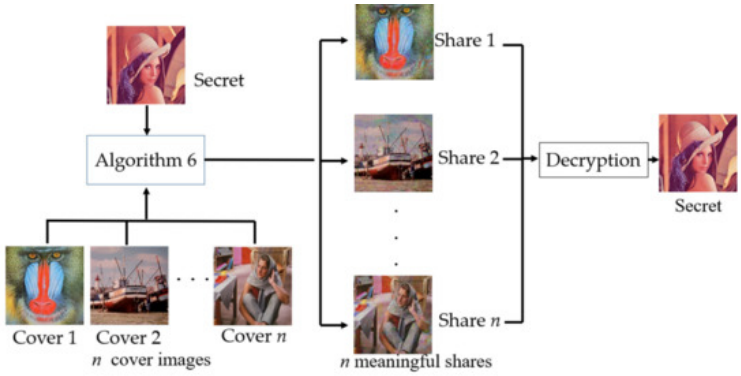


Figure 1.1: Yu-Hong Chen algorithm

Another solution is proposed in [4]; the proposed algorithm decomposes the grayscale image into the various bit-planes that constitute it and applies the visual cryptography algorithm for binary images on each of them, obtaining two shares for each of the bit-planes, which are then recomposed into two grayscale shares from which no information regarding the original image leaks. Even in this case, however, decryption must occur digitally.

An algorithm that preserves the decryption property of the scheme proposed by Naor and Shamir is presented in [2]. The proposed scheme uses ordered dithering techniques to approximate the input image with a binary image and, on it, applies the visual cryptography algorithm for binary images. The algorithm requires a considerable expansion of the image in terms of size: in figure 1.2 it is possible to see the shares relative to a single pixel of intensity 0 (the grey area is constituted by the alternation of white and black pixels).

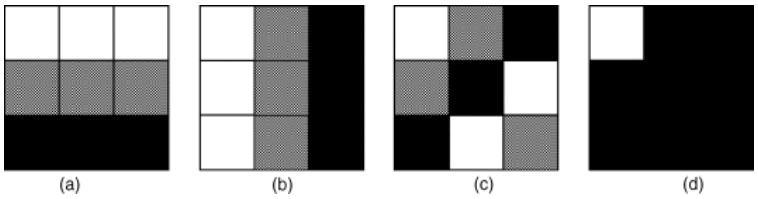


Figure 1.2: (a)(b)(c) respectively the patterns relative to shares 1, 2 and 3; (d) the pixel reconstructed by superimposing the shares

Below, two schemes for visual cryptography of 4 and 8-level grayscale images are proposed. They replace the two 2×2 matrices of the Naor and Shamir model with 4 (resp. 8) 3×3 (4×4) matrices, one for each gray intensity. The proposed schemes preserve the convenient and immediate decryption algorithm foreseen by the system for binary images.

2 Scheme for Binary Images

The "k out of n" model proposed by Naor and Shamir provided for the decomposition of the binary input image into n shares such that the superposition of any k of these determined the appearance of the input image. We will focus on the simpler 2 out of 2 scheme, in which the image is divided into 2 shares to be superimposed. The idea is to use complementary 2×2 matrices consisting of transparent and opaque pixels; the superposition of two complementary grids determines the perception of a black pixel, while the superposition of two equal grids determines the perception of a grey pixel (figure 2.1).

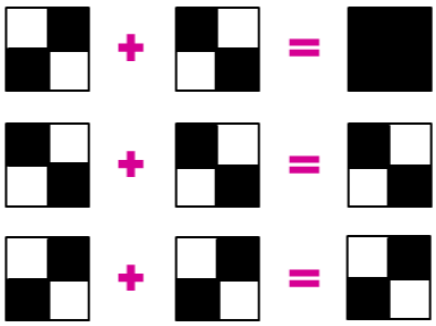


Figure 2.1: Visual cryptography scheme for binary images

Therefore, if the input image has dimensions $n \times m$, the two shares obtained from the application of the algorithm will have dimensions $2n \times 2m$, since a 2×2 matrix must correspond to each pixel. Suppose, for simplicity, we associate a white or black pixel to each of the two complementary matrices (figure 2.2). Then, we can build the two shares as $n \times m$ images in the same domain as the starting image, and then convert them into the superimposable shares by replacing each pixel with the corresponding matrix.

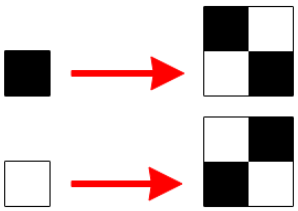


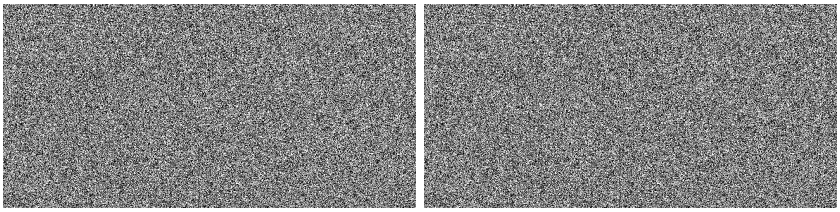
Figure 2.2: Conversion from pixel to semi-transparent matrix

From now on, I will refer to the two shares as "key" and "encrypted image". Any key can be used, provided it has the same dimensions as the input image; for security reasons, it is good practice for it to be generated following a random algorithm, so that each pixel has a 50% probability of being white or black. The encrypted image is constructed starting from the key and the plain image. Following the previous reasoning, we want a black pixel of the plain image to correspond to two distinct pixels in the key and encrypted image (since the superposition of two complementary matrices results in a totally black matrix) and a white pixel to correspond to two equal pixels (partially opaque matrix, perceived by the eye as grey). We can, therefore, calculate the encrypted image pixel by pixel by applying a simple operation on the corresponding pixels of the plain image and the key. In particular, for every white pixel of the plain image, we report the corresponding pixel of the key on the encrypted image, and for every black pixel, we report the negated value of the corresponding pixel of the key. Assuming we associate the value 1 to every white pixel and the value 0 to

every black pixel, it is evident how the operation in question is simply a negated version of the bitwise XOR. The cryptographic scheme in question, therefore, carries the same level of security as one-time-pad, whose encryption algorithm requires a key of the same dimension as the input and calculates the ciphertext as the result of the bitwise XOR between the two. We have, therefore, a guarantee that the two shares do not carry any relevant information about the starting data. The encrypted message is therefore calculated by applying the XOR pixel by pixel between the key and the negative of the input image.
A brief Python implementation of the encryption algorithm follows:

```
def encrypt_binary(img, key):
    bw_img = rgb_to_binary(img)
    neg_img = negative(bw_img)

    cypher = np.zeros((key.shape[0], key.shape[1],3), np.uint8)

    for i in range(key.shape[0]):
        for j in range(key.shape[1]):
            key_px = float(key[i][j][0])
            img_px = float(neg_img[i][j])
            cypher[i][j] = int(abs(key_px - img_px)) #XOR

    return cypher
```

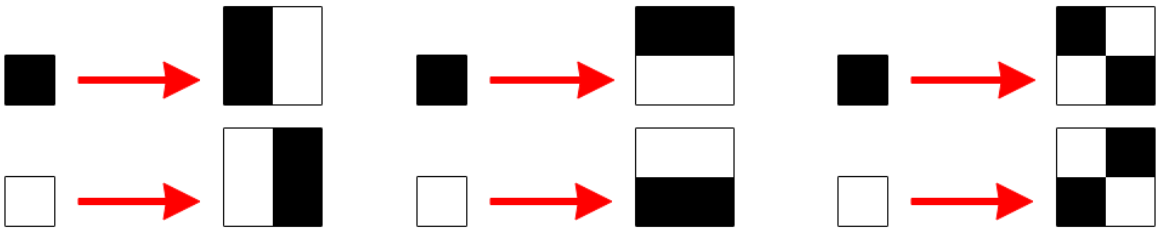
Figure 2.3 shows an example of application of the encryption algorithm on a binary image. From the two shares thus generated, if taken individually, it is not possible to trace back to the original information or parts of it.



(a) Plain image (b) Encrypted image (c) Random key

Figure 2.3

Currently, the two shares are not superimposable; the reconstruction of the starting image would require a second application of the XOR algorithm between the two. To conclude, it is therefore necessary to apply the bitwise transformation (figure 2.2) on the key and encrypted image. The type of filling shown in figure 2.2 is not the only one possible, as can be seen below (figure 2.4):



(a) (b) (c)

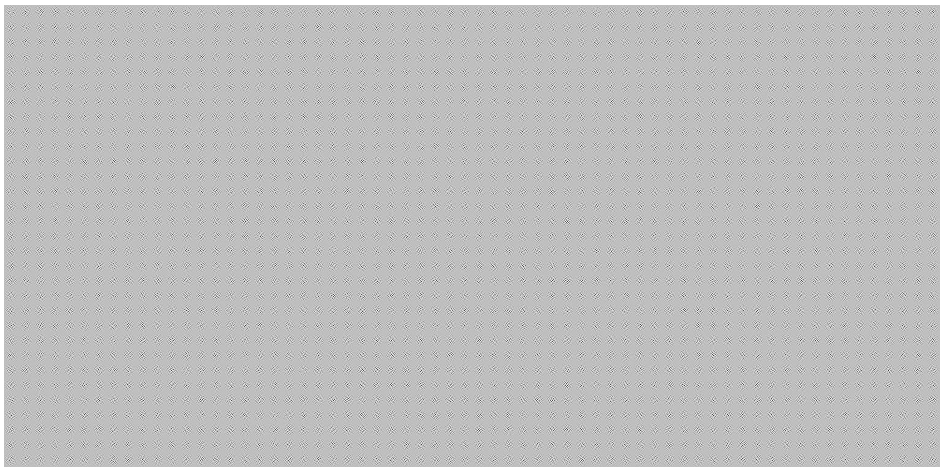
Figure 2.4: Possible conversion schemes for binary images

A Python implementation of the transformation shown in figure 2.4c follows:

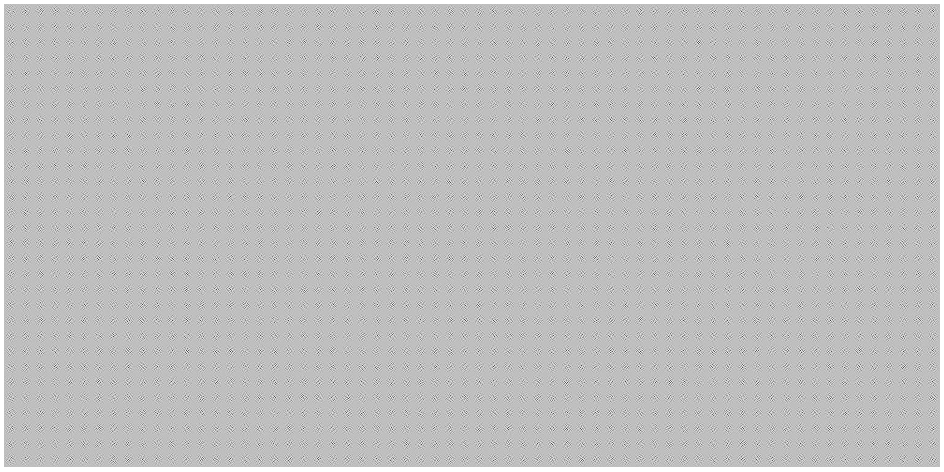
```
def VC_conversion_diagonal(img):
    VC_img = np.zeros((img.shape[0]*2,img.shape[1]*2,4), np.uint8)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            img_px_val = img[i][j][0]

            VC_img[i*2][j*2][3] = 255 - img_px_val

            VC_img[i*2][j*2 + 1][3] = img_px_val

            VC_img[i*2 + 1][j*2][3] = img_px_val

            VC_img[i*2 + 1][j*2 + 1][3] = 255 - img_px_val
    return VC_img
```

N.B. In the image thus generated, an opaque pixel will be of the type $[0, 0, 0, 255]$ and a transparent pixel will be of the type $[0, 0, 0, 0]$.

The application of the `VC_conversion_diagonal()` algorithm on the key and encrypted image previously obtained returns the two final semi-transparent shares (figure 2.5), which can be superimposed in order to reconstruct the initial data.



(a) Share relative to the encrypted image



(b) Share relative to the key

Figure 2.5

The superposition of the two shares can happen physically by printing on transparent paper, as well as with the aid of graphics software, by making the two png images match perfectly. In this case we will simulate the superposition using a simple Python script:

```
def superimpose(img1, img2):
    output_img = np.zeros_like(img1)

    for i in range(img1.shape[0]):
        for j in range(img1.shape[1]):
            if img1[i][j][3] == 0 and img2[i][j][3] == 0: # both pixels are transparent
                output_img[i][j] = [255, 255, 255, 255]
            else:
                output_img[i][j][3] = 255

    return output_img
```

In this case, the pixels given by the superposition of two transparent pixels are made totally white in order to simulate the superposition of the two shares on a white background. Below (figure 2.6) it is possible to observe the result given by the superposition of the two semi-transparent shares:

Figure 2.6: Reconstructed image

By zooming in on the grey area it is possible to observe how it is, in reality, constituted by the alternation of many white and black pixels.

As already mentioned, the reported scheme is completely similar to one-time-pad and for this reason it is fundamental that the same key is never used to encrypt two distinct messages. Suppose we want to encrypt two messages m_1 and m_2 and do it using the same key k both times.
Then we would calculate:

$$c_1 = m_1 \oplus k$$

$$c_2 = m_2 \oplus k$$

In this way, however, having access to both c_1 and c_2, it would be possible to calculate:

$$\hat{c} = c_1 \oplus c_2 = m_1 \oplus m_2$$

Since this procedure does not allow obtaining either of the two messages totally in clear, the serious security problem associated with adopting such behavior might not be evident. However, it should be noted how, from $\hat{c}$, patterns or other important information relative to the two messages could leak. This problem becomes evident if applied to our context of use. Suppose, in fact, we encrypt the following two images using the same key:

(a) (b)

Figure 2.7

Following all the procedures of the algorithm for both images and superimposing the two shares relative to the encrypted images, the following result is obtained:

Figure 2.8: Superposition of the two shares encrypted with the same key

3 Scheme for 4-Level Grayscale Images

The proposed scheme is very similar to the one adopted for binary images. In this context, it is necessary to adapt the encryption algorithm to a domain of 4 elements, compared to the previous one which operated on the set $\{0, 1\}$. Another crucial characteristic of the new scheme is the procedure of converting the shares into superimposable semi-transparent images. As before, we want to associate a different matrix composed of opaque and transparent pixels to each gray level. In particular, we want these matrices to guarantee a specific property: let M_1, M_2, M_3, M_4 be the 4 matrices, we want to guarantee that $\forall\, 1 \leq i \leq 4,\ 1 \leq n \leq 4 \Rightarrow \exists\, 1 \leq j \leq 4\ :\ M_i \ominus M_j$ presents exactly n transparent pixels, where $M_i \ominus M_j$ indicates the result of the superposition of the matrices M_i and M_j. That is, the gray levels are simulated through the number of transparent pixels contained within the matrix obtained after the superposition and, in particular, if the matrices present k opaque pixels, from the superposition matrices containing a number variable from k to $k + 3$ opaque pixels must derive. More precisely, we want that, however one of the matrices is chosen, there must exist one and only one that, superimposed on the first, returns each of the different possible intensities. To achieve this goal, 3×3 matrices colored with 5 opaque pixels and 4 transparent pixels were used. The scheme relative to the matrices chosen for the transformation follows:

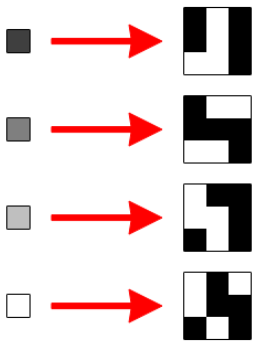


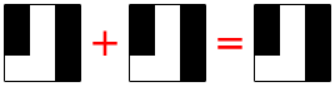
Figure 3.1: Conversion scheme for shares relative to 4-level grayscale images

From the scheme, it can be observed how the property is respected: in fact, chosen any of the 4 matrices and chosen a value $n \in \{1, 2, 3, 4\}$, there always exists a matrix that, superimposed on the first, returns one composed of exactly n transparent pixels and $9 - n$ opaque pixels. Below it is possible to see how, from one of the matrices, we are able to obtain each of the gray levels:

(a) Matrix of intensity 1

(b) Matrix of intensity 2

(c) Matrix of intensity 3



(d) Matrix of intensity 4

Figure 3.2

Let us now enter more into detail regarding the encryption algorithm. Observing the scheme reported in figure 3.1 and associating a value $\in \{1,2,3,4\}$ to each of the gray intensities (with the value 1 associated to the darkest pixel and the value 4 associated to the lightest pixel), it is possible to derive an encryption algorithm of the type $Enc : \{1,2,3,4\} \times \{1,2,3,4\} \to \{1,2,3,4\}$ in which $Enc(m,k) = c \iff M_c \ominus M_k$ presents exactly m transparent pixels. A detailed definition of the encryption function follows:

$$\begin{array}{llll} Enc(1,1)=4 & Enc(1,2)=3 & Enc(1,3)=2 & Enc(1,4)=1 \\ Enc(2,1)=3 & Enc(2,2)=4 & Enc(2,3)=1 & Enc(2,4)=2 \\ Enc(3,1)=2 & Enc(3,2)=1 & Enc(3,3)=4 & Enc(3,4)=3 \\ Enc(4,1)=1 & Enc(4,2)=2 & Enc(4,3)=3 & Enc(4,4)=4 \end{array}$$

From the representation just illustrated, it can be inferred that the proposed scheme enjoys the property of perfect secrecy, of which a brief description follows:

Def: Given a cryptosystem $SE = (KeyGen, Enc, Dec)$, with $KeyGen$ key generation algorithm, $Enc : K \times M \to C$ encryption algorithm and $Dec : K \times C \to M$ decryption algorithm, SE is said to be perfectly secure if:

$$\forall\, m_1, m_2 \in M, c \in C \Longrightarrow P[Enc(k, m_1) = c] \;=\; P[Enc(k, m_2) = c]$$

In this context, we indicate with K the key space, with M the message space and with C the ciphertext space; in our case, $M = K = C = \{1,2,3,4\}$. The notation $P[Enc(k,m) = c]$ indicates the probability as k varies, fixed m and c, that $Enc(k,m) = c$. Shannon's theorem now comes to our aid:

Shannon's Theorem: Let $SE = (KeyGen, Enc, Dec)$ and let $|M| = |K| = |C|$. SE offers perfect secrecy if and only if:
1. $\forall\, k \in K$, k is chosen with probability $1/|K|$
2. $\forall\, m \in M, c \in C, \exists_1 k \in K \mid Enc(k,m) = c$

Since both properties are guaranteed, we conclude that the proposed system enjoys perfect secrecy, exactly like One-Time-Pad. This is not particularly surprising; converting each value to base 2, in fact, it can be seen that the proposed encryption algorithm simply consists of calculating the bitwise XOR between the two input values:

$$1 \to 01 \quad 2 \to 10 \quad 3 \to 11 \quad 4 \to 100$$

$$\begin{array}{llll} 01 \oplus 01 = 00 & 01 \oplus 10 = 11 & 01 \oplus 11 = 10 & 01 \oplus 00 = 01 \\ 10 \oplus 01 = 11 & 10 \oplus 10 = 00 & 10 \oplus 11 = 01 & 10 \oplus 00 = 10 \\ 11 \oplus 01 = 10 & 11 \oplus 10 = 01 & 11 \oplus 11 = 00 & 11 \oplus 00 = 11 \\ 00 \oplus 01 = 01 & 00 \oplus 10 = 10 & 00 \oplus 11 = 11 & 00 \oplus 00 = 00 \end{array}$$

We come, therefore, to the implementation of the model. Also in this case, the system takes care of quantizing the color space of the image in order to convert the latter into a 4-level grayscale image. In order to make the result more pleasing and faithful to the starting image, the quantization error is redistributed according to the Floyd-Steinberg ordered dithering algorithm.

(a) Original image

(b) Quantized image

(c) Quantized image with dithering

Figure 3.3

A brief python implementation of the quantization algorithm used follows:

```
def quantize_and_dither(img, N=4): #N = number of gray levels (should be a power of 2)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            old_val = img[i][j][0]
            new_val = math.ceil(((old_val+1)/256.0)*N)*256/N - 1 #quantization
            error = new_val - old_val

            img[i][j] = new_val

            #floyd-steinberg dithering
            if(j+1 < img.shape[1]):
                img[i][j+1] = saturate(img[i][j+1][0] - error * 7/16)
            if(i+1 < img.shape[0]):
                img[i+1][j] = saturate(img[i+1][j][0] - error * 5/16)
                if(j-1 >= 0):
                    img[i+1][j-1] = saturate(img[i+1][j-1][0] - error * 3/16)
                if(j+1 < img.shape[1]):
                    img[i+1][j+1] = saturate(img[i+1][j+1][0] - error * 1/16)
    return img
```

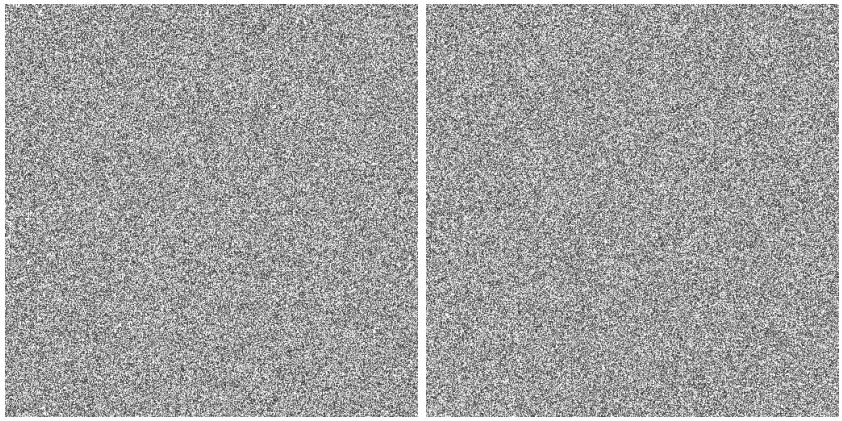
Before being passed to the `quantize_and_dither()` algorithm, the image should be converted into a 256-level grayscale image. The quantization formula (`new_val = ...`) converts each pixel based on the interval it belongs to according to the following conversion scheme:

$$[0, 63] \to 63 \qquad [64, 127] \to 127 \qquad [128, 191] \to 191 \qquad [192, 255] \to 255$$

As for the binary scheme, the key is generated randomly; in this case, it is generated as an image of the same dimension as the input image, in which each pixel is assigned, with equal probability, one of the values $\in \{63, 127, 191, 255\}$ (`randint(1,4)*64 - 1`). At this point the image and the key can be given as input to the following encryption algorithm:

```
def encrypt_image_4levels(img, key):
    cypher = np.zeros((key.shape[0], key.shape[1],3), np.uint8)

    for i in range(key.shape[0]):
        for j in range(key.shape[1]):
            key_val = key[i][j][0]
            img_val = img[i][j][0]

            cypher[i][j] = encrypt_4levels((int(key_val)+1)//64, (int(img_val)+1)//64)*64 - 1

    return cypher
```

The function `encrypt_4levels()` receives two integers $\in \{1, 2, 3, 4\}$ and returns a value chosen from the same set following the encryption algorithm seen previously in this chapter. Below (3.4) it is possible to observe the two shares obtained starting from the image 3.3c:



(a) Encrypted image (b) Random key

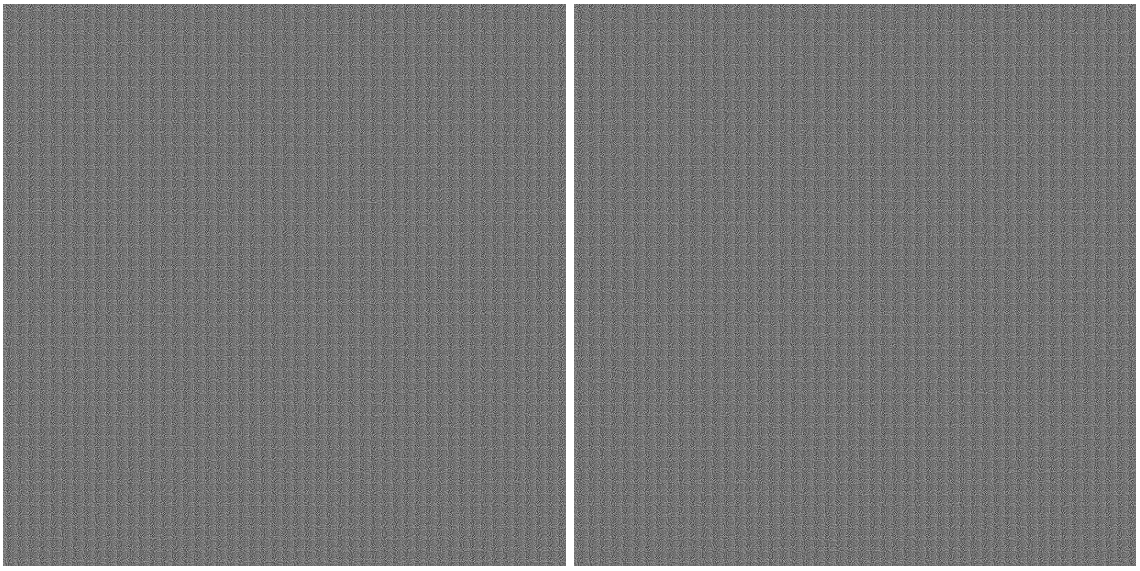
Figure 3.4

Similarly to what was seen in the scheme for binary images, on the two shares it is necessary to apply the transformation seen in figure 3.1 in order to make the data reconstructible by their superposition. The implementation of the algorithm in question follows:

```
greyscale_4levels = [[[1,0,1],
                      [1,0,1],
                      [0,0,1]], [[1,0,0],
                                 [1,1,1],
                                 [0,0,1]], [[0,1,1],
                                            [0,0,1],
                                            [1,0,1]], [[0,1,0],
                                                       [0,1,1],
                                                       [1,0,1]]]

def VC_conversion_greyscale_4levels(img):
    VC_img = np.zeros((img.shape[0]*3,img.shape[1]*3,4), np.uint8)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            px_val = img[i][j][0]
            index = (px_val + 1)//64 - 1
            for k in range(9):
                row = math.floor(k/3)
                column = k % 3
                VC_img[3*i+row][3*j+column][3] = greyscale_4levels[index][row][column]*255 #sets
                    transparency to either 0 or 255 (all pixels are actually black)

    return VC_img
```

The list `greyscale_4levels` contains the 4 conversion matrices.
N.B. The opaque pixels are represented by the value 1 and the transparent pixels by the value 0.

The result of the application of the conversion algorithm on the two shares follows:



(a) Share relative to the encrypted image (b) Share relative to the key

Figure 3.5

The procedure of generating the two shares is therefore concluded. At this point it will be sufficient to superimpose them in order to reconstruct the secret. Also in this case we simulate the superposition of the two images using the `superimpose()` algorithm illustrated in the previous chapter; the result follows:

Figure 3.6: Reconstructed image by superposition of the two shares

The imperative remains not to reuse the same key more than once, in order to prevent the cryptographic scheme from losing robustness. The use case relative to the encryption of the two following images is reported below:

(a) (b)

Figure 3.7

Encrypting both images using the same key and superimposing the encrypted shares, the following result is obtained:

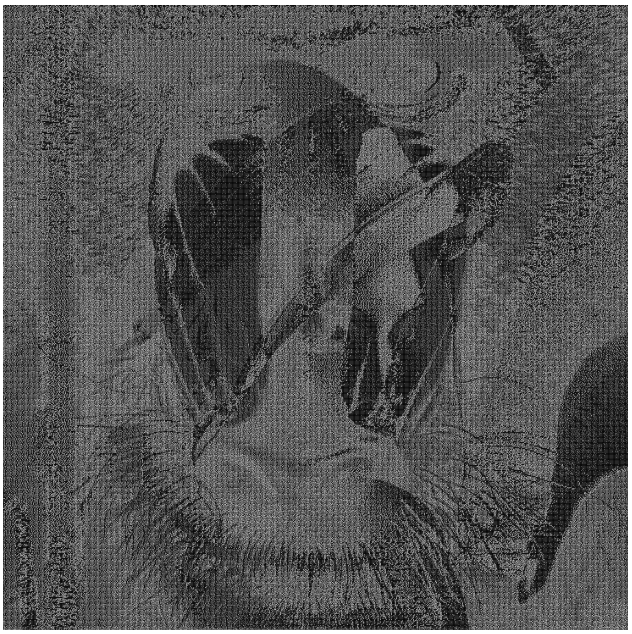


Figure 3.8: Superposition of the two shares encrypted with the same key

4 Scheme for 8-Level Grayscale Images

The scheme adopted to extend the domain of the images to 8 gray intensities follows directly from the one for quantized images at 4 levels. Also in this case we associate a different semi-transparent matrix to every gray intensity. In order to determine the minimum size necessary for the construction of the 8 matrices, it should be noted how each of them must present at least 7 transparent pixels (from 0 to 7 of these can be covered to simulate the different intensities) and at least 7 opaque pixels (to cover the transparent ones). The matrices must therefore contain at least 14 pixels, from which follows the choice of 4×4 matrices for a total of 16 pixels. The matrices were built starting from a slightly modified version of those used for the 4-level scheme; to each of them we add two different paddings along the upper border and the right lateral border, so as to obtain the 8 desired matrices, for which - similarly to what is required in chapter 3 - it holds that $\forall\, 1 \leq i \leq 8,\ 1 \leq n \leq 8 \Rightarrow \exists\, 1 \leq j \leq 8\ :\ M_i \ominus M_j$ presents exactly $n-1$ transparent pixels.

N.B. In the previous scheme we spoke of n transparent pixels, now of $n-1$; this is simply a choice linked to the perception of brightness of the final result of the superposition. As stated previously, only 14 of the 16 pixels that make up a 4×4 matrix are necessary and functional to the correct functioning of the visual cryptography mechanism; two pixels remain, each of which can be made opaque or transparent in all 8 matrices, in order to regulate the luminous intensity perceived in the final result.

The chosen conversion scheme follows:

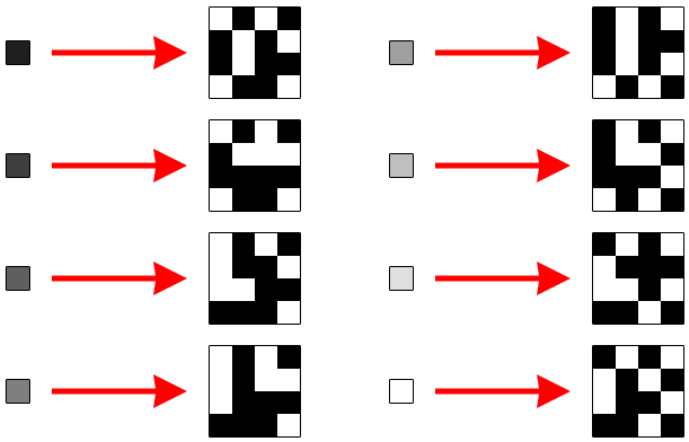


Figure 4.1: Conversion scheme for shares relative to 8-level grayscale images

It is also worth spending a few words on the new encryption algorithm, completely similar to the previous one although it works on a wider domain. It is a function of the type $Enc : \{1, 2, \ldots, 8\} \times \{1, 2, \ldots, 8\} \to \{1, 2, \ldots, 8\}$ in which $Enc(m, k) = c \iff M_c \ominus M_k$ presents exactly $m - 1$ transparent pixels. For brevity, the detailed definition of the result of the function for every pair of values belonging to the domain is not reported, which can however be deduced from the conversion scheme in figure 4.1. We note an interesting property of both encryption functions reported for grayscale images:

$$Enc(m, k) = c \iff M_m \ominus M_k \text{ presents exactly } c - 1 \ (c \text{ for the 4-level scheme}) \text{ transparent pixels}$$

This shows how the encryption function is the inverse function of itself; reapplying it on the encrypted image - using the same key - it is possible to reconstruct the starting data.
Shannon's theorem reported in the previous chapter comes to our aid again, to demonstrate that the new proposed scheme, in a completely analogous way to the first, enjoys perfect secrecy according to the definition used in the cryptographic field.
Let us now see the execution flow of the cryptographic scheme on a given image. Also in this case, after quantization, we apply Floyd-Steinberg in order to make the result more appreciable:

(a) Original image

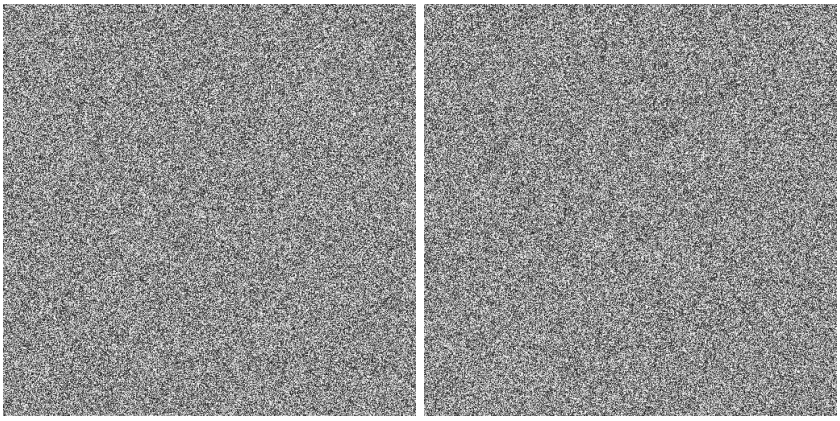
(b) Quantized image

(c) Quantized image with dithering

Figure 4.2

The function used to apply quantization and dithering is the same reported in chapter 3 and also in this case,

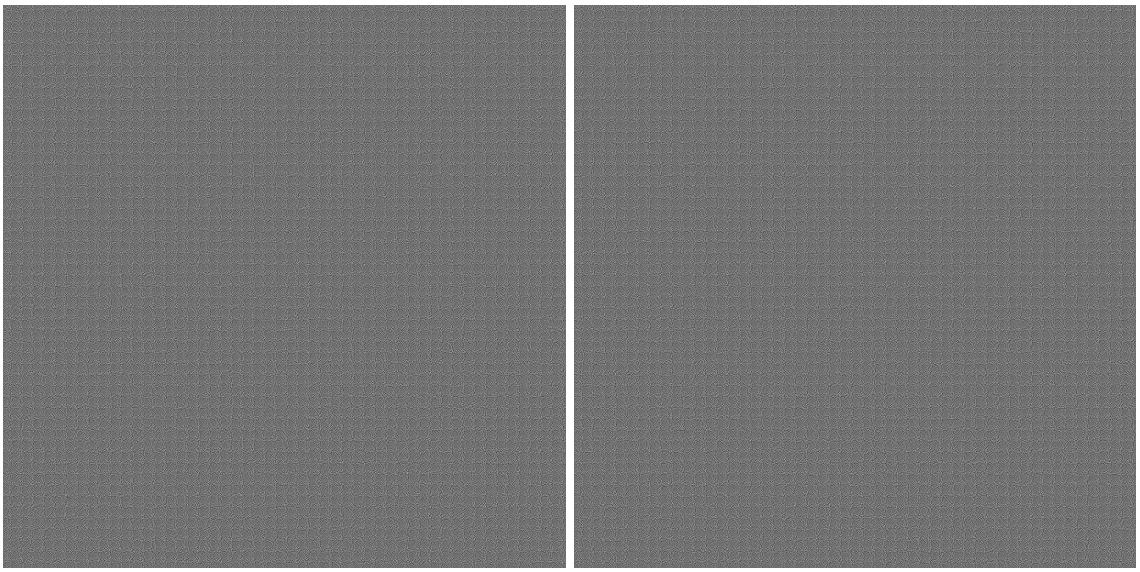
the key is generated by assigning to each of its pixels a value chosen following a uniform distribution on the domain.
Once the key is generated, we can apply the encryption algorithm on the given image (4.2c); the two generated shares are reported below:



(a) Encrypted image (b) Random key

Figure 4.3

The key generation algorithm and the encryption algorithm are completely analogous to those reported for the 4-level scheme and the same applies to the algorithm that applies the conversion shown in figure 4.1 on the shares. The shares obtained after the application of the conversion algorithm follow, whose dimensions are quadrupled compared to the shares in figure 4.3:



(a) Share relative to the encrypted image (b) Share relative to the key

Figure 4.4

It only remains for us to superimpose the two shares and observe the final result (4.5b). The superposition is again simulated through the tool defined in the previous chapter, but can also occur through the use of graphics software or by printing on transparent paper. Below the result is reported compared with that obtained from the 4-level scheme (figure 3.6):

(a) 4 levels (b) 8 levels

Figure 4.5: Reconstructed images by superposition of the two shares

A leap in quality is observed compared to the result obtained from the previous scheme, at the cost of a significant increase in the resolution of the shares. The input image had a resolution equal to 512×512, compared to the 1536×1536 of the shares of the 4-level model and the 2048×2048 of the 8-level scheme. Similarly to the previous schemes, the serious vulnerability deriving from the reuse of the same encryption key for distinct images remains. Using the 8-level scheme, we repeat the experiment seen in the previous chapters on images 3.7a and 3.7b, we obtain the following result:

Figure 4.6: Superposition of the two shares encrypted with the same key

Appendix

In order to facilitate the use of the software, a graphical interface was developed using the Python Gradio library. All the material cited in this report is available on github at the following link:

`https://github.com/JeeYou02/visual_cryptography`

References

[1] Y.-H. Chen and J. S.-T. Juan. Xor-based (n, n) visual cryptography schemes for grayscale or color images with meaningful shares. *Applied Sciences*, 12(19), 2022.

[2] C.-C. Lin and W.-H. Tsai. Visual cryptography for gray-level images by dithering techniques. *Pattern Recognition Letters*, 24(1):349–358, 2003.

[3] M. Naor and A. Shamir. Visual cryptography. In A. De Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, pages 1–12, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[4] D. Taghaddos and A. Latif. Visual cryptography for gray-scale images using bit-level. *Journal of Information Hiding and Multimedia Signal Processing*, 5:90–97, 01 2014.