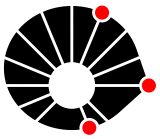
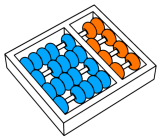




Overview on Visual Cryptography and Its Potential Uses

A. Gomes *H. Pedrini*

Relatório Técnico - IC-PFG-18-21
Projeto Final de Graduação
2018 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Overview on Visual Cryptography and Its Potential Uses

Alan R. Gomes* Helio Pedrini†

Abstract

In this technical report, it was presented an overview on the Visual Cryptography topic, concerning a few studies and potential applications based on the concepts described in academic literature. In addition, it was implemented two proofs of concepts in order to test and get familiar with the subject. The first was firstly presented by Naor and Shamir [1], and it consists on a hidden image being showed by overlapping two ciphered shares that are perceived as pure noise when seen individually. The second consists on uncovering n images hidden in n shares by overlapping them with a key share. The latter concept, studied by Weir and Yan [8], was explored in this report in order to elaborate set definitions for what we called virtual pixels and study their potentialities and limitations.

1 Introduction

Visual cryptography was first described in a paper written by Naor and Shamir [1], in which the authors include probabilistic models to increase the complexity of the generated shares, thereby making it harder - not to say impossible - to decrypt hidden content. For example, one might have two (or more) shared shares and overlap them carefully to produce the hidden image, exposing it with enough contrast that our visual system can interpret it. The explanation of how to uncover the images is simple and straightforward, in elegant opposition with the potential for enormously intricate encryption patterns and applications.

In addition to this simplicity, the method proposed by Naor and Shamir [1] for encrypting the content is quite secure. When separated, the shares are interpreted by our visual system as pure noise. Indeed, if someone obtains just one of the shared shares, it is impossible to guess what the original image is behind it. The present aim is to investigate the methods of visual cryptography and to advance the concept forward via exploration of its potential usages and applications.

The remainder of the text is organized as follows. Section 2 briefly presents some approaches related to the topic under investigation in this work. Section 3 describes the main aspects of the developed methodology. Section 4 presents our visual cryptography algorithm. Section 5 reports the results obtained in this work. Section 6 presents some concluding remarks.

*Institute of Computing, University of Campinas, 13083-852, Campinas, SP, Brazil
†Institute of Computing, University of Campinas, 13083-852, Campinas, SP, Brazil

2 Related Work

The work initiated by Naor and Shamir [1] also inspired many others in literature about Visual Cryptography, extending its capacities, such as image contrast [9], security, and pragmatical uses. Nakajima and Yamagushi [2], in 2002, continued the exploration of this matter applying a similar concept to natural images, displaying hidden images with gray-scale colors when overlapping ciphered shares. Moreover, the elaboration of a related process was also implemented for colored images by Hou [3], eventually offering three different methodologies for ciphering a colored (or gray-scale) image. Similarly, Murthy and Rao [4] also proposed other three modes for visual cryptography for colored images, and one mode for its access structure.

Beyond the capabilities of applying Visual Cryptography for colored image, a few researchers, such as Marwaha et al. [5], in 2010, and Cimato and Yang [6], in 2017, included visual cryptography into steganography, improving content protection for images. In addition, still concerned about content security, halftone techniques were used by Zhou et al. [7] to create watermarks visible through the generated shares when overlapped.

The concern of making the Visual Cryptography flexible for other real applications also were present in the works of Weir and Yan [8], published in 2009, which included concepts of a key share uncovering the image hidden by many others. In fact, the concept created by them was the most explored in this report in order to elaborate useful and/or playful contents using Visual Cryptography.

3 Methodology

Visual crytopgraphy was implemented here using Python language, version 3.5.2, in addition to the influence of public, open-source libraries such as OpenCV (version 3.4.2) and Numpy (version 1.15.1). The libraries are extremely useful to this work as a result of their extended communities and resource-rich documentation.

The resulting materials were generated as a proof of concept rather than a product itself, and the purposed materials were designed to explore the potential uses of the main concepts initiated by Naor and Shami [1]. The aim of this investigation is to come up with as many elaborations of the concept as time allowed.

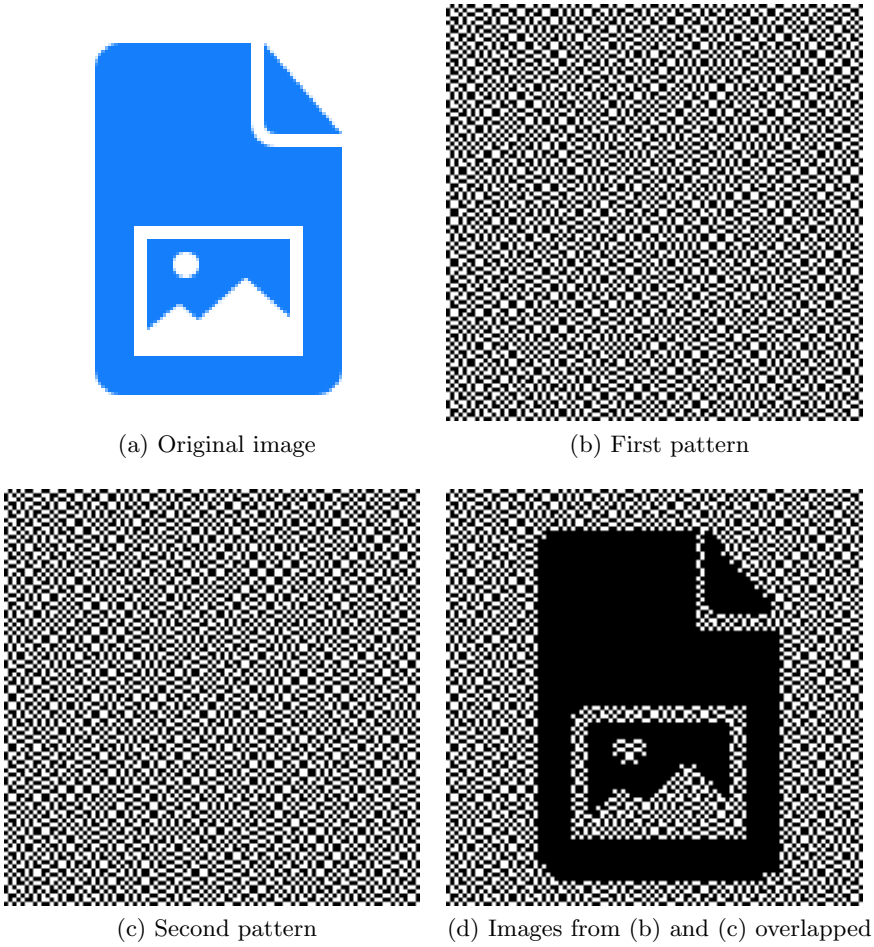
4 Visual Cryptography Algorithm

Two important algorithms are applied in this project. The first was developed by Naor and Shamir [1]. The second is a branched idea, initiated by Weir and Yan [8], constructed to deal with intersected objects that could be displayed independently with one single key share.

4.1 The Main Algorithm

The main algorithm consists in transforming an image into two ciphered shares that, when overlapped, can display the original foreground object. In order to accomplish this goal, the

original image needs to be gray-scale, and be binarized according to a suitable threshold, particular to each image, in a way that the resulting foreground can still be perceived through the black and white contrast. Then, the binary image is transformed into two final shares, perceived as noise by our visual system when seen individually. As an example of the results, we can see in Figure 1b, Figure 1c and the simulation of both transparent images overlapped in Figure 1d.

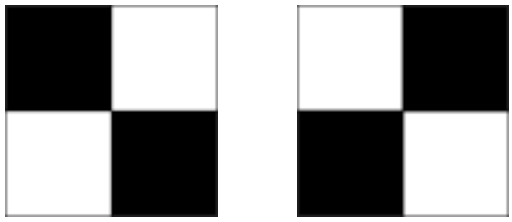


(a) Original image (b) First pattern

(c) Second pattern (d) Images from (b) and (c) overlapped

Figure 1: Example of the application of visual cryptography to cipher one source image into two complementary shares.

The final step performs the core aim of the whole algorithmic process. In order to give this explanation some context, let us call a square containing b^2 pixels a "virtual pixel", b being b an interger representing the size of one side of the square. For this report, the virtual pixels are constructed with $b = 2$, mostly, having their size increased programmatically when needed. Indeed, when one of the shares is lying over the other, the resulting shares need to be constructed in a way in which virtual pixels can be perceived as black among the randomly generated noise. In other words, the virtual black pixel must have all of its pixels



(a) Pattern 1 (b) Pattern 2

Figure 2: Patterns used to construct the shares. When different patterns are overlapped, all pixels become black.

– or the major part of its pixels – painted black. To achieve this goal, the patterns presented in Figure 2a and Figure 2b are used.

In brief, any black pixel from the binary image will result in one of shares receiving one pattern, while the other share receives the complementary pattern. Note that when the patterns are overlapped, the four pixels of the virtual pixel become like a logical "or" sum. On the other hand, any white pixel from the binary image will result in both ciphered shares being constructed with the same pattern, no matter which, such that half of the pixels will remain white when overlapped. However, in order to choose which share will acquire the first or second pattern, a "coin" is flipped. If the pixel is black and the coin ends up tails, the first share acquires the first pattern, while the second share acquires the second pattern. If the pixel is black and the coin ends up heads, then the opposite will occur. Otherwise, when the pixel is white, the coin will decide which pattern both shares will acquire. For example, if the flip ends up tails, both shares acquire the first pattern, and if heads, both acquire the second pattern.

The general pseudocode of the method is described in Listing 1. It produces two shares.

Figure 1 provides an example of one of the algorithm results. A playful part of this process emerges when the white pixels are transformed into transparent pixels after adding an alpha channel set to 0. The shares can then be printed on transparent paper. When these two shares are overlapped, the hidden image then emerges visually in print form.

Playfulness aside, the results are limited to creating only two shares for any given hidden image. With that constraint in mind and also thinking about possible user experiences and applications, we can approach the question: what if, with a single key share, one could uncover different images hidden in different shares? Is this even possible?

The answer is yes, and this process is explained in the next section.

4.2 Single-share-to-N algorithm

The goal of what we call "single-share-to-N" algorithm is to use a single key share to uncover images overlapping with N other shares. This has many possible uses. During the progression of the project, many ideas were suggested. For example, the following:

- Hide a whole text inside a ciphered paper and create a single key that can decipher all the letters inside it;

Listing 1: Main Algorithm

```
CIPHER_IMG(img):
    first_share <- Matrix with twice the size of img
    snd_share <- Matrix with twice the size of img
    for col in range(0, img.shape[1]):
        for row in range(0, img.shape[0]):
            coin = FLIP_COIN()
            if coin == HEAD:
                if img[row][col] == BLACK:
                    PAINT_PATTERN_1(first_share, row, col)
                    PAINT_PATTERN_2(snd_share, row, col)
                if img[row][col] == WHITE:
                    PAINT_PATTERN_1(first_share, row, col)
                    PAINT_PATTERN_1(snd_share, row, col)
            else:
                if img[row][col] == BLACK:
                    PAINT_PATTERN_2(first_share, row, col)
                    PAINT_PATTERN_1(snd_share, row, col)
                if img[row][col] == WHITE:
                    PAINT_PATTERN_2(first_share, row, col)
                    PAINT_PATTERN_2(snd_share, row, col)
    return first_share, snd_share
```

- Create hidden maps;
- Create puzzles;
- Different passwords for user access that could be revealed with a single key (private to each user);
- Reveal answers for questions in student's books;
- And many other applications on creative industries.

Given the number of possibilities suggested, this project could not include examples for each of them. Nevertheless, all of these ideas are open for potential development in future studies.

To focus on the algorithm itself, given a set I, such that $I = \{img_1, \ldots, img_k, \ldots, img_n\}$, where img_i is the i-th image, the algorithm will produce one key and n share, one for each image. The key share is created from any arbitrary image img_k from the set I.

With img_k determined, the img_k is processed by the main algorithm that generates the two shares. Let us name one of them "key" and the other "complement". The region where the foreground object occupies one key image will be represented by the set B. The region where the foreground object occupies the complement image will be called C. The region on the key image that occupies the same structural pattern and the same co-ordinate space on the complement image will be represented by the set A. Also, let us consider the

complementary set of A, A^C, as the set where all the patterns attributed to virtual pixels in the set A are inverted. For example, if one virtual pixel in A has pattern 1, the virtual pixel in A^C will have pattern 2, and vice-versa. The diagram displayed in Figure 3 illustrates the given definitions.

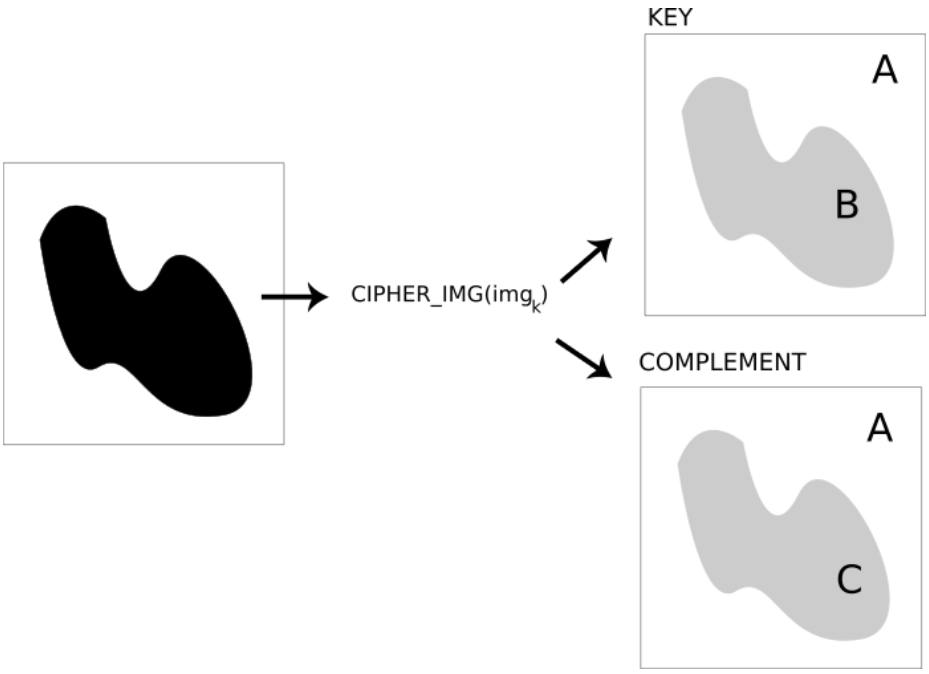


Figure 3: Set diagram for ciphered resultant shares after processing img_k into *CIPHER_IMG* function. The key image includes set A and B, and complement img has sets C and A, being C the set of inverted virtual pixels from B.

Given another arbitrary image img_p, such that $p \neq k$ and $p \in \{1, ..., n\}$, the ciphered share for img_p is constructed using the sets illustrated in Figure 3. The rules for the share construction are expressed as follows:

- The region where foreground of img_k intersects the foreground of image img_p will receive the virtual pixels from set C.
- The region where the foreground from the img_k is, but not the foreground of image img_p, will receive the virtual pixels from set B.
- The region where the foreground from image img_p is, but not the foreground of the img_k, will receive the virtual pixels from A^C.
- The region where both foreground objects are not located will receive virtual pixels from set A.

The rules are also illustrated in the diagram presented in Figure 4.

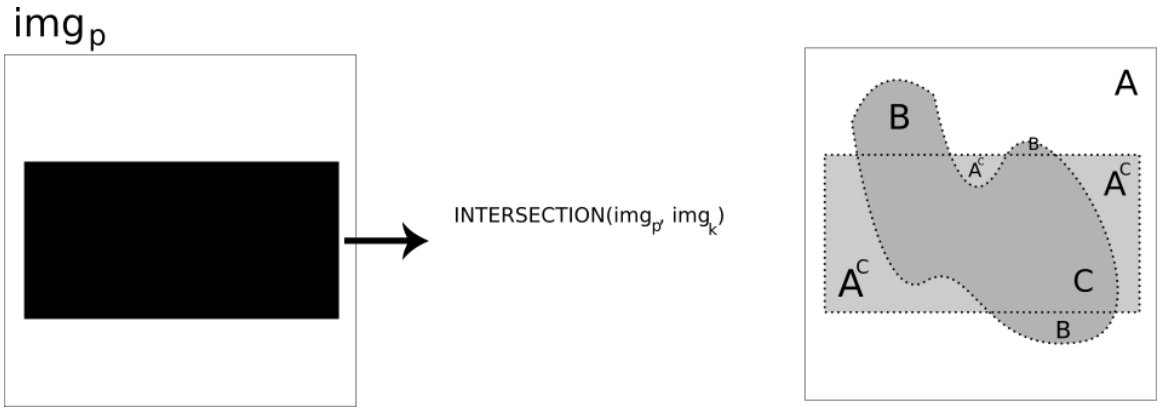


Figure 4: Diagram representing how regions of img_p's share are going to be defined according to the set definitions from the key share and its complement share.

When the shares are overlapped, if any region is overlapped by another that is constructed under the same set, the region will represent the white pixels. If the overlapped regions are constructed after complementary sets, they will represent the black pixels. In addition to this, any other image can be ciphered in the same way. So, given $n-1$ other images, we will have one single key ciphered share uncovering all the $n-1$ others. But before explaining the algorithm, let us define the functions *PAINT_PATTERN* and *PAINT_INVERTED_PATTERN*.

The first *PAINT_PATTERN* function is responsible for copying the pattern related to the virtual pixel created after processing the pixel at position (r,c) in the original image. The second *PAINT_INVERTED_PATTERN* function is similar, but instead of copying the same pattern, it acquires the complementary pattern. The latter is important for getting the virtual pixels from set A^C, the complementary set of A. The pseudocode is described in Listing 2.

This procedure is highly flexible, which makes it useful for many types of applications. As an example, in this project a set of 13 images are employed, with a "four-crossed arrows" image being the first determined, generating the key share and also its complement share. The rest of the procedure generates the other shares. A few shares and their overlapped results are described in Figure 5.

Note that the results are not entirely undecryptable. Since we base the other shares on the key share and its complement, partial decryption may occur by overlapping pairs from the set of other shares. Parts of the images related to each share may then become visible. Given another image, img_q, and its set diagram drawn in Figure 6, placing the img_q's share over the img_p's share will result in an image where the intersected foreground area is white, and the rest of the foreground is black.

Listing 2: Single-share-to-N Algorithm

```
PAINT_PATTERN( new_img, ciphered, r, c ):
    new_img[2*r][2*c] = ciphered[2*r][2*c]
    new_img[2*r][2*c+1] = ciphered[2*r][2*c+1]
    new_img[2*r+1][2*c] = ciphered[2*r+1][2*c]
    new_img[2*r+1][2*c+1] = ciphered[2*r+1][2*c+1]

PAINT_INVERTED_PATTERN( new_img, ciphered, r, c ):
    new_img[2*r][2*c] = 255 - ciphered[2*r][2*c]
    new_img[2*r][2*c+1] = 255 - ciphered[2*r][2*c+1]
    new_img[2*r+1][2*c] = 255 - ciphered[2*r+1][2*c]
    new_img[2*r+1][2*c+1] = 255 - ciphered[2*r+1][2*c+1]

INTERSECTION(img, source):
    new_img <- matrix twice the size of img
    key, complement <- CIPHER_IMG(source)
    for r in range(0, img.shape[0]):
        for c in range(0, img.shape[1]):
            if img[r][c] is BLACK and source[r][c] is BLACK:
                PAINT_PATTERN( new_img, complement, r, c ) // set C
            if img[r][c] is BLACK and source[r][c] is WHITE:
                PAINT_INVERTED_PATTERN( new_img, key, r, c ) // set A^C
            if img[r][c] is WHITE and source[r][c] is BLACK:
                PAINT_PATTERN( new_img, key, r, c ) // set B
            else:
                PAINT_PATTERN( new_img, key, r, c ) // set A

CREATE_INTERSECTIONS(IMGS, f):
    key, complement = CIPHER_IMG(f)
    for img in IMGS:
        img = INTERSECTION(img, key)
        save(img)
    end-for
return
```

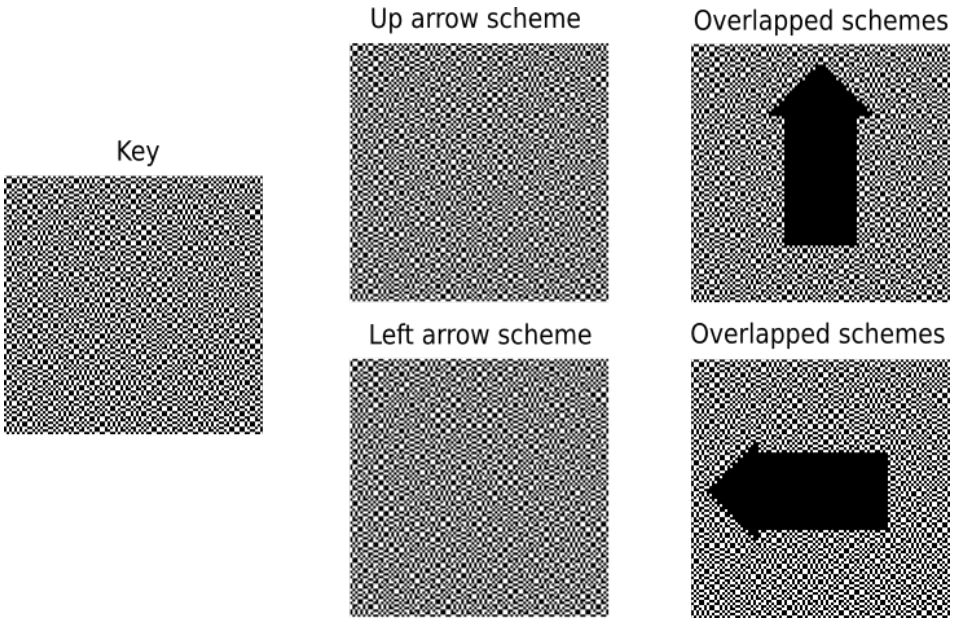


Figure 5: Diagram representing the results of placing the key share over the two others.

5 Results

This project also included the development of Python modules that could help us to achieve different uses of visual cryptography. As discussed, the libraries used are mostly OpenCV and Numpy, among other standard Python libraries such as *os* for file management. Each package is responsible for a specific part of the image processing, with each being important for the transformation pipelines. The scope of the packages are described as follows:

- Package Amplify: responsible for increasing the size of the pixels according to the parameter p.
- Package Cipher: responsible for generating two shares starting from an image as argument.
- Package Overlap: responsible for replacing one piece of image by other (and smaller) one.
- Package Printfy: responsible for adding alpha channel to all images and setting the alpha value of white pixels to 0.
- Package Resources: responsible for basic procedures, such as saving, loading, thresholding, turning transparent pixels to white, and defining Color labels used in the code.

In order to present a use case, we created a script called *construct_map.py*. The input for the script is a blank image that works as the background map, in which 13 types of

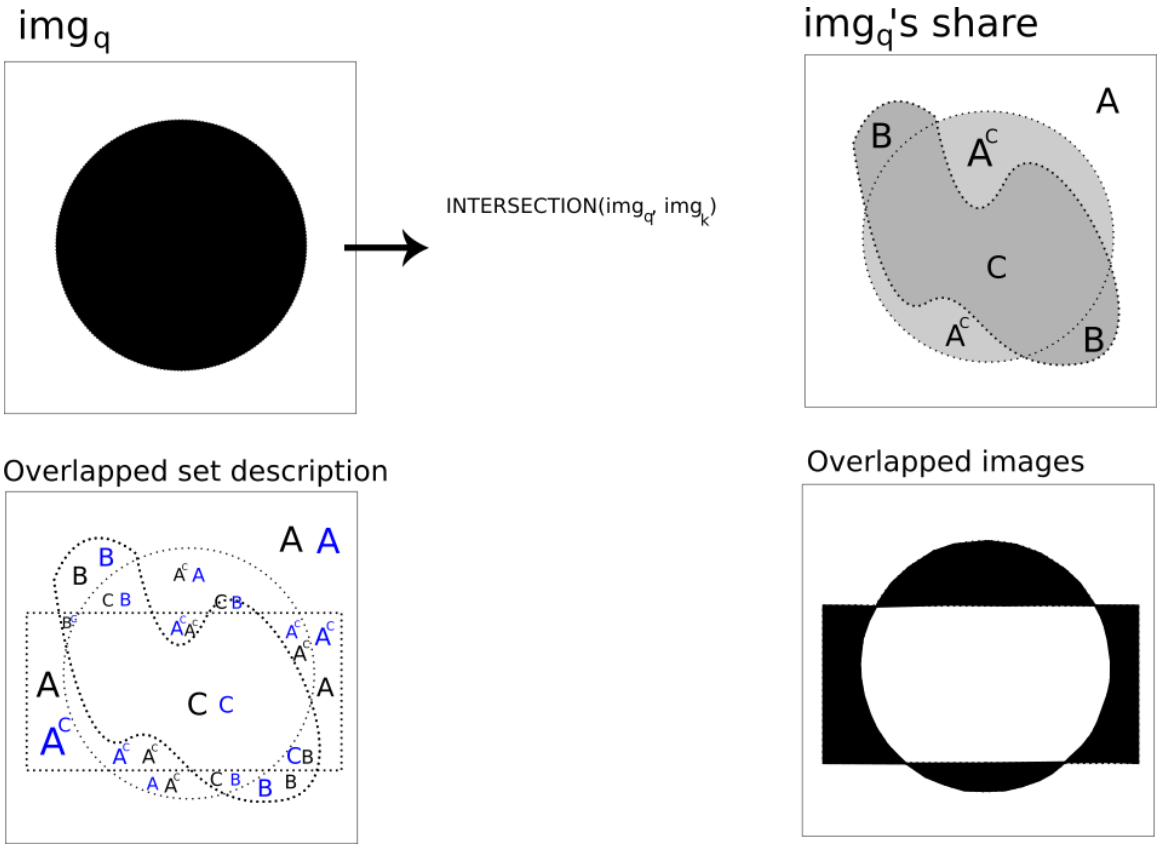


Figure 6: Set diagram for img_q, and overlapped set diagrams with img_p, and finally the resultant image after overlapping the two shares.

images are placed that express possible directions. In order to describe the type of image and where the image will be placed in the map, we use an input file called *map.in*.

The map file is constructed by specifying the size of the final map, as well as the coordinates to replace the map region with the shares. There are images that have two arrows pointing in different directions in order to create a labyrinth for the player. Inside the map file these arrows are represented using the following order: up, right, down, left.

The label for arrows are only the first letter of each word. When there are two directions, the letters have to respect the given order. For example, an image with an arrow pointing to the right and another pointing down are labeled as *rd* inside the map file.

The input template is expressed in Listing 3, where r_i and c_i integers, and $image\_label_i$ a string. To illustrate it, consider the map.in shown in Listing 4 as our map description.

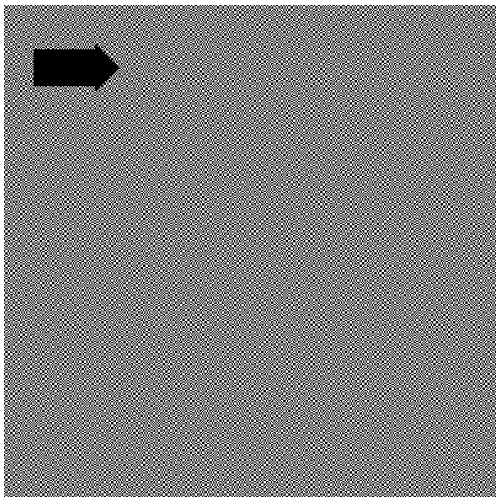
In order to use the generated map and the key, based upon the suggested approach, one might need to move the key share over the map to try to find the hidden tips until the four-arrowed image can be found. In Figure 7, the movement of the key frame is illustrated on a storyboard. However, the images in Figure 7 are re-sized, producing a loss of ciphered data through the mathematical transformation generated by LaTeX packages.

Listing 3: map.in

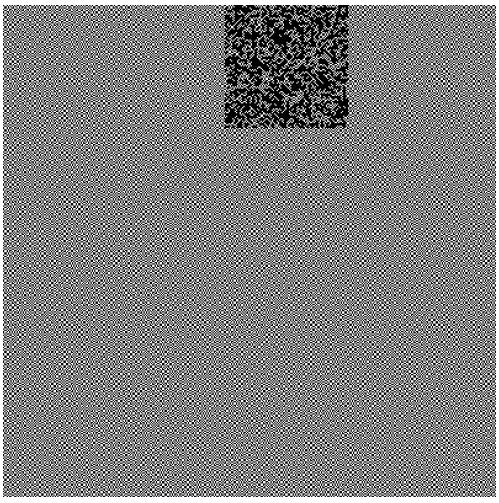
```
size_X size_Y
r_1 c_1 image_label_1
...
r_n c_n image_label_n
```

Listing 4: map.in

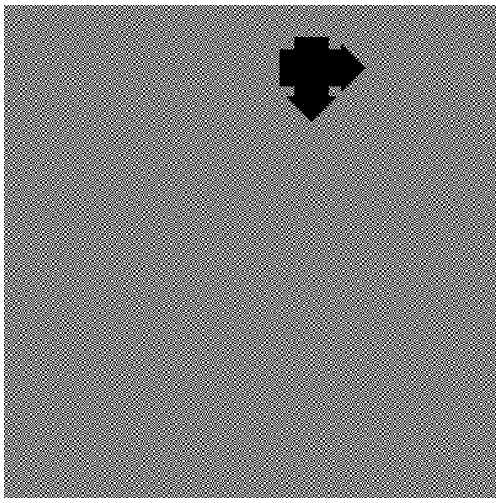
```
800 800
0 0 r
0 200 r
0 400 rd
0 600 d
200 0 d
200 200 dl
200 400 l
200 600 d
400 0 u
400 200 r
400 400 l
400 600 d
600 0 x
600 200 l
600 400 l
600 600 l
```



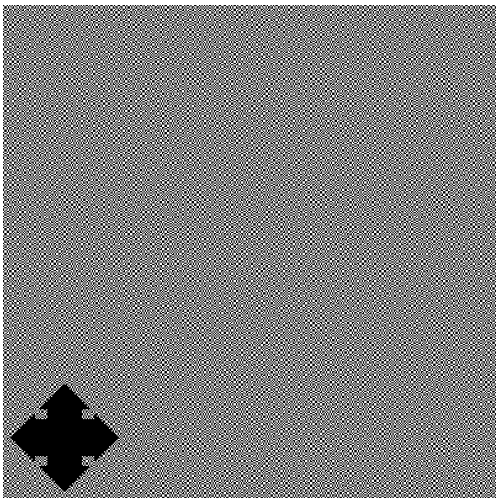
(a) Player have its key share over the right arrow share



(b) Player has its key over the map, but not overlapping any share



(c) The player has its key overlapping the share related to the double-directed arrows



(d) The player found the final destination

Figure 7: Story board illustrating the stages of a use. In (a), the player finds a first clue: an arrow pointing to the right. In (b), we see the noise generated by the key share when overlapped at a position in the map where there is no share. In (c), another type of arrow is displayed by the same key share. In (d), we see the stage where the player found the final destination

6 Conclusions

The uses of visual cryptography do not limit to only what was exposed in this work. There are many possibilities for future studies, such as exploring the movement liberties that one could have using a key share, and visualizing different hidden content. The aim of the

project was not just to display a map as a result of its progress, but also to expose the reader to new possibilities with visual cryptography. In order to do that, it was explored how the single-share-to-N algorithm was constructed and the concepts behind it - mostly provided by previous works from Nomi and Shami [1], and from Weir and Yan [8]-, pushing forward the mindset around the subject.

In that way, a proof of concept was constructed using open source tools, such as Python, OpenCV and Numpy. The result here exposed was a labyrinth, using the branched algorithm to construct the shares that would compose the whole map. The same procedure could also be used for any application that could need a key to uncover different contents.

The project and concepts it generated also raised a few questions about the new possibilities of working with visual cryptography: if we move the key through a straight line, could we get get an animation from it? Or, can we rotate the key share in 90 degrees in the same position and get another from the same overlapped cipher? Or even find good materials and procedures that could enhance visualization of the images in a physical paper? Basically, the goal of the suggested questions is to explicitly state that there is still much what to explore when it comes to visual cryptography.

References

[1] Naor, M., & Shamir, A. (1994, May). *Visual Cryptography* . In Workshop on the Theory and Application of of Cryptographic Techniques, pp. 1-12. Springer, Berlin, Heidelberg.

[2] Nakajima, M., & Yamaguchi, Y. (2002, January). *Extended Visual Cryptography for Natural Images*. Journal of WSCG 10(2), pp. 303-310.

[3] Hou, Y. C. (2003). *Visual Cryptography for Color Images*. Pattern Recognition, 36(7), pp. 1619-1629.

[4] Murthy, G. R. S., & Rao, D. D. (2012). *Visual Cryptography using Color Images*. International Journal of Electrical and Electronics Engineering, 2(1), pp.31-33.

[5] Marwaha, P., & Marwaha, P. (2010, July). *Visual Cryptographic Steganography in Images* . In International Conference on Computing Communication and Networking Technologies (pp. 1-6). IEEE.

[6] Cimato, S., & Yang, C. N. (Eds.). (2017, March). *Visual Cryptography and Secret Image Sharing*. CRC Press.

[7] Zhou, Z., Arce, G. R., & Di Crescenzo, G. (2006). *Halftone Visual Cryptography*. IEEE Transactions on Image Processing, 15(8), 2441-2453.

[8] Weir, J., & Yan, W. (2009, May). *Sharing Multiple Secrets using Visual Cryptography*. In IEEE International Symposium on Circuits and Systems, (pp. 509-512). IEEE.

[9] Blundo, C., D'Arco, P., De Santis, A., & Stinson, D. R. (2003). *Contrast Optimal Threshold Visual Cryptography Schemes*. SIAM Journal on Discrete Mathematics, 16(2), 224-261.