ECE 6960 Final Project Report: Implementations and Evaluations of Visual Cryptography Algorithms and Its Applications

Wan Lee, Haoze Zhang, and Cheng-Hsiang Chiu
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT-84112
wan.lee; haoze.zhang; cheng-hsiang.chiu@utah.edu

I. Introduction

Visual Cryptography (VC) is a technique to secure images and decrypt using human's eyes and is first introduced by Moni Naor and Adi Shamir [1], [2]. This type of cryptography combines cipher algorithms and secret sharing. There are many cipher algorithms used in VC (e.g., XOR, Halftone). Secret sharing is the idea of dividing the encrypted data or key into several pieces, namely *shares*. To decrypt the encrypted data, users must have all the shares. There are other secret sharing algorithms in which only k out of n shares are needed to decrypt the ciphertext, called (k, n) secret sharing. However, (k, n) is not in the scope of this project. We focus on (n, n) secret sharing.

VC can be applied in black-white, gray-scale, and full color images. We target color images in the project. To represent a color, people can either use Red-Green-Blue (RGB) or Cyan-Magenta-Yellow (CMYK) color model. Red, green, and blue are the primitive colors in the RGB model. Cyan, magenta, and yellow are the primitive colors in the CMYK model. Figure 1 shows the two color models. Researchers use CMYK model in VC the most because it makes more sense for human's eyes to see the black color when more colors are overlapped.

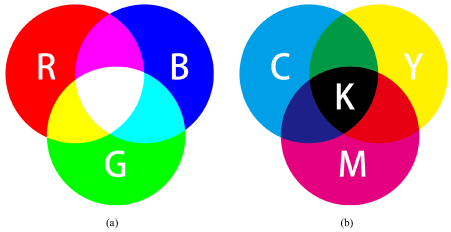

Fig. 1: Two color models. (a) represents the RGB model. (b) represents the CMYK model.

There are many papers talking about VC. However, there are fundamental algorithms which are the foundation blocks to these papers. The purpose of the final project is to implement three primary algorithms, XOR, Modular-Arithmetic, and Halftone [3], [4], and compared the performance using Mean Square Error (MSE) and Peak Signal-to-Noise Ratio (PSNR). Besides, we implemented two applications. One is to secure an image using Advanced Encryption Standard (AES) and VC [5]. The other is to generate meaningful image shares [4], [6].

The organization of this report is the followings. In Section II we talk about the three algorithms. In Section III we talk about the two implemented applications. Section IV presents the experimental results. Section V we discuss the labor division. Next, we show the lessons we learned from the project in Section VI. In Section VII the link to the github repository of our final project is available. Last, we conclude in Section VIII.

II. Algorithms

We implemented three algorithms based on the papers [3], [4]. Each algorithm is widely used in VC. Basically, many existing algorithms are based on the three algorithms.

A. Halftone

Halftone is a technique that simulates continue-tone image using dots. It aims to generate a gradient-like effect image. Interestingly, this technique can also be applied in visual cryptography. Halftone visual cryptography algorithm first does color decomposition to separate the original image into three halftone images which are cyan, magenta, and yellow halftone images. Each pixel in those three halftone images only contains two values which are either 255 or 0. For example, if a pixel at (x,y) position is 255 in the magenta halftone image, it means the magenta color presents at pixel (x,y). Otherwise the value is 0. After generating the three halftone images, we will use the following rule to expand each pixel to a 2x2 pixel block.

1) For each pixel of the cyan halftone images, if the value of the cyan component in the halftone images is 255 (the cyan component presents), then expand the original pixel to a 2x2 block. Fill the top-left and bottom-right pixel with cyan pixel and leave the other two pixels blank. If the value of the cyan component in the halftone images is 0 (the cyan component is absent), then expand the original pixel to a 2x2 block. Fill the top-right and bottom-left pixel with cyan pixel. Leave the other two pixels blank.
2) Repeat step 1 until every pixel of the halftone images are expanded, hence three shares (cyan, magenta, and yellow) are generated.
3) Generate a mask which has the same block numbers of the

three shares images. For each block. assignment black pixel to the top-left and bottom-right pixel, and leave the other two pixel blank

4) Finally, stack four shares, cyan, magenta, yellow, and black together to obtain the decrypted image. By stacking four shares together, we can generate 8 color combines show in figure 5.

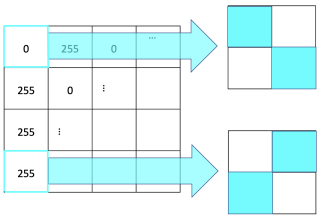

Fig. 2: The process of encrypting a cyan halftone image to generate a cyan share.

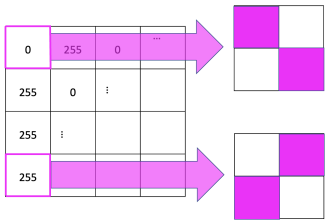

Fig. 3: The process of encrypting a magenta halftone image to generate a magenta share.

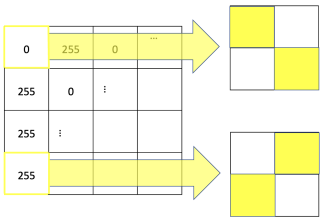

Fig. 4: The process of encrypting a yellow halftone image to generate a yellow share.

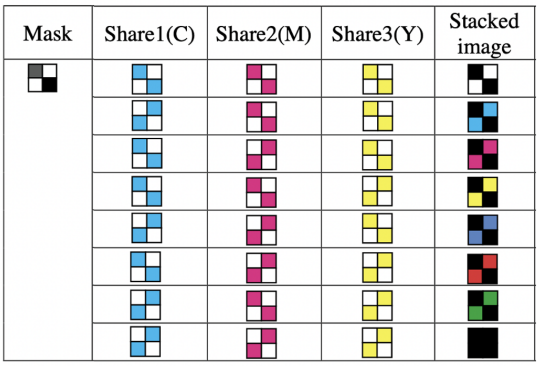

Fig. 5: Eight possible color combinations by stacking three shares, C, M, Y, and mask together.

B. XOR

XOR operation is one of the basic technology that used in Visual Cryptography. In our case, this algorithm can be explained as encrypting a number that will use XOR operation with a secret number.

A (k,n) visual cryptography scheme encrypts a secret image into n share images (printed on transparencies) [7]. But in our case, we use an (N,N) sharing scheme instead of (k,n), that means k = n. The process of the whole algorithm is that, every image will combine with sets of pixels and each pixel has pixel value, algorithm extend to make every pixel value in this image to work with XOR operation. Then original image will be XORed into n shares, we limit our 2 <shares <8. Because a large number of shares will not make any additional function to the XOR algorithm. Every shares is the same size that form into a secret image. This is basic idea of encrypt. Decryption, is the inverse process of encryption, the secret image is XORed with the n shares oaf images, the combine them together, return back to the original image. Since our size is (N,N), all shares image along with the secret image, decryption will be applied to every image. Even if one of the share image missed, it still requires $2^{(m*n)}$ states [7], m and n is the the size of original image, to retrieve the final image where each state is equiprobable, making it hard to decrypt.

Advantage of XOR algorithm, easily we can see it is easy built-in cipher system, and it is not required high cost for the computation system. Most important is this algorithm can be applied on image.

Disadvantage is obvious, that low cast with high risk, attacker can easily attach and steal original information from security system. Hence, this algorithm is not secure enough.

Here is the sudocode for XOR algorithm. Figure 6 and 7 are two XOR encryption shares, note that they are different, expansion is required.

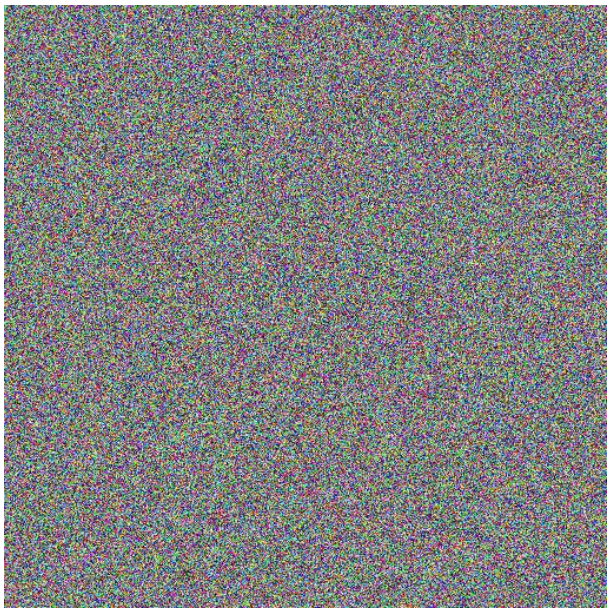

Fig. 6: Share 1 of XOR.

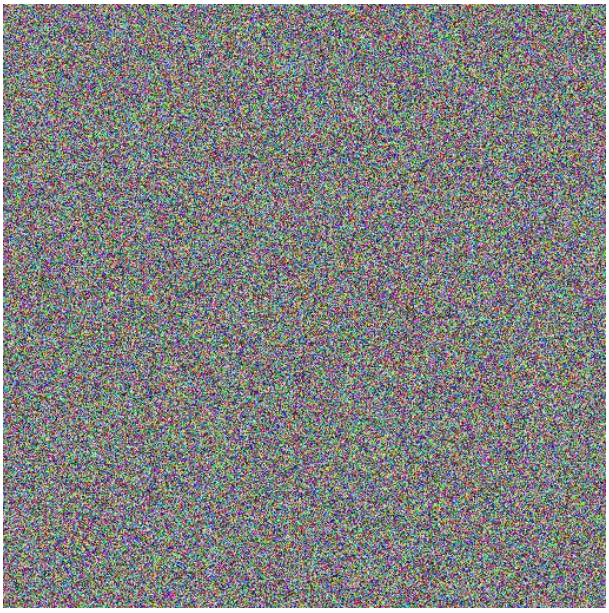

Fig. 7: Share 2 of XOR.

Input: input_image, share_size
Output: output_shares
$array = input\_imagetoarray$;
$(row, column, depth) = array.shape$;
choose $random.randint(0, 256, size)$;
$size = (row, column, depth, share\_size)$;
For each position (i,j) in the secret image,
n share pixels $R_1(i, j)$ to $R_n(i, j)$
Generate by the random number.
Construct every pixels r
$r_1 = Random()$;
$r_n = Random()$;
for *n=0; n<share_size; n++* **do**
| r_n = $r_1 \oplus r_2 \oplus$ .....$\oplus \mathrm{r}_{(n-1)}$
end
Save(shares)

C. Modular-Arithmetic

Modular-Arithmetic is different with XOR. This encryption technique uses the ideal of modular arithmetic and its cyclic ring nature. As previous Halftone mentions that each pixel is between 0-255. Based on this, this algorithm takes modulo 256. Similar to XOR, original image is added to n share images, same that 2 < n <8, of the same size taken to modulo 256 so that we can get a fianl secret image by forming share images together. This is the process of encryption. To decrypt, we apply some operations to the secret image, that each n shares image, (secret image − n share images (mod 256)) to get the final decrypted image. Same as XOR, an (N, N) sharing scheme, mean that all n share images along with the secret image is required to decrypt the image.

Advantage of modular arithmetic algorithm, also easy built-in encryption and decryption system and significantly low cost. The disadvantage is that attacker can be easily track the secret images and steal original image.

Input: input_image, share_size
Output: output_shares
$array = input\_imagetoarray$;
$(row, column, depth) = array.shape$;
choose $random.randint(0, 256, size)$;
$size = (row, column, depth, share\_size)$;
For each position (i,j) in the secret image,
n share pixels $R_1(i, j)$ to $R_n(i, j)$
Generate by the random number.
Construct every pixels r
$r_1 = Random()$;
$r_n = Random()$;
for *n=0; n<share_size; n++* **do**
| r_n = ($r_1 + r_2 +$ ..... $+ r_{(n-1)}$) mod 256
end
Save(shares)

Here are two shares of Modular arithmetic algorithm, note that they are different image. Expanding to see the different.

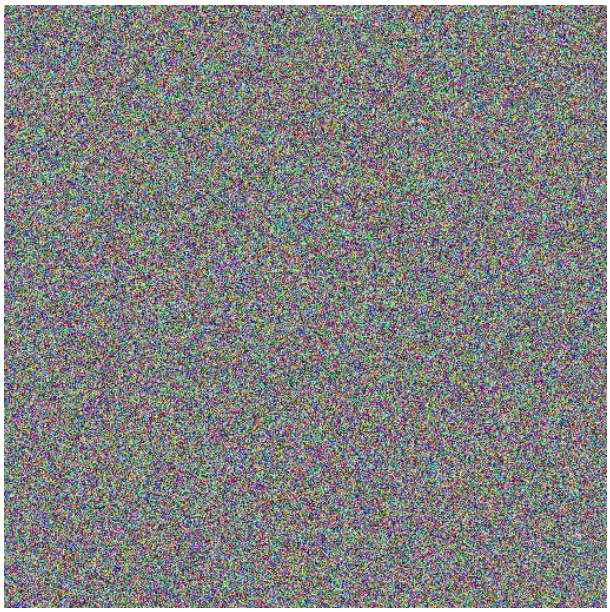

Fig. 8: Share 1 of Modular.

III. Applications

We implemented two applications. One is to secure an image using AES and VC. The other is to generate meaningful shares.

A. Application I - AES AND VC

This algorithm intends to secure a key used in AES by converting the key into an image and splitting it into two shares using visual secret sharing techniques [5]. The algorithm includes encryption phase and decryption phase. Figure 10 shows the encryption. There are two inputs. One is the image `I` to be encrypted and the other is the key `K`. `SK` is the hashed value of `K` using `SHA256`. `BI` is the Base64 string of `I`. Next, we pass `SK` and `BI` to `AES256` module and get the encrypted

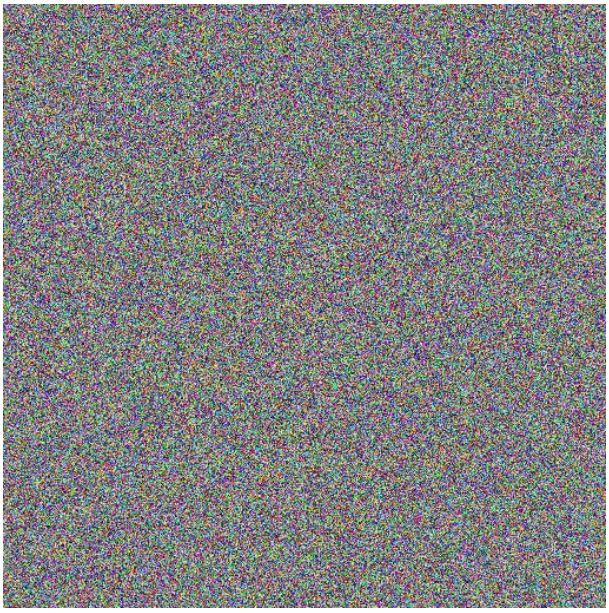

Fig. 9: Share 2 of Modular.

image CI. Then, we convert K to KI and split it to share P and R.

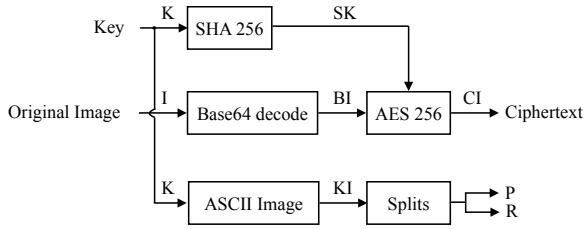

Fig. 10: The process of encryption. Two shares P and R are generated.

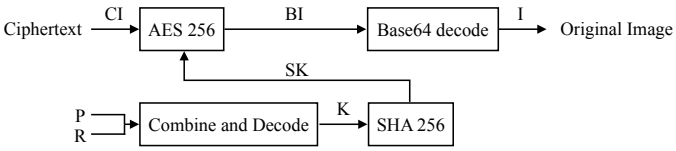

Fig. 11: The process of decryption.

Figure 11 shows the decryption. There are three inputs needed for the decryption. To decrypt the ciphertext CI, we need the *key*, SK, which is a hashed value of the original key. To get SK, we need to combine share P and R, decode the combined data and pass the decoded data to SHA256 to get the hashed value SK. After getting SK, we can decrypt CI with SK using the decryption module of AES256. Next, we decode BI and get the original image back.

Algorithm 1 shows the pseudo code of encryption. There are two inputs, K (key) and I (original image). We obtain SK using sha256 on K, get BI using Base64 on I, and get CI using the Encrypt function defined in class AESCipher. Next, we generate a C image and use that to generate share R and P.

Algorithm 1 Encryption(K, I)

Input: K, I
Output: CI, P, R
SK = sha256(K)
BI = base64.b64encode(I)
CI = AESCipher(BI, SK).Encrypt()
w = 255
h = length(K)
for *i=0; i < h; ++i* **do**
  j = ASCII(K[i])
  for *k=0;k < 255; ++k* **do**
    if $k < h$ **then**
      C[i][k][0] = 0
    end
    else
      C[i][k][0] = 1
    end
  end
end
for *i = 0; i < h; ++i* **do**
  for *j = 0; j < 255; ++j* **do**
    R[i][j][0] = random_generator(0,1)
  end
end
for *i=0;i<h;++i* **do**
  for *j=0;j<255;++j* **do**
    P[i][j][0] = R[i][j][0] $\wedge$ C[i][j][0]
  end
end

Algorithm 2 shows the pseudo code of decryption. There are three inputs, CI (ciphertext), P (share), and R (share). We obtain CK back by performing xor operation on P and R. Next, we get the K (key) back by calculating the number of zeros in CK and use the key to get its hashed value SK. Then, we pass CI and SK to the Decrypt function defined in class AESCipher to decrypt the image back.

In Algorithm 1 and Algorithm 2, we see a Encrypt and Decrypt function of class AESCipher. We use Python's *Crypto.Cipher* module to define our own AESCipher. Hence, we did not implement AES but used a Python module to simplify the design of our class AESCipher.

B. Application II - Meaningful Shares

The shares generated in the three algorithms introduced above are like *random noises* to human eyes. That is, the shares do not present meaningful images. In the application, we implemented the algorithm to encrypt a secret image and used two cover images together to generate two meaningful shares [6], [4]. To get the secret image back we simply overlay the two shares.

The algorithm includes five steps. Figure 12 shows the whole process. The first step is to perform the CMYK Decomposition on the two cover images CI1 and CI2 and the secret image SI. All of the images are of dimension $N*M$. The second step is to covert the images decomposed in the CMYK channel to their halftone. The detailed descriptions

Algorithm 2 Decryption(CI, P, R)

Input: CI, P, R
Output: I
for *i=0;i<P.width;++i* **do**
  for *j=0;j<P.height;++j* **do**
    CK[i][j][0] = P[i][j][0] $\wedge$ R[i][j][0]
  end
end
for *i=0;i< length(CK);++i* **do**
  count = 0
  for *j =0;j<length(CK[i]);++j* **do**
    if $CK[i][j][0] = 0$ **then**
      count = count + 1
    end
  end
  K[i] = ASCII(count)
end
SK = sha256(K)
for *c in CI* **do**
  ch = ASCII(c)
  txt.append(int(ch))
end
for *t in txt* **do**
  text += ASCII(t)
end
I = AESCipher(text, SK).Decrypt()

of `CMYK Decomposition` and `Halftone Conversion` are given in Section II-A.

The third step is to extract half of the pixels from the halftone images. After the extraction, the dimension of an image is reduced from $N * M$ to $\frac{N*M}{2}$. The purpose is to reduce the dimension of the halftone images such that the dimension of shares does not expand over 2 times bigger. We can either skip the even rows and keep the odd rows or vice versa.

Next, in the fourth step we encode the cover and secret images using `Cover Coding Table` (CCT) or `Secret Coding Table` (SCT), respectively.

Figure 13 shows the encoding table for `CI1_HE` and `CI2_HE`. According to the pixel at the same location, we encode `CI1_HE` and `CI2_HE` and generate two cover image patters. For example, the color of `CI1_HE` at location (i, j) = (100, 150) is blue and the pixel of `CI2_HE` at the same location is yellow. We refer to the fifth column, color blue, and fourth row, color yellow, and get two $2 * 2$ cover image patterns in Figure 13. To encode `SI_HE` we refer to SCT, shown in Figure 14. Based on the color, we generate two $2*2$ secret image patterns for the pixel of `SI_HE` at a specific location. For example, at location (i, j) = (100, 150) the color is red, we go to the sixth column and get two patterns.

After obtaining two $2 * 2$ cover image patterns and two $2 * 2$ secret image patterns, we can generate two shares in the final step. Figure 15 shows the mechanism. For a pixel at *odd* row in `Share 1`, we use a block which consists of cover image pattern one followed by secret image pattern one; in `Share 2`, the block consists of cover image pattern two followed by secrete image pattern two. For a pixel at *even* row

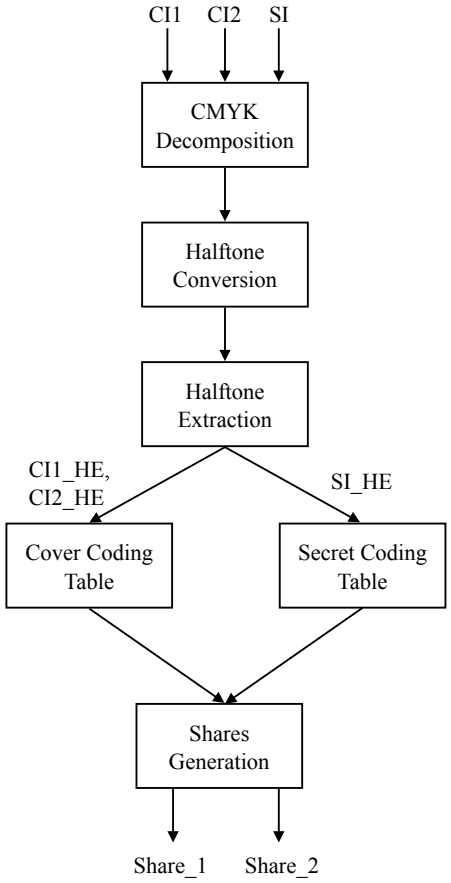

Fig. 12: The process of the algorithm [6], [4]. `CI1` and `CI2` refer to the cover image 1 and 2, respectively. `SI` denotes the secret image.

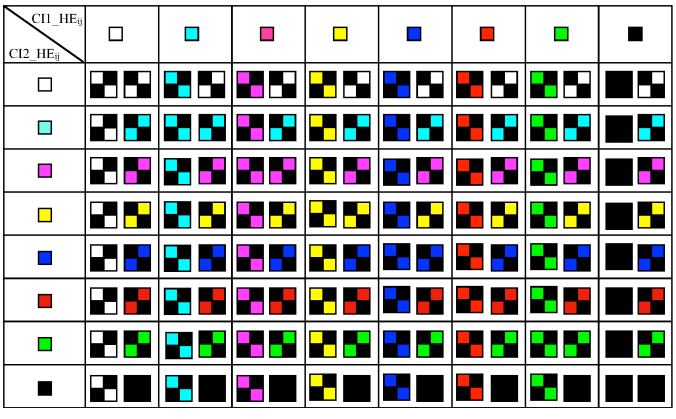

Fig. 13: The Cover Coding Table. The column denotes the color of `CI1_HE` and the row denotes the color of `CI2_HE`.

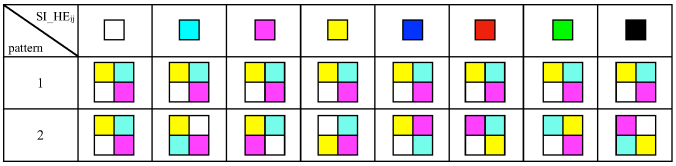

Fig. 14: The Secret Coding Table.

in `Share 1`, secret image pattern one is followed by cover

image pattern one followed by secret image pattern one in a block; in `Share 2`, secrete image pattern two comes before cover image pattern two. For example, for a pixel at (1, 100) `CI1_HE` is blue, `CI2_HE` is yellow and `SI_HE` is red, we generate the pixel at (1, 100) for `Share 1` using the block shown in Figure 15(a). The same rule applies to `Share 2`. If the pixel at (2, 100), we use the blocks shown in Figure 15(b) to generate the pixel for `Share 1` and `Share 2`. Since we use a block with dimension of $4 * 2$ for a pixel, the dimension of the two shares is $2N * 2M$.

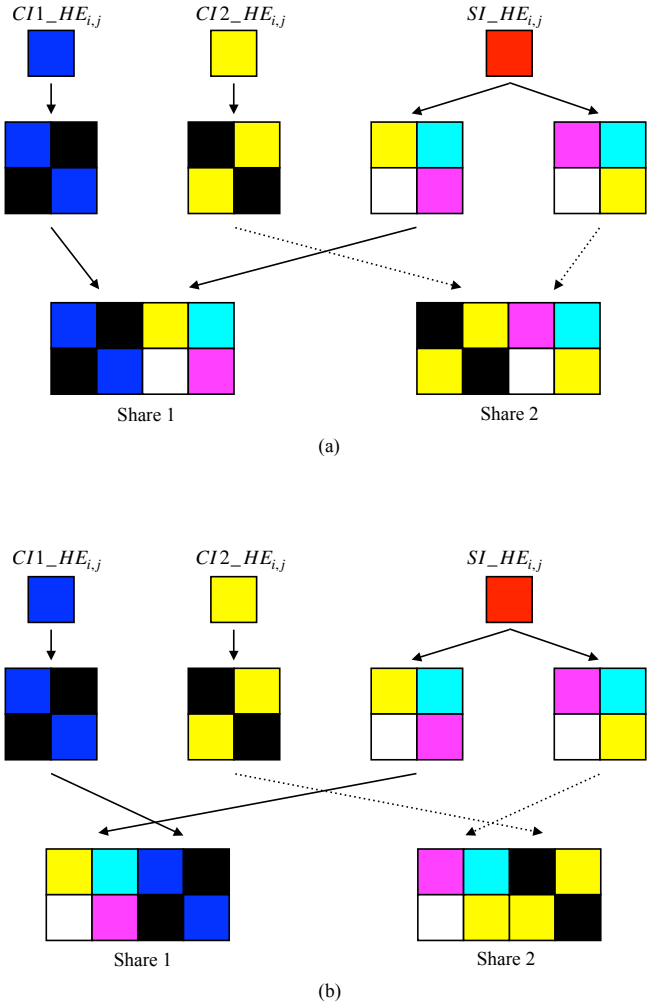

Fig. 15: Demonstration of generating two shares. (a) refers to the mechanism of odd row. (b) refers to the even row.

Algorithm 3 shows the pseudo code of the first step `CMYK Decomposition`. We decomposed the CMYK channel of an image by creating three temporary array. Each one of them contains only one channel. Then we save the three temporary array.

Algorithm 4 shows the pseudo code of the second step `Halftone Conversion`. The inputs are the CMYK decomposition of an image. We convert the CMYK channels of image to binary and save the halftone.

Algorithm 5 shows the third step `Halftone Extraction`. We extracted a half of the halftone image.

Algorithm 3 CMYK(image)

Input: $image$
Output: $image\_c, image\_m, image\_y$
for *x=0;x<image.width;++x* **do**
  for *y=0;y<image.height;++y* **do**
    pixel = image[x][y]
    image_c[x][y] = tuple(pixel[0], 0, 0, 0)
    image_m[x][y] = tuple(0, pixel[1], 0, 0)
    image_y[x][y] = tuple(0, 0, pixel[2], 0)
  end
end
Save(image_c, image_m, image_y)

Algorithm 4 Halftone(image_c, image_m, image_y)

Input: $image\_c, image\_m, image\_y$
Output: $image\_halftone$
for *x=0;x<image_c.width;++x* **do**
  for *y=0;y<image_c.height;++y* **do**
    pixel_c = binary(image_c[x][y])
    pixel_m = binary(image_m[x][y])
    pixel_y = binary(image_y[x][y])
    image_halftone[x][y] = tuple(pixel_c, pixel_m, pixel_y, 0)
  end
end
Save(image_halftone)

Since a pixel can be one of the eight colors, Algorithm 6 only demonstrates the pseudo code of CCT when the pixel is blue, as shown in Figure 13. There are two halftone extractions of cover images. For every pixel in the two halftone extractions, we generated a $2 * 2$ pattern. Pattern one is for halftone extraction of cover image one and pattern two is for halftone extraction of cover image two. There are eight colors we used in CCT. The encoding of each color for CMYK model

Algorithm 5 Extraction(image_halftone)

Input: $image\_halftone$
Output: $image\_halftone\_extraction$
for *x=0;x<image_halftone.width;++x* **do**
  for *y=0;y<image_halftone.height;++y* **do**
    if *x%2 = 0* **then**
      image_halftone_extraction[x/2][y] = image_halftone[x][y]
    end
  end
end
Save(image_halftone_extraction)

is described in the following,

$$color\ white : tuple(0,0,0,0)$$
$$color\ cyan : tuple(255,0,0,0)$$
$$color\ magenta : tuple(0,255,0,0)$$
$$color\ yellow : tuple(0,0,255,0)$$
$$color\ blue : tuple(255,255,0,0)$$
$$color\ red : tuple(0,255,255,0)$$
$$color\ green : tuple(255,0,255,0)$$
$$color\ black : tuple(255,255,255,0)$$

Algorithm 6 CCT(pixel=blue, pattern)

Input: $pixel, pattern$
Output: cct
if *pixel = blue* **then**
  if *pattern = 1* **then**
    cct[0][0] = tuple(255,255,0,0)
    cct[0][1] = tuple(255,255,255,0)
    cct[1][0] = tuple(255,255,255,0)
    cct[1][1] = tuple(255,255,0,0)
  end
  else
    cct[0][0] = tuple(255,255,255,0)
    cct[0][1] = tuple(255,255,0,0)
    cct[1][0] = tuple(255,255,0,0)
    cct[1][1] = tuple(255,255,255,0)
  end
end

Algorithm 7 shows the pseudo code of implementing SCT for color red. For a pixel in the halftone extraction of the secret image, we generated two patterns `sct1` and `sct2`. Each one is of dimension $2 * 2$.

Algorithm 7 SCT(pixel=red)

Input: $pixel$
Output: $sct1, sct2$
if *pixel = red* **then**
  sct1[0][0] = tuple(0, 0, 255, 0)
  sct1[0][1] = tuple(255, 0, 0, 0)
  sct1[1][0] = tuple(0,0,0,0)
  sct1[1][1] = tuple(0, 255, 0, 0)
  sct2[0][0] = tuple(0, 255, 0, 0)
  sct2[0][1] = tuple(255, 0, 0, 0)
  sct2[1][0] = tuple(0,0,0,0)
  sct2[1][1] = tuple(0, 0, 255, 0)
end

Algorithm 8 shows the final step of generating two shares `share_1` and `share_2`. We used function `CCT` and `SCT` to obtain `cct_1`, `cct_2`, `sct_1`, and `sct_2`. Then, we generated `share_1` and `share_2` based on the placement illustrated in Figure 15.

What we just described above is the implementation of the algorithm proposed in the paper [6], [4]. The algorithm relies on the extraction module to reduce the dimension of the halftone images such that the dimension of the two shares is $2N * 2M$. As we will see in the experiment, the decrypted image is merely recognizable and not very clear. Hence, we propose another method in which no extraction step is needed and a new way to generate two shares.

Algorithm 8 Generate_Shares(image_halftone_extractions)

Input: $image\_halftone\_extractions$
Output: $share\_1, share\_2$
for *x=0;x<image_halftone_extraction_1.width;++x* **do**
  for *y=0;y<image_halftone_extraction_1.height;++y* **do**
    cct_1 = CCT(image_halftone_extraction_1[x][y],1)
    cct_2 = CCT(image_halftone_extraction_2[x][y],2)
    sct_1, sct_2 = SCT(image_halftone_extraction_s[x][y])

    if *x%2 = 1* **then**
      share_1[4*x][2*y] = cct_1[0][0]
      share_1[4*x][2*y+1] = cct_1[0][1]
      share_1[4*x+1][2*y] = cct_1[1][0]
      share_1[4*x+1][2*y+1] = cct_1[1][1]
      share_1[4*x+2][2*y] = sct_1[0][0]
      share_1[4*x+2][2*y+1] = sct_1[0][1]
      share_1[4*x+3][2*y] = sct_1[1][0]
      share_1[4*x+3][2*y+1] = sct_1[1][1]

      share_2[4*x][2*y] = cct_2[0][0]
      share_2[4*x][2*y+1] = cct_2[0][1]
      share_2[4*x+1][2*y] = cct_2[1][0]
      share_2[4*x+1][2*y+1] = cct_2[1][1]
      share_2[4*x+2][2*y] = sct_2[0][0]
      share_2[4*x+2][2*y+1] = sct_2[0][1]
      share_2[4*x+3][2*y] = sct_2[1][0]
      share_2[4*x+3][2*y+1] = sct_2[1][1]
    end
    else
      share_1[4*x][2*y] = sct_1[0][0]
      share_1[4*x][2*y+1] = sct_1[0][1]
      share_1[4*x+1][2*y] = sct_1[1][0]
      share_1[4*x+1][2*y+1] = sct_1[1][1]
      share_1[4*x+2][2*y] = cct_1[0][0]
      share_1[4*x+2][2*y+1] = cct_1[0][1]
      share_1[4*x+3][2*y] = cct_1[1][0]
      share_1[4*x+3][2*y+1] = cct_1[1][1]

      share_2[4*x][2*y] = sct_2[0][0]
      share_2[4*x][2*y+1] = sct_2[0][1]
      share_2[4*x+1][2*y] = sct_2[1][0]
      share_2[4*x+1][2*y+1] = sct_2[1][1]
      share_2[4*x+2][2*y] = cct_2[0][0]
      share_2[4*x+2][2*y+1] = cct_2[0][1]
      share_2[4*x+3][2*y] = cct_2[1][0]
      share_2[4*x+3][2*y+1] = cct_2[1][1]
    end
  end
end
Save(share1, share2)

Our proposed process is similar to Figure 12 except no `Halftone Extraction`. We use the same CCT and SCT. In the last step, we use a different mechanism. Figure 16 shows the mechanism. We duplicate the cover image pattern and put the pattern in the diagonal position. Same rule applies to the secret image pattern as well. Instead of generating a block of dimension $4 * 2$, we generate a block of dimension $4 * 4$. Therefore, the dimension of the two shares is $4N * 4M$.

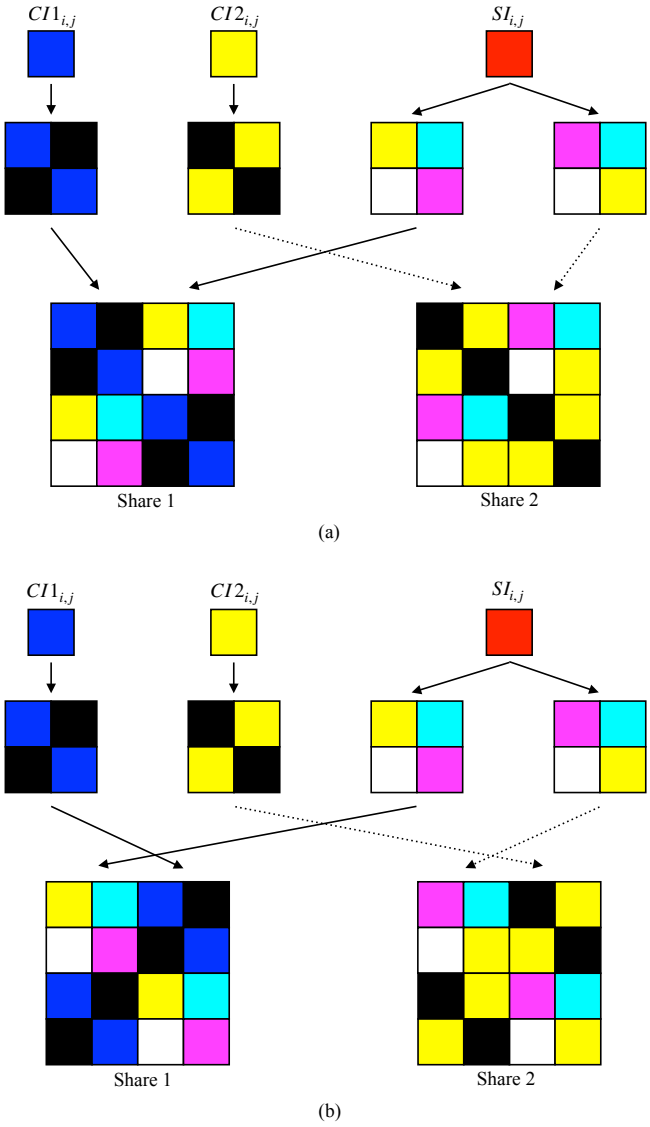

Fig. 16: Demonstration of generating two shares with our proposed method. (a) refers to the mechanism of odd row. (b) refers to the even row.

IV. Experimental Results

We implemented all algorithms and applications with Python 3. We also evaluated the performance across Meas Square Error (MSE) and Peak Signal-to-Noise Ratio (PSNR).

A. Performance Metrics

MSE is an estimator measuring the average of the squares of the errors. The higher the MSE, the worse the performance is. MSE of the original image and the decrypted image is defined in the following,

$$MSE = \tag{1}$$

$$\frac{1}{N * M} \sum_{i,j=0,0}^{N,M} (pixel_{i,j}^{original} - pixel_{i,j}^{decrypted})^2 \tag{2}$$

where the dimension of both images is $N * M$, $pixel_{i,j}^{original}$ refers to the pixel of the original image at position (i, j), and $pixel_{i,j}^{decrypted}$ refers to the pixel of the decrypted image at position (i, j).

PSNR is a ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation. The higher the PSNR, the better the performance. PSNR is defined in the following,

$$PSNR = \tag{3}$$

$$20 * \log_{10}(\frac{255.0}{\sqrt{MSE}}) \tag{4}$$

where 255.0 is the maximum value a pixel can have and MSE is measured in Equation 2

B. Comparisons Between Algorithms

TABLE I: Performance comparisons among three algorithms.

Algorithm	MSE	PSNR
XOR	9.46	38.37
Modular-Arithmetic	9.46	38.37
Halftone	108.59	27.77

Table I shows the MSE and PSNR comparions among the three algorithms. We can see that Halftone has the highest MSE and the lowest PSNR, which means Halftone performs the worst. The reason is that Halftone rounds the pixel value to either 255 or 0. That is, we lose some information during the process. The lost information directly leads to a high MSE and low PSNR. XOR and Modular algorithms do not round up or down any pixel.

C. Visualizations of the Encryption and Decryption

Figure 17 shows the decryption process of Halftone algorithm, shown in Section II-A. We stacked four shares, C share, M share, Y share, and mask share, and got the decrypted image back. Although the decrypted image is not totally identical to the original image, we can still recognize the original image without big problem.

D. The Applications

We implemented two applications. The first one is the application of AES and VC. The second one is the application of generating meaningful shares.

For the first application, we used a string ”this is ece 6960 hardware cryptography” as the key and Figure 18 is the image we encrypted. Figure 19 and 20 show the two shares. Figure 21 shows the decrypted image. We can not see any big difference

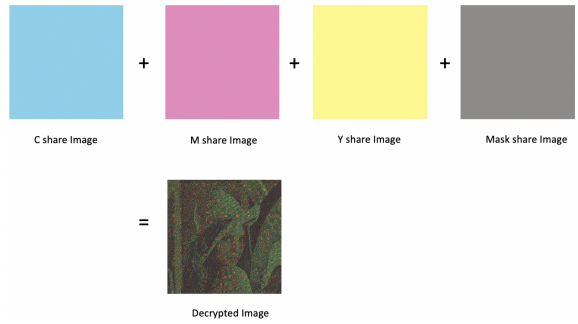

Fig. 17: The process of halftone visual decryption. Stacking the fours shares, `Cyan share`, `Magenta share`, `Yellow share`, and `Mask share` together to generate the decrypted image.

Fig. 18: The image we wanted to encrypt in application one.

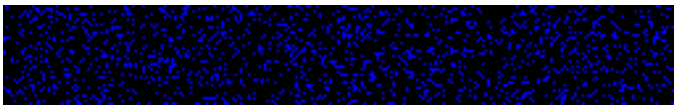

Fig. 19: R share.

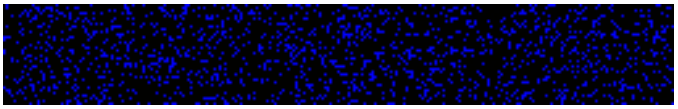

Fig. 20: P share.

Fig. 21: The decrypted image.

between the decrypted image and the original image. The MSE of this algorithm is 0 and PSNR is 100.

For the second application, Figure 22 and Figure 23 show the cover image one and two, respectively. Figure 24 shows the secret image. Figure 25 and Figure 26 show the two meaningful shares. Figure 27 shows the decrypted images while overlaying the two shares.

Fig. 22: Cover image one.

Fig. 23: Cover image two.

Fig. 24: Secret image.

We can summarize the second application with the following procedure, shown in Figure 28.

For our proposed method, `Share one`, `Share two`, and the decrypted secret images are shown in Figure 29, Figure 30, and Figure 31, respectively. We can easily see that the

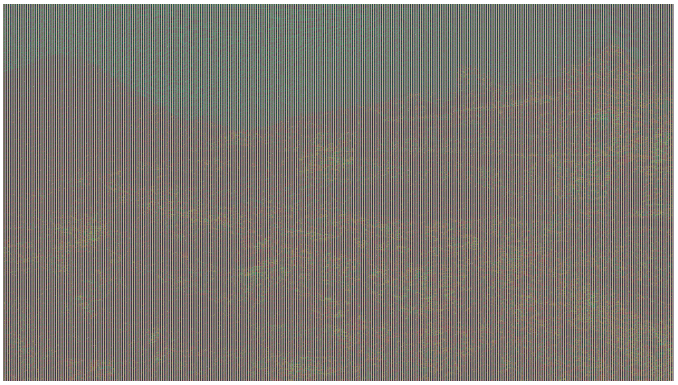

Fig. 25: Share one.

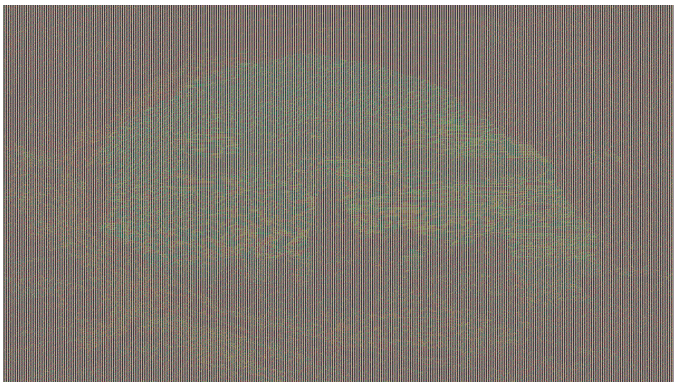

Fig. 26: Share two.

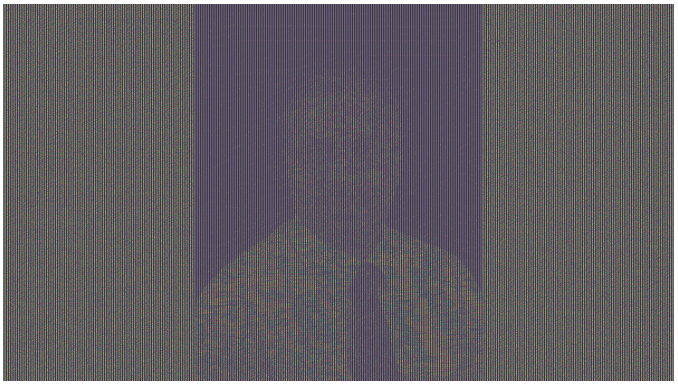

Fig. 27: The decrypted secret image.

decrypted secret image is more recognizable and more clear using our method than Figure 27. Since our method does not perform extraction, we do not lose much information and can get a more clear decrypted image. The disadvantage of our method is that the dimension is $2 * 2 = 4$ times bigger than the paper.

V. Labor Division

The final project includes three algorithms and two applications. The division is described in the following:

1) Wan Lee implemented Halftone algorithm, explained in Section II-A.
2) Haoze Zhang implemented XOR and Modular algorithms, explained in Section II-B and II-C.

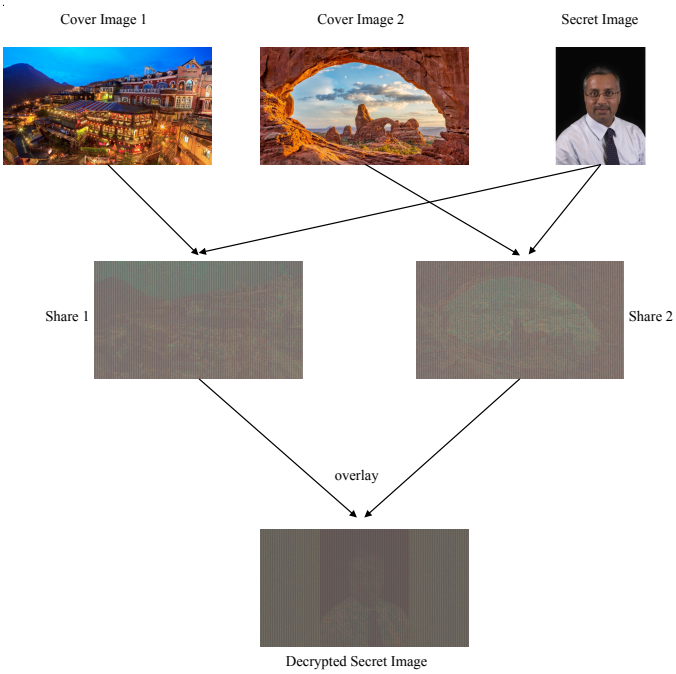

Fig. 28: The encryption and decryption of secret image.

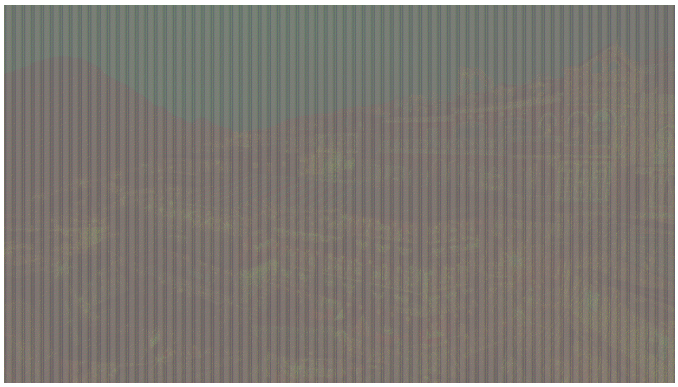

Fig. 29: Share one using our proposed method.

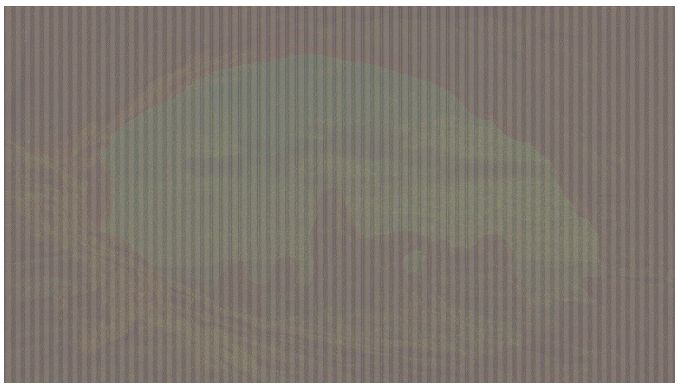

Fig. 30: Share two using our proposed method.

3) Cheng-Hsiang Chiu implemented application AES, explained in Section III-A.
4) Wan Lee, Haoze Zhang, and Cheng-Hsiang Chiu together implemented the application, meaningful shares, explained in Section III-B.

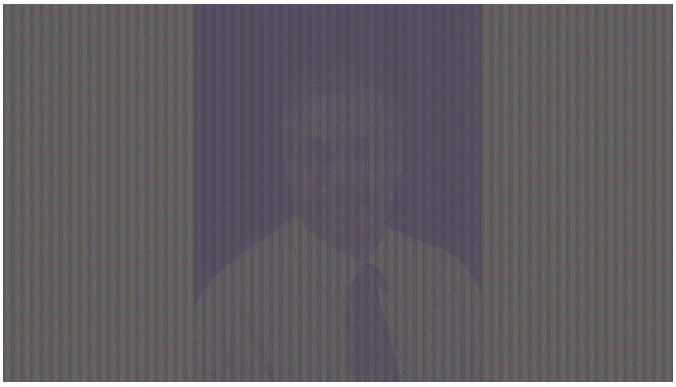

Fig. 31: The decrypted secret image using our proposed method.

VI. What We Learned

The first lesson we learned is the concept of *secret sharing*. The concept lets us to divide the "secret" into several shares and only to get the "secret" back with all shares (e.g., (n, n) algorithm) or some of the shares (e.g., (k, n) algorithm). Secret sharing is useful in cryptography. For example, some algorithms introduce *Key sharing* to create several key shares instead of one private key.

The second lesson is halftone algorithm. The algorithm decomposes an image to C image, M image, and Y image. This algorithm is useful when we targeted color images. When combing halftone algorithm and secret sharing concept, we can develop visual cryptography for color images.

The third lesson is the color encoding and the placement of the encoding, which can hide the secret image in the cover images and allow us to generate meaningful shares.

VII. Availability

The source code is available at https://github.com/cheng-hsiang-chiu/ECE6960-Final-Project

VIII. Conclusion

We have implemented three algorithms and evaluated the performance across MSE and PSNR. We also implemented two applications. One of the application is to secure an image using AES and VC. The MSE and PSNR of this application shows the image is perfectly decrypted. The second application is to generate meaningful shares using a published algorithm and our own method. Our own method has a more clear decrypted image than the paper visually. However, our method leads to a 4 times bigger in dimension.

References

[1] A. S. Moni Naor, "Visual Cryptography," in *Advances in Cryptology - EUROCRYPT*, 1994.

[2] A. Shamir, "How to share a secret," in *Communications of the ACM*, vol. 22, 1979, p. 612–613.

[3] W. Y. Jonathan Weir, "A Comprehensive Study of Visual Cryptography," in *Transactions on Data Hiding and Multimedia Security*, 2010, p. 70–105.

[4] Y.-C. Hou, "Visual cryptography for color images," in *Pattern Recognition*, vol. 36, 2003, pp. 1619–1629.

[5] V. R. A. L. Venkata Krishna Pavan Kalubandi, Hemanth Vaddi, "A Novel Image Encryption Algorithm using AES and Visual Cryptography," in *International Conference on Next Generation Computing Technologies*, 2016.

[6] R.-W. Y. Hsien-Chu Wu, Hao-Cheng Wang, "Color Visual Cryptography Scheme Using Meaningful Shares," in *International Conference on Intelligent Systems Design and Applications (ISDA)*, 2008, pp. 26–28.

[7] S. N.Soman, "XOR-Based visual cryptography," in *International Journal on Cybernetics and Informatics (IJCI)*, vol. 5, 2016, p. 2.