Image and Video Encryption using Wavelet transform and XOR logic

Samriddh Bhalla
Electronics and Communication Engg
Indian Institute of Technology, Roorkee
Roorkee, India
samriddh2004@gmail.com

Divyansh Jain
Electronics and Communication Engg
Indian Institute of Technology, Roorkee
Roorkee, India
kumardivyanshjain@gmail.com

***Abstract*—The purpose of the project was to understand the basics of digital image processing and to be able to implement certain algorithms using python. The project orignally comprised image encryption with wavelet transform. the purpose of scrambling is to prevent the image from statistical analysis attacks. Using the algorithm we were successfully able to encrypt an image and then decrypt it back to original image. Later on encryption of a video (live webcam) was also done successfully**

***Index Terms*—Image Encryption, Video encryption, wavelet transform, scrambling, XOR**

I. Introduction

The advancement of technology has been a boon to mankind. Unfortunately, a lot of people have used this advancement in a negative way; some of it being tampering and stealing the data of other users. It is necessary for us to be able to save ourselves from such people. The data may be in the form of audio, video, image, text etc. To give an example, consider a cracker stealing a personal image and using it for blackmailing, defamation purposes.

This is a report on the project. Section 2 explains the theory behind the encryption technique used in this project. Following section, section 3 explains about the theory of wavelet transform. Section 4 gives details about the algorithms implemented in the code. Section 5 discusses about the python code. Section 6 is a general discussion about the project and section 7 states the results.

II. Image Encryption

Encryption is a process of convert message data into a more secure data. Encryption and decryption usually require the use of a key. Based on the type of key used we can have symmetric and asymmetric encryption. In the symmetric key scheme the encryption and decryption key are the same and unknown to rest of the users. In asymmetric key scheme, the encryption key is known to public and the decryption key is private in nature. We have used a symmetric key scheme for encryption. The secret key used is an image produced by an algorithm. This algorithm converts a given image to a gray scale secret key.

III. Wavelet Transform

An important image transform is discrete wavelet transform(DWT). Haar wavelet transform has been used here. The basics of wavelet transform is that it converts the image into 4 component. The image is basically filtered using a low pass filter(L) and a high pass filter(H), row wise. The resultant 2 components are again filtered but this time column wise with L and H giving us 4 components - low low(LL), low high(LH), high low(HL) and high high(HH). Since the process is separable the process may be performed firstly column wise and then row wise. A 2 dimensional inverse wavelet transform(IDWT) will reconstruct back the original image.

In haar wavelet transform, the low pass filter is basically the half of sum of the $(2*i)^{th}$ and $(2*i + 1)^{th}$ element and the high pass filter is basically the half of difference of the $(2*i + 1)^{th}$ from $(2*i)^{th}$ element

Considering an example 1D image with pixel values - [a, b, c, d, ...]. The corresponding 1D discrete haar wavelet transformation will be as follows -

$$L = [\frac{(a+b)}{2}, \frac{(c+d)}{2}, ...] \tag{1}$$

$$H = [\frac{(a-b)}{2}, \frac{(c-d)}{2}, ...] \tag{2}$$

IV. Algorithm and explanation

The algorithm is divided into 3 parts

- Generation of secret key
- Scrambling and encryption
- Decryption and unscrambling

The size of an individual image is fixed for a specific image key. Also the image must be a square image, or else it will be converted to a square image using zero padding

A. Generation of secret key

1) Select a colour image
2) Convert the image into a square image with zero padding
3) Rotate color image to three directions (left, right and down)
4) Cut all 4 images and randomly permute all the pixels
5) Use XOR logic to combine the 4 resultant images into one image into one ”primary key” image
6) Split primary key into 3 channel B, G, R
7) Flip B to three directions (left to right, up to down and right to left)

8) Rotate B and flip it to three directions (left to right, up to down and right to left)
9) Do this for G and R channel
10) Use XOR logic to combine all the images into a single image
11) This image is the secret key

B. Scrambling and Encryption

1) Input message image (colour image)
2) Generate secret key from the message signal if required
3) Split the image into its 3 channel- B, G, R
4) Scramble each channel B, G, R using DWT
   a) Obtain 4 coefficients of the transform LL,(LH, HL, HH)
   b) Multiply last 3 coefficients with -1
   c) The final coefficients will be LL,(-LH, -HL, -HH)
   d) Use IDWT to obtain scrambled image
5) Encrypt every channel B, G, R by secret key using XOR
6) Merge the channels to create the cipher image

C. Decryption and Unscrambling

1) Input cipher image
2) Split the image into 3 channel B, G, R
3) Decrypt the each channel using secret key and obtain scrambled images
4) Unscramble each image (Apply scrambling on each image)
5) Merge the channel to obtain decrypted image

A video is a series of images that are shown one after another at a rate such that the human eye perceives it as a moving image. The encryption of video is therefore simply encryption of these individual images (known as frames). In this project the encryption of a live webcam video is demonstrated. Each next frame is selected after 2ms.

V. Python Code Related Explanations

The implementation of the algorithm in python required certain language related knowledge.

A. Packages imported

The following packages were imported

- cv2 - open cv
- pywt - open source package python wavelets
- numpy - for n dimension arrays
- time - to calculate runtime
- matlibplot

B. Functions used

- `time.time()` - find current system time
- `cv2.imread()` - reads the image
- `cv2.warpAffline()`
- `cv2.imwrite()` - saves images to disk
- `cv2.getRotationMatrix2D()` - rotates matrix
- `numpy.random.shuffle()` - random shuffling of image along rows
- `cv2.bitwise_xor()` - self explanatory
- `cv2.split()` - splits colour image in the order B,G,R
- `cv2.flip()` - flipping the image
- `cv2.imshow()` - display an image in window
- `cv.cvtColor()` - used to convert coloured image to grey scale
- `pywt.dwt2()` - finds 2 dimensional DWT
- `pywt.idwt2()` - finds 2 dimensional IDWT
- `np.clip()` - clips an ndarray between 2 given values
- `numpy.round()` - rounds ndarray to nearest integer
- `astype()` - converts ndarray to a given datatype

It is important to know that cv2 stores coloured images in BGR form whereas numpy does it in RGB format. The grey scale image stored in '.png' format is automatically to 3 channel image with all channels containing same pixel information i.e. R = G = B. The '.png' image stores indivisual pixels in uint8 (0 - 255). Scrambling process doesn't honour this range and thus the output image needs to be clipped between 0 and 255 so as to be able to be read by the cv2.imshow() function which otherwise throws an error. Clipping of the values is better that cyclic conversion of resultant out of range data to uint8 since the latter produces visible distortions.

VI. Discussion

The algorithm was tried and tested on multiple images of various dimensions. The recovery was however not lossless. The loss occurred because of the clipping operation that limited the pixel value between 0 and 255. The error is however not noticeable by the naked eye and must be scaled up by a significant factor(say about 10) to be slightly prominent.

The video was also encrypted and successfully decrypted (with a similiar type and amount of error one might expect with that of an image). The video was generated and by the webcam and was encrypted; decryption being done side by side.

VII. Results

Corresponding each image, a secret key, the encrypted and decrypted image is shown. The time taken (in seconds) for each algorithm for different images is shown in the table below

Image Size	Generate Secret Key	Encryption	Decryption
Figure 1 (640 X 640)	0.12499	0.28123	0.29685
Figure 2 (1554X1555)	0.56244	1.51552	1.58259
Figure 3 (550 X 550)	0.23435	0.28123	0.29685
Figure 4 (825 X 550)	0.18748	0.36278	0.38241

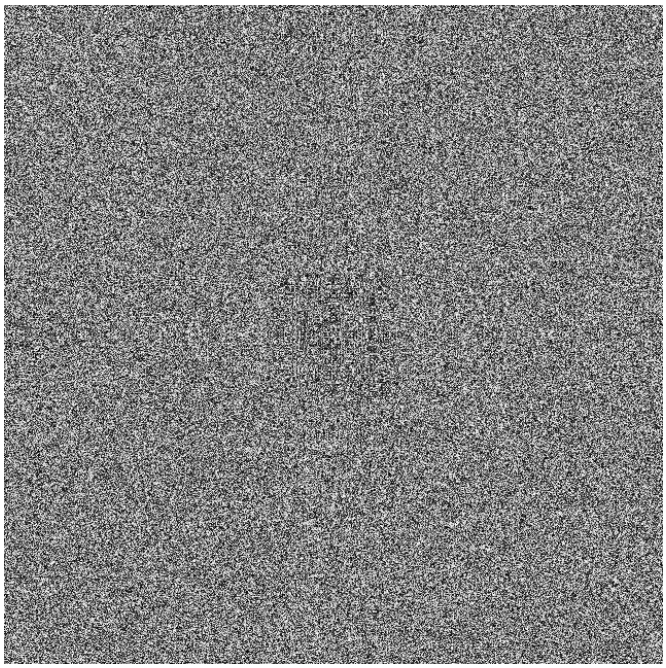
Acknowledgment

I would like to thank professor Saumik Bhattacharya for helping me complete this project.

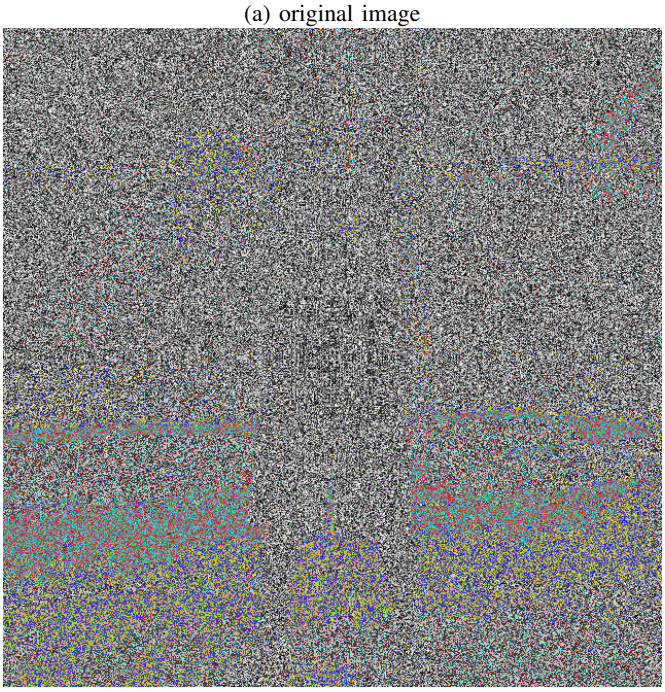
References

[1] Dira, A. (2016). Fast Image Encryption based on Random Image Key. International Journal of Computer Applications, 134(3), pp.35-43.

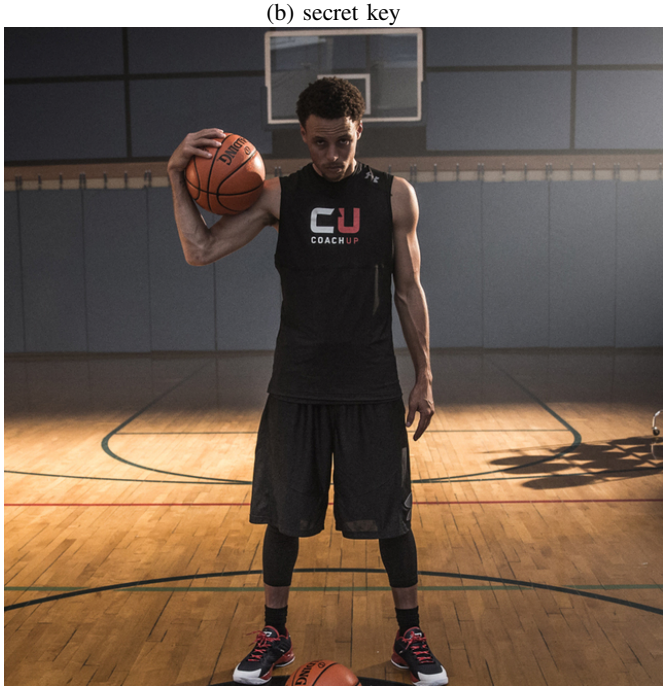
(a) original image



(b) secret key



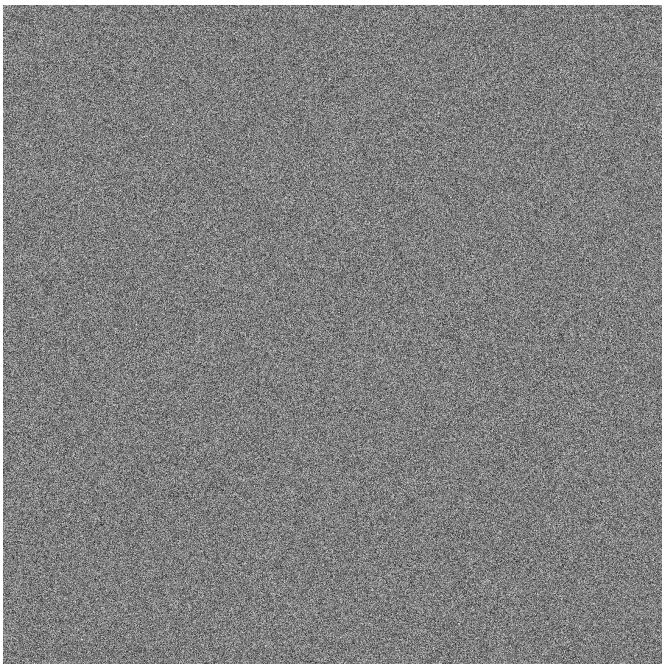
(c) encrypted image



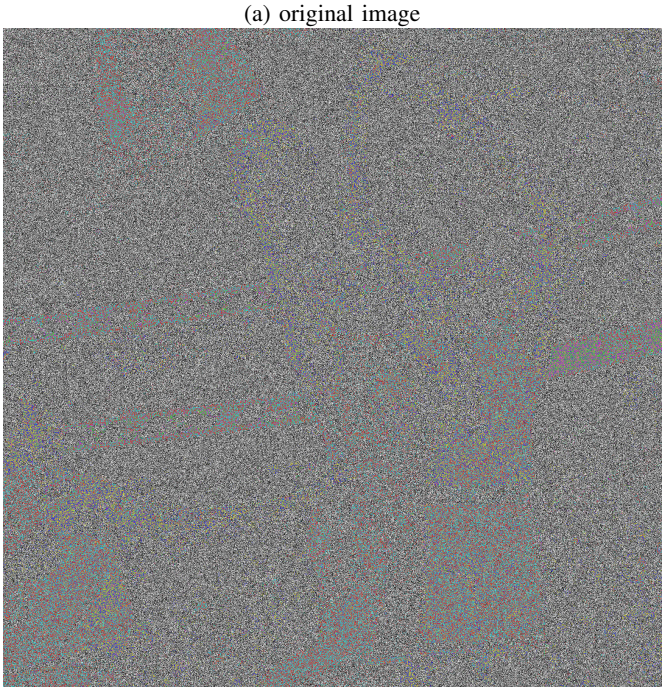
(d) decrypted image

Fig. 1: Stephen Curry (640 X 640)

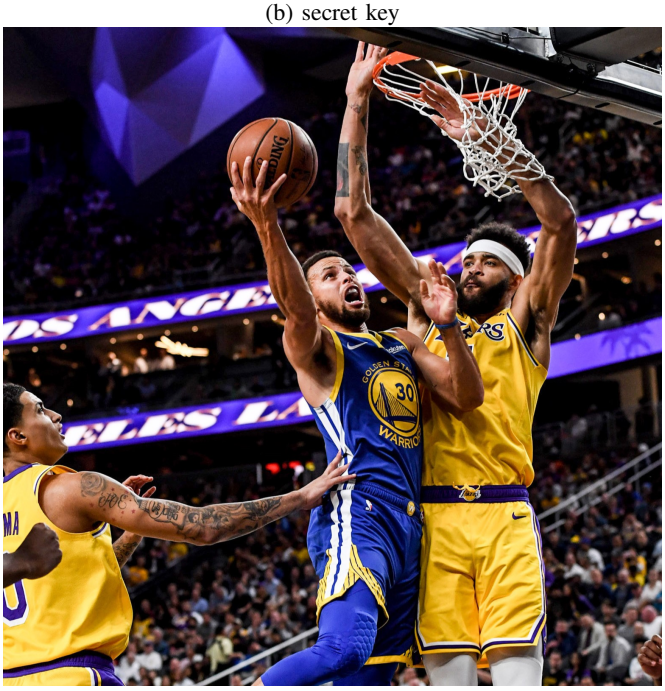
(a) original image



(b) secret key



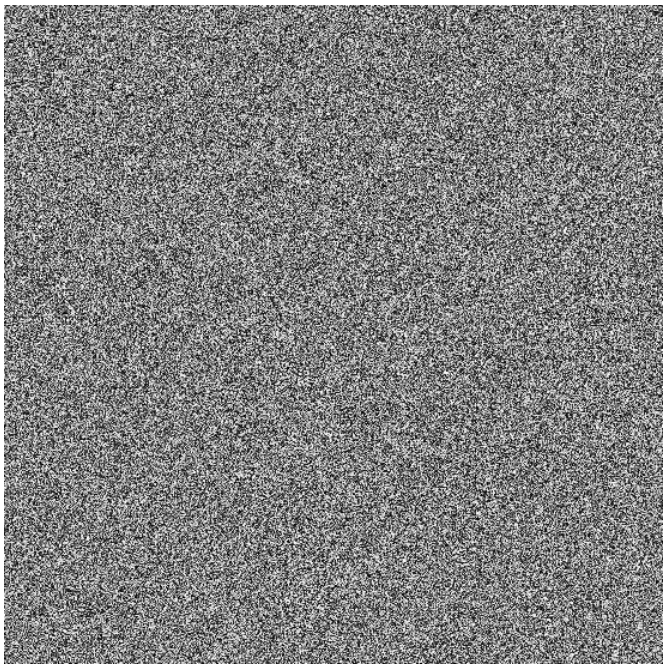
(c) encrypted image



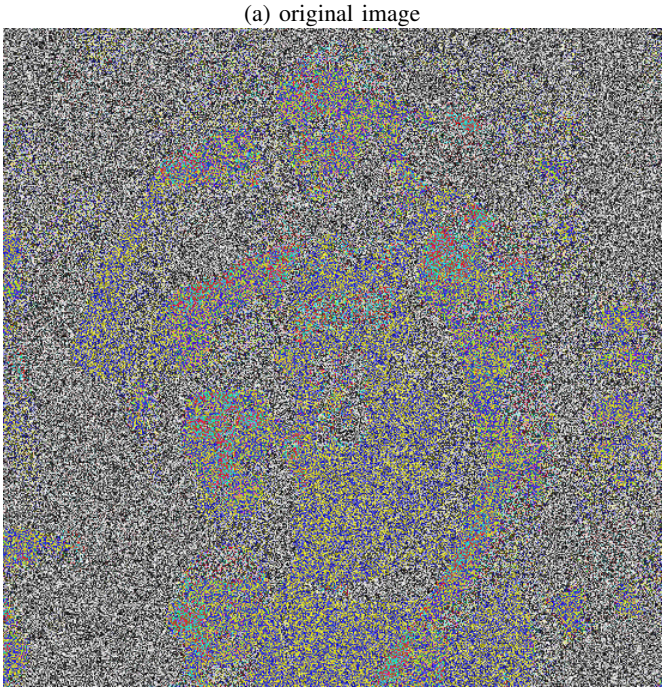
(d) decrypted image

Fig. 2: Stephen Curry and JaVale McGee (1554X1555)

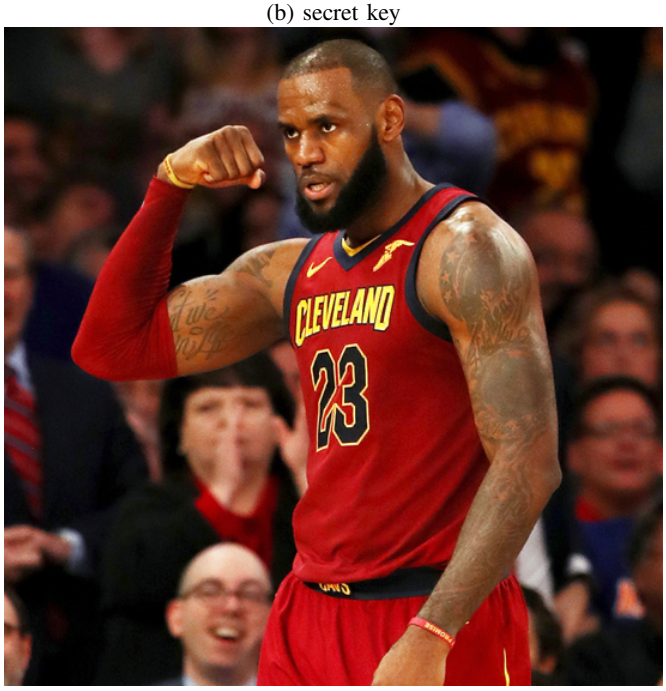
(a) original image



(b) secret key



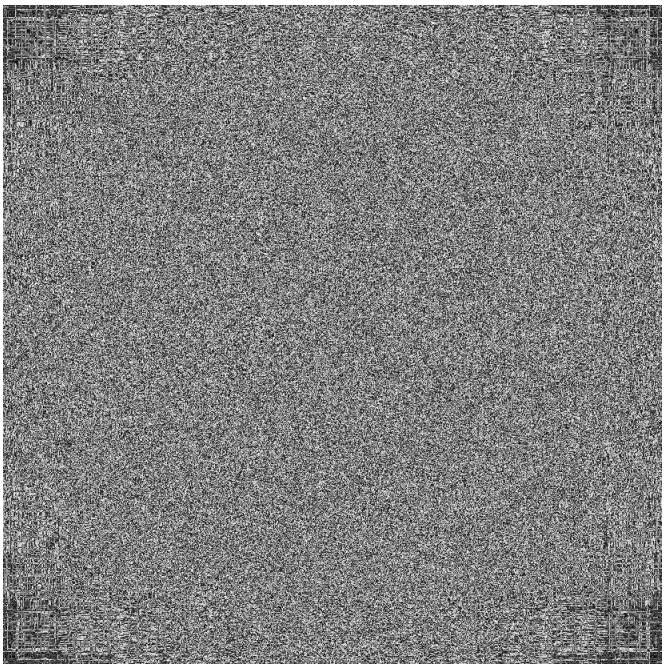
(c) encrypted image



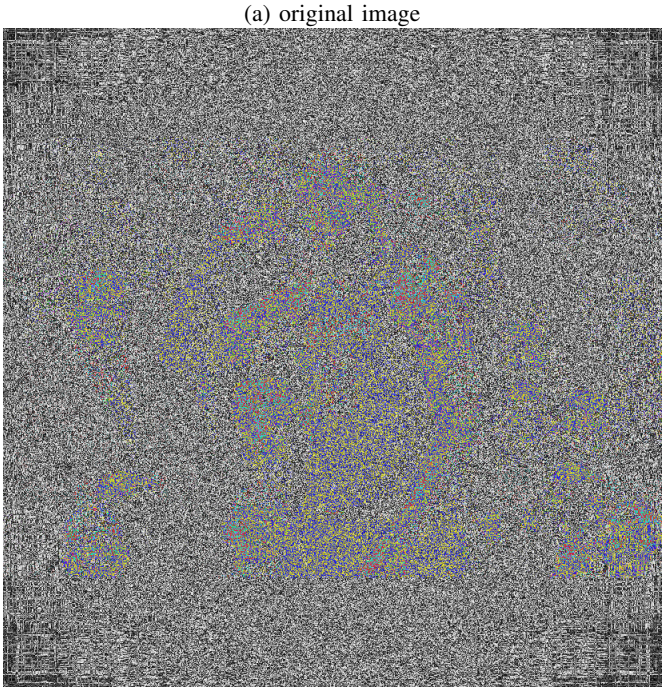
(d) decrypted image

Fig. 3: King James (550 X 550)

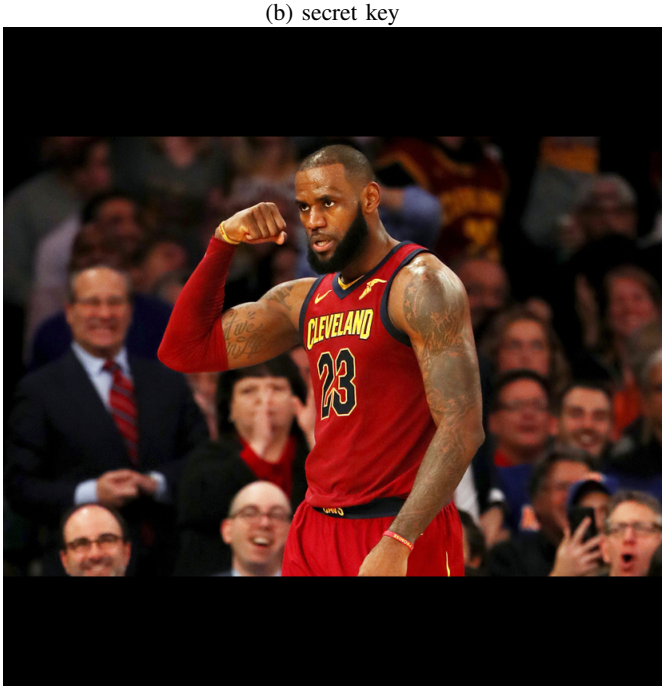
(a) original image



(b) secret key



(c) encrypted image



(d) decrypted image

Fig. 4: King James with zero padding (825 X 550)

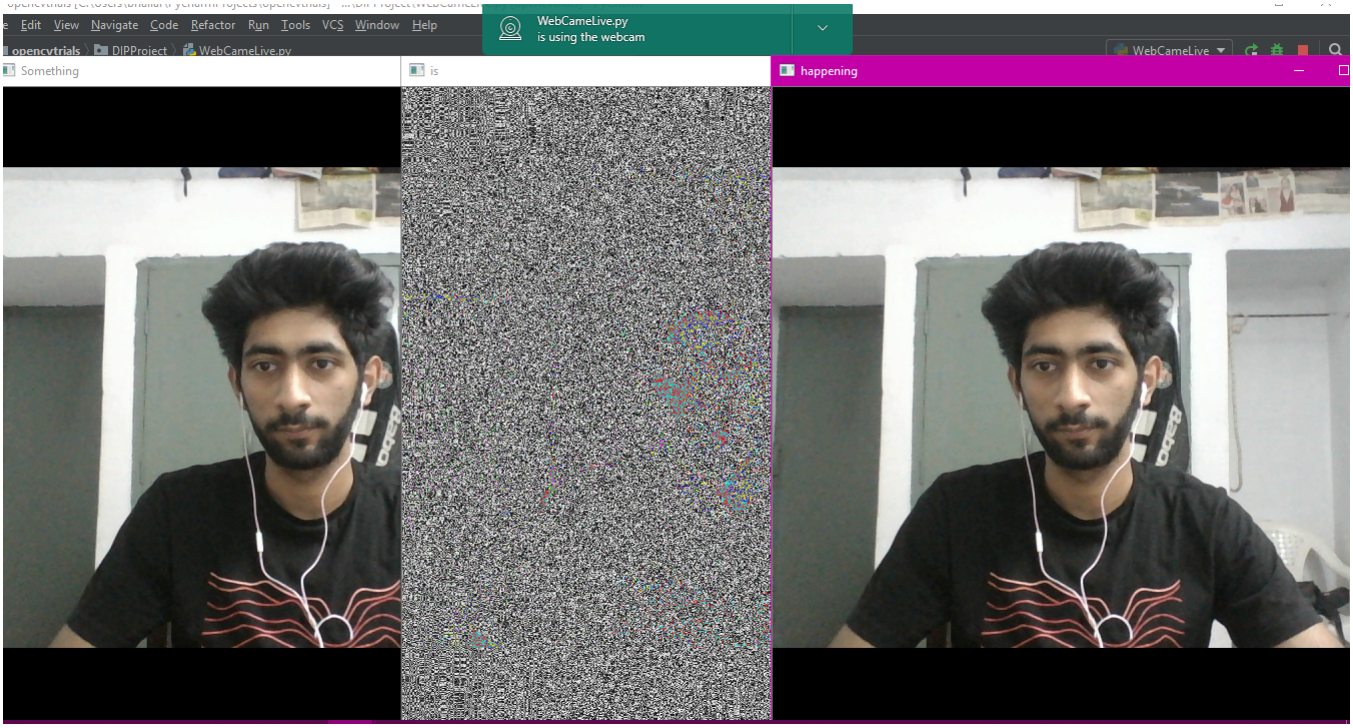


Fig. 5: Video encryption and decryption example