Self Embedding Watermarking System

Arbab Hamd Rizwan
Department of Computer Science
National University of Computer and Emerging Sciences
Lahore, Pakistan
arbabhamd@gmail.com

Aashar Naseem
Department of Computer Science
National University of Computer and Emerging Sciences
Lahore, Pakistan
aasharnaseem@gmail.com

Dr. Asif Mahmood Gilani
Department of Computer Science
National University of Computer and Emerging Sciences
Lahore, Pakistan
asif.gilani@nu.edu.pk

Hunzlah Malik
Department of Computer Science
National University of Computer and Emerging Sciences
Lahore, Pakistan
hunzlahmalik@gmail.com

Usama Aslam
Department of Computer Science
National University of Computer and Emerging Sciences
Lahore, Pakistan
usamavostro14@gmail.com

***Abstract*— Media editing software is easily accessible on the internet, which leads to the issue of manipulating and tampering with either an image or a video. To address this issue, the study developed a web-based application that watermarks media digitally and allows users to detect tampering and restore the media to its original state. The system watermarks an image by inserting the authentication and restoration bits into the LSB of the sub-blocks of the image, which are used to detect image tampering and restoration. Similarly, in the video case, our system first selects each frame from the stream of frames and watermarks it by inserting both the authentication and restoration bits into the LSB of the video's extracted frame/s sub-blocks. To determine whether or not the media has been tampered with, compute the hash value of the watermarked media and compare it to the authentication bits, which is the hash value computed from the original image. If the hash values are the same, the image would not be tampered with; otherwise, it has been tampered with. Similarly, the original media content is restored using data extracted from restoration bits. The experimental results show that the image can be restored up to 75%, implying that even if 75% of the image has been altered, it can be restored to its original state. Similarly, in the case of video, our experimental results show that in the case of temporal tampering in the video it can be detected by this system but cannot be recovered. In contrast, in the case of spatial tampering it can be recovered depending on the percentage of tampering in each frame, for each frame it can be restored up to 75% of the content, implying that even of 75% of that frame has been tampered.**

Keywords—bits embedding, watermarking, multimedia recovery, multi-threading

I. INTRODUCTION

Editing software has come a long way in the last decade. As a result, the general public now has access to complex editing tools, making it easier to manipulate digital assets like photographs and movies. This casts considerable questions on the credibility of any media offered. Image tampering is primarily concerned with changing the image as a whole. Conversely, video is made up of multiple frames that can be regarded as individual images.

Therefore, video tampering has been categorized into two types:

1. Temporal tampering refers to interframe editing manipulation. This type of tampering includes adding, removing, or changing the sequence of frames in a video.
2. Spatial tampering includes manipulating objects within a frame and is referred to as intraframe manipulation.

Through watermarking, we can detect tampering in an image or video. Our project will be focused on detecting and restoring these altered images and videos. Watermarking images entails breaking them down into smaller chunks and inserting data into the two LSBs. Two types of data are inserted into the LSB. The first type is the authentication bits which are the hashes of blocks and the other is the recovery bits which contain the average values of the other blocks. Using these two bits the system can detect alteration and restore the original image if it has been tampered with. Numerous approaches are investigated in research publications, varying depending on the number of blocks the image is divided into and the watermarking method [1 - 5]. For this study, “A novel triple recovery information embedding approach for self-embedded digital image watermarking” [1] is being used for the watermarking and restoration process.

This method could also be extended to support video watermarking, authentication, and restoration. This approach can also be utilized on video by detecting the keyframes and then processing these frames as if they were images [6]. Upon further studying the watermarking method using keyframes only in [6], it was noticed that video restoration would result in glitches for a large number of frame tampering. This is because only the keyframe is watermarked in the entire window of frames, and the tampered frames are being replaced with the keyframe. While this method may be useful to deal with temporal tampering, it does not work well against spatial tampering.

To deal with the abovementioned issue, this study extends the watermarking methods to support video watermarking and restoration. This was accomplished by watermarking each frame with the same method used on a single image. However, this method is very expensive in terms of computation. This is why multi-threading and parallel processing was also implemented to make the watermarking and restoration process faster.

The significant contributions of this work are as follows:

- We introduced a better implementation that is computationally faster than the results mentioned in "A novel triple recovery information embedding approach for self-embedded digital

image watermarking" [1] by treating the whole image as a single array and using strides to manipulate the image.

- We added RGB support for watermarking and restoration process.
- We extended the method used in "A novel triple recovery information embedding approach for self-embedded digital image watermarking" [1] to work with video watermarking and authentication.
- We implemented multi-threading and parallel computing to make the process of video watermarking and restoration faster.
- Using the watermarking method, we have also increased the PSNR value of the watermarked image as compared to the ones mentioned in the triple recovery approach [1].
- Made a comparison of two different watermarking methods [1, 2].
- We also improved the outcomes for the recovered image. Initially, the recovered area on the image had a rectangular texture, which is difficult to see in the results of the triple recovery approach [1]. The value of the area where the recovered block ends is the average of the blocks, not the actual value, which would give the recovered areas of the image a rectangular border look. To address this issue, we used interpolation to smoothen the recovered block, particularly around the edges. This will deal with the recovered image's boxed texture. This, however, blurs out the recovered area by a small amount. It can be addressed in future works by applying interpolation to the entire image rather than just the recovered portion.

II. Related work

Many kinds of research relating to this field have been conducted in the past. Each research either has its unique method or improves a previous method. Here, we will discuss the various methods related to our project. The three main methods for image watermarking include embedding data in bits into its least significant bits (to avoid reducing image quality). During our study, we came across the following image watermarking methods.

A. Triple Recovery Information Embedding Approach

The embedded bits in the fragile watermarking of the image contain both the authentication bits and the recovery bits, which are used to detect tamper detection and recover the image. This method [1] divides the original image into 16x16 non-overlapping main blocks, after which a lookup table is generated from which four random main blocks, known as partner blocks, are chosen. Then, each partner block is divided into 4x4 blocks, the average value of the partner block is calculated and converted to binary, and similarly, other blocks are converted to binary, and each of these binary bits from the partner blocks is merged into other partner blocks. The recovery bits are then permuted using a key. The partner blocks are then subdivided into 16x16 blocks, and these 16x16 blocks are subdivided even further into 8x8 blocks. The recovery bits are then embedded in the first and second LSBs of the first three 8x8 subblocks. The algorithm then checks to see if all partner blocks have been grouped before combining the divided blocks to create the original watermarked image.

B. Pixel-wise Authentication Based Self-Embedding Fragile Watermarking Method

This method [2] involves watermarking the original image by dividing it into four equal parts known as main blocks. The algorithm then divides the main blocks into 4x2 or 2x4 subblocks, depending on the type of block. The recovery bits of each of the four main blocks are formed by computing the average values of the divided blocks and combining them. The recovery bits for each main block are then created by combining the recovery bits from the two main blocks in the other half of the image. Following the creation of the recovery bits, the four MSBs of the pixel, the recovery bit, and the two-pixel position bits are used to generate two authentication bits for each pixel. The authentication bits are then embedded in the pixel's first and second LSBs, while the recovery bit is embedded in the pixel's third LSB.

C. Hash Function Based Fragile Watermarking

This method is an older version of the triple recovery method, the main difference between this method and the triple recovery method is the accuracy and process of block division. The image is watermarked in this method [3] by first dividing it into 32x32 non-overlapping blocks, which are then divided into 16x16 non-overlapping sub-blocks. The 256-bit hash value of the first three subblocks is then calculated using a hashing algorithm. The hash value 16x16 bit binary watermarked is then generated. The watermark computed by the first three subblocks is then modified in the fourth subblock. The same procedure is followed for all 32x32 blocks. The blocks are then combined to form the original, watermarked image.

D. Fragile Watermarking based on Adaptive Selection of Embedding Mode

This method is from older research as compared to the ones mentioned before [1-3]. Like other watermark embedding schemes, this method divides the image into N/b^2 (where N is assumed to be multiple of b) non-overlapping image blocks with sizes b x b, which is later allocated an authentication bit generated through a hash function. These bits are stored in the LSBs and used as references when an image needs to be checked for tampering [4]. However, this method was not considered for the implementation of this project due to the lower tamper recovery rates and the fact that other methods can be considered the latest work in this field.

E. Video Watermarking

The concept of image watermarking was extended to videos as well. This method was proposed in "Video-Tampering Detection and Content Reconstruction via Self-Embedding," [4]. This paper uses the already present image watermarking methods and applies those to videos. The system is carried out by first selecting a keyframe in a window of frames and then watermarking that keyframe by any of the image watermarking methods available. However, a video can have many keyframes, and each keyframe will need to be processed and watermarked to watermark the whole video completely. This process can be time-consuming, and thus, it will require a lot of processing power and good programming logic for a faster watermarking process.

However, the algorithm mentioned above for video watermarking is not the best solution for video recovery. The main reason is the watermarking process in which we only watermark keyframes and then use those frames to replace the tampered frames. In case some frames are missing, the same watermarked keyframe extracted from a frame window is placed into the position of the frames that were removed (temporal tampering). The system will work and provide excellent results when removing a few frames. Assuming the attacker tampers and removes two frames, the above-mentioned method will work perfectly in this case. On the contrary, when a large portion of the video is trimmed (assuming 5 seconds of video), we would be able to deal with temporal tampering by adding the keyframes in place of removed frames. It will create a glitch in the video because the same keyframe would be repeated to cover all the missing frames.

The above-mentioned method is unable to deal with such a scenario, which is why we will not deal with temporal tampering and will shift our main focus to detecting and recovering spatial tampering. One of the main drawbacks of this approach is the computation cost because we would be applying image watermarking on all of the frames of video, which can take a lot of time. To deal with this, we will implement multithreading and parallel processing on the server-side by using Lithops, a multi-cloud serverless programming framework.

F. Spatiotemporal Trident Networks: Detection and Localization of Object Removal Tampering

Unlike other methods, which embed data into the image or video. This method uses artificial intelligence techniques to detect object removal tampering in videos. This is carried out by training a CNN model [7] by giving it various datasets of videos. Once the model has finished training, it can process a video and detect object removal tampering without placing any data into the video itself. The inputted video is processed in a series of odd-numbered frames, which are divided and inserted through three different branches of inputs. Once they have been processed, they are labeled as pristine or forged frames. If even one forged frame is found in a frame window, the whole video is marked as tampered with. This method may seem useful and reliable, but it has limitations. The main limitation of this method was that it could only detect object removal tampering. The other limitation is that, unlike other methods, restoring a tampered video is impossible because no data is placed in the video, which can be used to restore the original video.

G. Keyframe Extraction

In animation and cinematography, a keyframe (or keyframe) is a graphic or shot that marks any seamless transition's beginning and finish locations. Because their position in time is measured in frames on a strip of film or on a digital video editing timeline, these are referred to as frames. The position of the keyframes on the film, video, or animation determines which movement the viewer will see, whereas the sequence of keyframes determines the timing of the movement. The remaining frames are filled with "in-betweens" because only two or three crucial frames throughout a second do not generate the appearance of movement. There are many methods for keyframe extraction, but only a few methods will be discussed here, including the histogram method, motion perceived energy model, and cascaded map-reduce method.

1) Key-Frame-Extraction Based on Perceived Motion Energy Model

This method uses motion patterns of a shot to determine whether that frame is a keyframe or not. A motion pattern of a shot is usually composed of a motion acceleration process, followed by a deceleration process. Such a motion pattern usually reflects an action in events [8]. This method works by building a motion model used in the paper as a triangle model of perceived motion energy (PME). A motion triangle is generated for each frame, and its PME value is compared. Each motion pattern's turning point and acceleration deceleration are selected as a key frame.

2) Cascaded Map Reduce Method

This method makes use of Apache Hadoop, MapReduce Framework, and HDFS. These technologies are used because depending on the size and frames per second of a video, it can have a lot of frames even if the video size is 5MB. For a video, 20 – 30 frames per second are quite common. If a video is even 10 seconds long, it will result in many frames; for our system, a 10-second video is quite small. The cascaded algorithm contains three MapReduce algorithms for the whole algorithm. This method works similarly to the histogram method that will be discussed next. A color histogram is calculated for each frame, and frame difference is calculated using Euclidean distance, mean, and standard deviation of absolute difference. These are used to compute the threshold, which will identify keyframes by checking if a specific frame has a value above the threshold [9].

3) Keyframe Extraction Based on Dynamic Color Histogram

It is the method that will be used for this keyframe extraction algorithm. One frame is used to generate two different types of histograms. One is a color histogram, and the other is a fast wavelength histogram; they both give a sequence of keyframes. The keyframes are selected through an optimized k-means algorithm for both attribute features (color and fast wavelength histogram). The two keyframe sequences are compared through mutual information, and redundant keyframes are removed. This method performs keyframe extraction better because its color information and texture detail features are extracted by descriptors, which are selected automatically by the optimal k-means algorithm [10].

H. Lithops, a Multi-cloud Serverless Programming Framework

Lithops is a robust multi-cloud framework that allows local, multi-process Python programs to scale seamlessly into huge amounts of cloud resources [11]. It is an open-source project that is also available on GitHub. Lithops makes it simple to spawn hundreds or thousands of jobs to complete a huge work in a matter of seconds. Lithops may be considered a dynamic job orchestrator geared for running jobs in a serverless computing environment. The system will work well to use multithreading on the server-side so that the user does not have to do all the computing on their side. This framework can use various serverless platforms such as AWS Lambda, Google Cloud Run, or IBM Cloud. Lithops works by transferring the locally created classes and transferring them to the cloud. In the background, it normally uses an object storage service (e.g., AWS S3) to store this information and other intermediate data.

I. *Conclusion*

In conclusion, it is necessary to properly consider the optimal method for tamper detection and restoration of multimedia. Therefore, we will follow two different methods [1, 2] that give the best results in terms of percentage of tamper detection and restoration while also keeping in account the quality of multimedia. Both these methods shall be implemented, and their results will be compared. Moreover, based on the obtained results, the method with the best results will be extended to support video tamper detection and restoration. Our method for video watermarking will mainly be to watermark all video frames, but this is computationally very expensive, which is why we will be using different approaches to reduce this time as much as possible.

III. METHODOLOGY

We have implemented two different watermarking methods. One is the triple recovery method [1], and the other is the pixel-wise authentication method [2]. Reference papers [1, 2] were followed to implement both methods.

Initially, both methods were implemented as the reference papers stated. Their results and execution timings were compared to decide which one will be extended support video watermarking. The results obtained from implementing the triple recovery method [1] were similar to the ones mentioned in the paper. For the embedding process, the execution time for a grayscale image (Lena) of 512 × 512 dimensions was around 180 seconds. The PSNR value obtained was similar to the one mentioned in the paper. The next implementation used the pixel-wise method [2]. The same image was used to test this method and the results obtained were the same as the ones mentioned in the reference paper. However, due to the low PSNR values of the watermarked image, it was impractical to use this method on a high-resolution image. That being the case, the triple recovery method [1] was focused on for this study and extended to support RGB images and video watermarking.

The main methodology is as explained in [1]. But we are using a different approach for hashing and image block making and a major part comes in the restoration.

For recovery, we are using a different approach. Instead of dividing the image into 16 blocks then 16x16 blocks and then 4x4 blocks just replace this whole 4x4 block with the average values, we stored in the partner blocks.

For the recovery, the process is as follows.

1- Extract the auth bits

2- Using the auth bits identify the areas that are manipulated.

3- Make a mask image equal to the size of the original image.

4- Extract the recovery bits

5- Try the make the original stored image

6- Interpolate that image to the original size

7- Replace the pixels of the original image using the mask array and interpolated image.

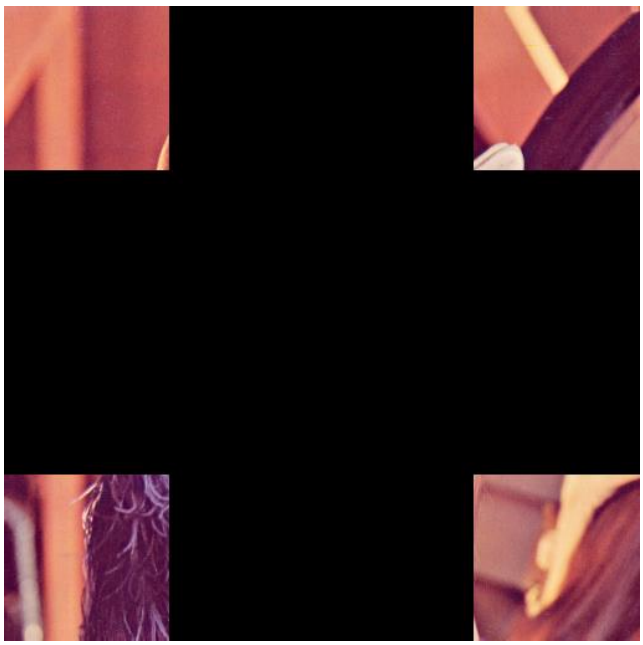
As mentioned above the original paper is shirking using the nearest neighbor algorithm. But there are many methods of interpolation. We can use any, but the condition is which interpolation is best for shirking and then enlarging the same image. Using our comparison, we deduced that we get the best PSNR and SSMI using the area interpolation for shirking, and LANCZOS4 for enlarging performs best regarding the PSNR and SSMI.

1: Extract the auth bits.

Extract the auth bits as explained in the original paper.

Make a mask image and use NEAREST interpolation to make the size of the original image. For image edited as

Figure 1: Tampered Image



The mask should be like this. The Black area represents the tempered areas and the white represents the non-tempered areas.

Figure 2: Mask Image to Detect Tampered Area

4. Extract the recovery bits using the method described in the paper

5. Now we need to use the extracted recovery bits and mask images for this part.

We will try to extract the tempered area parts using the non-tempered part. After that we need to construct the original image, it will be the stored value of the image. While constructing if the area is not tempered we can use the bits of the given image.

6. Using the constructed image to and interpolate is by 4 times enlarge using LANCZOS4

7. Next, we need to replace the parts of the original array by using the mask array. Replace all the pixels which are tempered with.

Image manipulation:

For the embedding method in [1], we are required to make any blocks of images. The traditional method of making image blocks includes the copy of pixels to another array and copying is very time-consuming. And time increases by the size of the image. For a method to work and value the time should be minimum, so we started our search of making blocks without copying the data. We know that the grayscale image is a 2D array. But if we look at the memory of that 2D array, it's contiguous. It means that for the 2D array the system is managing the memory. What if we are the ones managing the memory? For example, we will have full control over our image shape for the grayscale image of size 4x4. We will need 3*3=9bytes, and that 9bytes will be contiguous in memory. But for the image to show us in 2D shape we have to change the row after every 3 cols. For Image[row][cols], Image [0][0] means that we are accessing the pixel at byte 0, and Image [1][1] means that we are accessing the pixel at $3*1+1= 4^{th}$ byte.

For simplicity let's just say that the rows and columns are our axes. Axes 0 represents the rows and axes 1 represents the columns. And we also need to specify how many jumps are required to move along the axes. We are going to name it strides. So, for a 1D image to be able to represent in a 2D image we need two axes and two strides. For the above example, we need two axes and two strides. Axes0 (row) and axes1 (column), the jump for axes0 will be 3, and the jump for axes1 will be 1. Because to move to the next row we need to skip 3btyes, and to move along the next column we need to skip 1 byte.

The previous example showed that the 1D array could be represented and manipulated as 2D. For the specific pixel we have to follow this formula, pixel=axis0value*stride0+axis1Value*stride0. So, the access the image [2][2], the pixel will be at $2*3+2*1=8^{th}$ byte. There are just the demonstrations of how we can have representation of our choice. This not only saves time but also saves the memory of copying the pixels. The making of blocks in the image using the above strides method is explained in the "Efficiently splitting an image into tiles in Python using NumPy" [12] article. This article explains the process and then implements it in the python language, but get the idea of how we are making our 16blocks, then 16x16 blocks then 8x8 blocks, and then 4x4 blocks.

IV. Experimantal results

A. Image Experimental Results

In the case of an image, the thing which affects the entire system process is the size of the image the greater the size more time is needed to process the image which includes watermarking and recovering the image. In the table below the tamper, attacks are performed on Blonde Women's images. The dimensions of the image are (512,512) and the PSNR value and the SSIM value of both the watermarked image and recovered image are shown.
In the tamper column, the first digits indicate the attack percentage while the alphabets after the underscore show the attack region. The complete form of the regions mentioned in the table are LZ stands for the left zone, RZ stands for the right zone, CZ stands for the center zone and TZ stands for the top zone, etc.

TABLE I. PSNR and SSIM

Tamper	**PSNR**		**SSMI**	
	Watermark	**Recover**	**Recover**	**Watermark**
75_LZ	44.45362	27.6925	0.822362	0.985673
75_TZ	44.45362	27.4275	0.811678	0.985673
50_TZ	44.45362	28.8401	0.859006	0.985673
50_HCZ	44.45362	28.6855	0.872447	0.985673
75_LZ	44.45362	28.5682	0.658666	0.995066
75_CZ	44.45362	27.5166	0.627256	0.995066
75_LZ	44.45362	30.3242	0.933836	0.969975
75_RZ	44.45362	28.4724	0.931869	0.969975
50_VCZ	44.45362	30.7782	0.887884	0.986936
25_CZ	44.45362	30.0423	0.935459	0.986936
75_LZ	44.45362	28.6286	0.958274	0.975372
75_TZ	44.45362	30.5280	0.973823	0.975372
75_OZ	44.45362	33.6077	0.960633	0.975372
50_TZ	44.45362	32.7419	0.978705	0.975372
50_LZ	44.45362	33.4000	0.963927	0.975372
75_BZ	44.45362	32.4473	0.796497	0.987493
75_TZ	44.45362	33.0009	0.813282	0.987493
75_LZ	44.45362	33.0110	0.948564	0.980327
50_VCZ	44.45362	33.2361	0.952073	0.980327
75_LZ	44.45362	27.6925	0.729628	0.994636

Table II shows the time comparisons of the time taken to watermark the blonde women's image, the authentication time to authenticate the image, and finally the recovery time to recover the image. As mentioned earlier the time depends heavily on the size and dimensions of the image.

TABLE II. Time Values

Time		
Watermarked	**Authenticate**	**Recovered**
2.432039	0.455464	0.003966
2.432039	0.408056	0.002594
2.432039	0.415336	0.002755
2.432039	0.411475	0.002606
2.432039	0.421629	0.003278
2.432039	0.401182	0.002731
2.432039	0.414425	0.002388
2.432039	0.405309	0.00246
2.432039	0.413929	0.002329
2.432039	0.412717	0.00242

2.432039	0.408573	0.002532
2.432039	0.4134	0.003356
2.432039	0.486874	0.00241
2.432039	0.404045	0.002229
2.432039	0.413292	0.00223
2.432039	0.401593	0.002209
2.432039	0.418031	0.002255
2.432039	0.529003	0.002271
2.432039	0.480682	0.002278
2.432039	0.455464	0.003966

Tables I and II show us the experimental results of the image named Blonde Women on which different tampering attacks are performed and as a result, the PSNR value, SSIM, and time is taken to perform various functionalities are shown.

B. Video Experimental Results

In the video, the main thing which affects the system is the size of the video and frames per second(fps) of the video. As threading is used in the video to process the frames faster, the time results also depend on the machine's capabilities.

TABLE III. Video Results

Video	Size	Tamper %	Recovered %	Completely Recovered ?
Video1	1.15 MB	50	50	Yes
Video2	600 KB	10	10	Yes
Video3	550 KB	40	40	Yes
Video4	2.28 MB	60	50	Yes
Video5	1.1 MB	65	0	No
Video6	1.06	20	20	Yes

In the experimental results of the video, it can be seen that a video that has been tampered with more than 50% is either not recovered or partially recovered. The size of the video plays an important role in the processing of the video, if the size is large means a large number of frames which in turn means that a lot of time will be taken to process the video. Moreover, as we are working on a block level the quality of the recovered video is reduced especially in the recovered region.

REFERENCES

[1] Gul, E., Ozturk, S. A novel triple recovery information embedding approach for self-embedded digital image watermarking. Multimed Tools Appl 79, 31239–31264 (2020). https://doi.org/10.1007/s11042-020-09548-4

[2] Gul, E., Ozturk, S. A novel pixel-wise authentication-based self-embedding fragile watermarking method. Multimedia Systems 27, 531–545 (2021). https://doi.org/10.1007/s00530-021-00751-3

[3] Gul, E., Ozturk, S. A novel hash function based fragile watermarking method for image integrity. Multimed Tools Appl 78, 17701–17718 (2019). https://doi.org/10.1007/s11042-018-7084-0

[4] Qin, C., Wang, H., & Sun, X. (2016). Self-embedding fragile watermarking based on reference-data interleaving and adaptive selection of embedding mode (Pages 233–250, Volume 373). Information Sciences. https://doi.org/10.1016/j.ins.2016.09.001.

[5] Kim, C., & Yang, C.-N. (2021). Self-Embedding Fragile Watermarking Scheme to Detect Image Tampering Using AMBTC and OPAP Approaches. Applied Sciences, 11(3), 1146. https://doi.org/10.3390/app11031146

[6] V. Amanipour and S. Ghaemmaghami, "Video-Tampering Detection and Content Reconstruction via Self-Embedding," in IEEE Transactions on Instrumentation and Measurement, vol. 67, no. 3, pp. 505-515, March 2018, doi: 10.1109/TIM.2017.2777620.

[7] Q. Yang, D. Yu, Z. Zhang, Y. Yao, and L. Chen, "Spatiotemporal Trident Networks: Detection and Localization of Object Removal Tampering in Video Passive Forensics," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 31, no. 10, pp. 4131-4144, Oct. 2021, doi: 10.1109/TCSVT.2020.3046240.

[8] Tianming Liu, Hong-Jiang Zhang and Feihu Qi, "A novel video key-frame-extraction algorithm based on perceived motion energy model," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 10, pp. 1006-1013, Oct. 2003, doi: 10.1109/TCSVT.2003.816521.

[9] P. Deshmane, "MRKFE: Designing cascaded map-reduce algorithm for key frame extraction," 2017 International Conference on Information Communication and Embedded Systems (ICICES), 2017, pp. 1-6, doi: 10.1109/ICICES.2017.8070719.

[10] Z. Zong and Q. Gong, "Key frame extraction based on dynamic color histogram and fast wavelet histogram," 2017 IEEE International Conference on Information and Automation (ICIA), 2017, pp. 183-188, doi: 10.1109/ICInfA.2017.8078903.

[11] Lithops, a Multi-cloud Serverless Programming Framework. (2021, March 9). ITNext. Retrieved June 1, 2022, from https://itnext.io/lithops-a-multi-cloud-serverless-programming-framework-fd97f0d5e9e4

[12] Doundoulakis, l., 2021. Efficiently splitting an image into tiles in Python using NumPy. [online] Medium. Available at: https://towardsdatascience.com/efficiently-splitting-an-image-into-tiles-in-python-using-numpy-d1bf0dd7b6f7 [Accessed 28 December 2021].