electronics

MDPI

Article

Security Improvements of JPEG Images Using Image De-Identification

Ho-Seok Kang [1], Seongjun Cha [2] and Sung-Ryul Kim [3,*]

[1] Institute for Ubiquitous Information Technology and Applications (UbiTA), Konkuk University, Seoul 05029, Republic of Korea; hsriver@gmail.com
[2] INTHESMART (iTS) Corporation, Seoul 03082, Republic of Korea; cktjdwns97@gmail.com
[3] Department of Computer Engineering, Konkuk University, Seoul 05029, Republic of Korea
* Correspondence: kimsr@konkuk.ac.kr; Tel.: +82-2-450-4134

Abstract: Today, as data is easily exposed through various channels, such as storing data in cloud services or exchanging data through a SNS (Social Network Service), related privacy issues are receiving a significant amount of attention. In addition, for data that are more sensitive to personal information, such as medical images, more attention should be paid to privacy protection. De-identification is a common method for privacy protection. Typically, it is a method of deleting or masking individual identifiers and omitting quasi-identifiers such as birth dates. In the case of images, de-identification is performed by mosaic processing or applying various effects. In this paper, we present a method of de-identifying an image by encrypting only some of the data in the JPEG (Joint Photograph Experts Group) image format, one of the most common image compression formats, so that the entire image cannot be recognized. The purpose of this paper is to protect images by encrypting only small parts, and not the entire image. This work is suitable for the fast and safe transmission and verification of high-capacity images. We have shown that images can be de-identified by encrypting data from the DHT (Define Huffman Table) segment among the JPEG header segments. Through experiments, we confirmed that that these images could not be identified after encrypting only a minimal portion, compared to previous studies that encrypted entire images, and the encryption speed and decryption speed were also faster and more effective than the results of previous studies. A model was implemented to de-identify images using AES-256 (Advanced Encryption Standard-256) and symmetric key encryption algorithm in the Huffman tables of JPEG headers, resulting in the ability to render entire images unidentifiable quickly and effectively.

Keywords: JPEG image; encryption; security

check for updates

Citation: Kang, H.-S.; Cha, S.; Kim, S.-R. Security Improvements of JPEG Images Using Image De-Identification. *Electronics* **2024**, *13*, 1332. https://doi.org/10.3390/electronics13071332

Academic Editor: Dimitris Apostolou

Received: 5 March 2024
Revised: 30 March 2024
Accepted: 31 March 2024
Published: 2 April 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The production of data is increasing exponentially due to the contemporary development of mobile technology, SNS, and IoT. The data produced can become more easily exposed when they are stored on an external server or are freely exchanged through the Internet. Some data comprise sensitive personal information, such as medical images. Many researchers have actively studied the security and privacy protection of various data according to various environments [1]. Image data must be protected more carefully because they can easily expose sensitive information, such as people or places, and there is a high risk of identifying an individual in an image. Algorithms of M. Zeghid et al. [2] and Singh et al. [3] evaluated ways of protecting data by encrypting image data. These studies allow us to protect the privacy of users by making an original image unidentifiable when an unauthorized person accesses the data. This study aimed to propose a method to make an image unidentifiable by encrypting only a part of the JPEG image, which is one of the image formats commonly used on the Internet. If it is possible to make an image unidentifiable by encrypting a specific part of the JPEG image data that significantly changes

the image, it will be possible to expect more effective performance than the results of that algorithms [2,3], which encrypt the entire image. In this paper, we realized a model that made entire images unidentifiable by encrypting their Huffman code table, one component of the JPEG header, to satisfy the above conditions. The purpose of this paper is to allow encryption to be maintained in real time if many image files are transferred.

Section 2 explains the AES encryption algorithm, the process of compressing a JPEG, and the structure of the JPEG header, using related studies, background knowledge, and preliminary work. Section 3 implements the proposed model and examines its performance. Finally, Section 4 describes the conclusion and future tasks.

2. Preliminaries

2.1. AES

Advanced Encryption Standard, AES [4], an algorithm, was developed as the encryption strength of the DES (Data Encryption Standard) weakened [4]. AES is a symmetric block cipher established by the National Institute of Standards and Technology, NIST, in 2001. It is easy to implement, fast, and highly secure. Therefore, there has been no valid attack against AES encryption yet. AES can have encryption keys of 128, 192, and 256 bits, and they are called AES-128, AES-192, and AES-256, respectively [5]. Moreover, the number of rounds to be executed varies according to the length of the key used, and 10, 12, and 14 rounds are executed, respectively. When the length of the encryption key becomes longer, the encryption speed and decryption speed relatively decrease, and the level of security increases. A feature of AES is that encryption is performed in fixed block units of 128 bits. When performing encryption, various block cipher modes can be selected. CBC (Cipher Block Chaining) and ECB (Electronic Code Block) modes are representative modes, and it is recommended to use CBC mode. CBC stands for cipher-block chaining [6], which was developed by IBM in 1976. Each block is XORed with the encryption result of the previous block before being encrypted, and the first block is XORed with the initialization vector (IV), an initial vector. Since encryption is performed in the block unit of 128 bits, AES carries out padding that fills the missing part with a specific value for blocks smaller than 128 bits. PKCS5 and PKCS7 are representative methods. This study used AES-256, the CBC mode for the Block Cipher Mode, and the PKCS5 method for the padding.

2.2. JPEG Compression

Joint Photographic Experts Group, JPEG [7], is a standard established by ISO (International Organization for Standardization) and ITU-T (International Telecommunication Union Telecommunication Standardization Sector), and is a lossy compression method created for still images [8]. The lossy compression refers to artificially removing some data to reduce the size of the original file. JPEG loses data in the process of removing high-frequency components during the JPEG compression process. Since it tries to eliminate parts that are difficult to distinguish with the human eye, the result is not much different from the original. If an image is compressed to a JPEG format, its size can be compressed to up to 1/50 of the original image file [9]. This format is widely used on the Internet because the file size is small. Since it uses a lossy compression method, the image quality and file size vary depending on the compression rate. A higher compression rate makes the size of a compressed file smaller and decreases the quality of the image.

Figure 1 depicts the process of JPEG compression. JPEG compression proceeds in the following order: image color space transformation, sub-sampling, DCT (Discrete Cosine Transformation), quantization, zigzag scanning, and entropy coding. Image color space conversion, down-sampling, DCT, and quantization are the processes that lose data. The majority of data is lost during the quantization process. Zigzag scanning and entropy coding are the processes of data compression, so no data are lost during these steps.

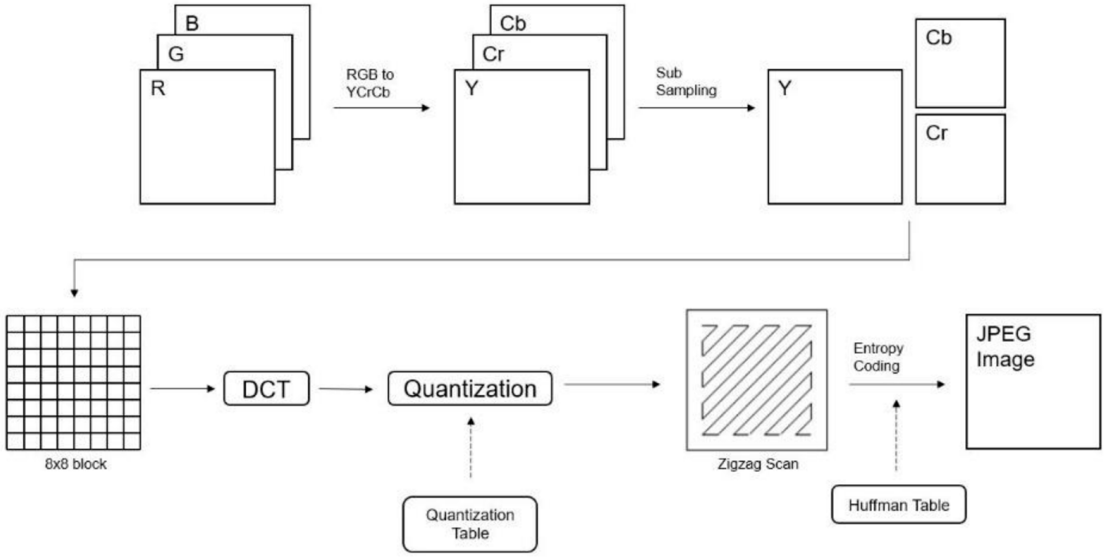


Figure 1. Overview of JPEG image compression.

The color space of the image is converted from RGB to YCbCr. The Y component has brightness information, and Cr and Cb contain color difference information.

The sub-sampling or down-sampling process is a process of reducing the number of Cr and Cb components while maintaining the Y component. Human eyes are more sensitive to brightness information than color information. Therefore, the process leaves the Y component intact, while it compresses Cr and Cb components, which contain color information. Douglas A. Kerr [10] described the sub-sampling process in detail. The following Figure 2 expresses sub-sampling in the ratio of J:x:y, where J refers to the width of the pixel block to be sampled, and samples are extracted from a pixel block (size = 2 * J). x and y refer to the number of samples extracted from the first rows of J pixels and from their second rows, respectively.

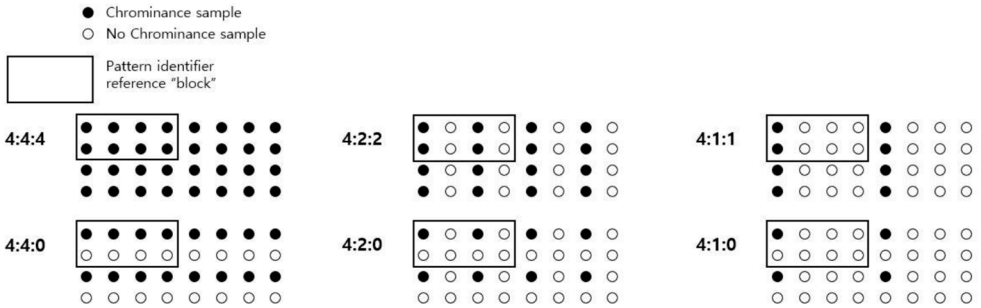


Figure 2. Chrominance subsampling patterns.

As shown in Figure 2, there are several sub-sampling patterns. Figure 3 shows the shape of the pixel block after conducting sub-sampling according to the pattern above. As can be seen in the figures, 4:4:4 indicates that all four components are extracted from the first and second rows, which means that sub-sampling is not carried out.

The most commonly used ratio in JPEG compression is 4:2:0, which means extracting 2 and 0 from the first row and the second row, respectively. Since it discards 6 values out of 8-pixel values, it can reduce the size by that much. After sub-sampling, each channel of Y, Cr, and Cb is divided into 8*8 blocks and the following process is performed for each channel.

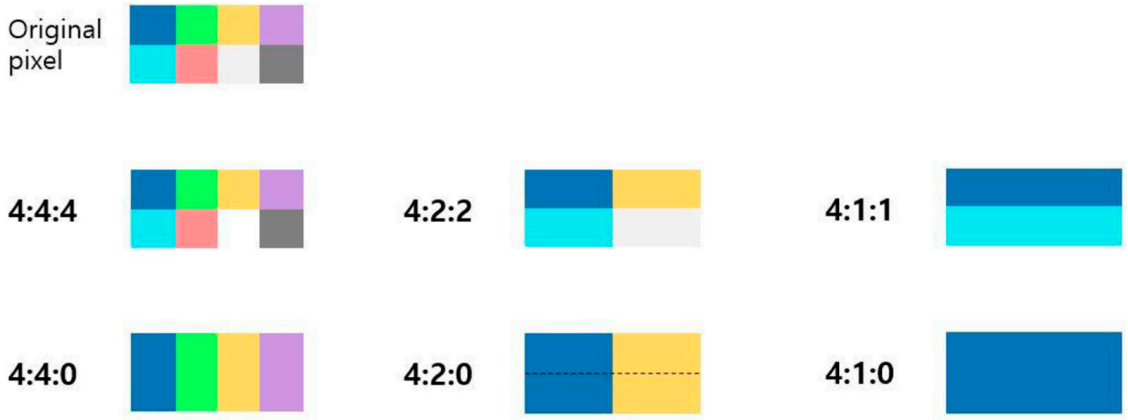


Figure 3. Pixel blocks after conducting sub-sampling.

DCT is an algorithm proposed by N. Ahmed, T. Natarajan, and K. R. Rao [11], and it is used to transform an image from the spatial domain to the frequency domain. Since the conversion result values are always real numbers, they are easy to process, and DCT is widely used in signal processing and image processing. JPEG converts an 8*8 block composed of values in the range of [0, 255] to the range of [−128, 127] and applies DCT to each block. The DCT conversion formula is as follows.

$$G_{u,v} = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} I_{x,y}\cos\frac{2x+1}{16}\pi\cos\frac{2y+1}{16}\pi \quad (1)$$

G stands for the transformed frequency matrix, while G_u and G_v mean the (u, v)th element of G. I refers to the matrix of the image block before conversion, and I_x and I_y mean the (x, y)th element of I.

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}}, & if\ k = 0 \\ 1, & otherwise \end{cases} \quad (2)$$

The DC coefficient refers to the value of (0, 0)th element that is firstly obtained from the block converted by DCT. The other 63 values are called the AC coefficients. The DC coefficient has important information that determines the brightness of the entire block.

The next step is quantization. This is the step used to remove the high-frequency component of the block obtained in the DCT process. Quantization is performed by applying the quantization table to each element. At this time, the quantization table must be input to the encoder by the user or the application performing JPEG compression. The quantization table has integer values between 1 and 255, and the goal is to remove the high-frequency component, which is not visually important, through quantization [9].

The quantization process rounds up the value obtained by dividing each DCT element by the quantization table. As a result, due to the structure of the quantization table, DC coefficients are divided into smaller values, and AC coefficients are divided by a larger value when they are located closer to the lower right. Consequently, many AC coefficients become zero, resulting in the loss of high-frequency components. More details can be found in [12].

Zigzag scanning is performed as the next step. This step converts the quantized DCT coefficients in the form of a two-dimensional array into a one-dimensional data form. The values are scanned in the order and the values are stored in a one-dimensional form.

The next step is entropy coding. Entropy encoding is an encoding method that shortens the length of a code representing a symbol according to the occurrence frequency of the symbol. First, it encodes quantized DC coefficients and AC coefficients in different ways. The DC coefficient is coded through the DPCM (Differential Pulse Code Modulation) method, which calculates the difference between the DC coefficient of the previous block

and that of the current block. Run-length encoding is applied to AC coefficients. Run-length encoding expresses data appearing continuously with the same value only with the number of them and the repeated value. Since most AC coefficients are 0 in the JPEG compression, run-length coding is used for 0, which can drastically compress the size of the data. Lastly, Huffman encoding is performed on the coded DC and AC coefficients.

Huffman coding is an algorithm proposed by David A. Huffman [13] and is one of the entropy coding types used for lossless compression. This method encodes items with different lengths according to the occurrence frequency of a data symbol. Huffman encoding uses a longer code for an item with a lower occurrence frequency and a shorter code for an item with a higher occurrence frequency. For this reason, it is inefficient to use data containing nonrepetitive values. However, since JPEG compression performs quantization to make most of the high-frequency components 0, it is suitable for conducting Huffman encoding. The Huffman algorithm creates a prefix by producing a binary tree, and it is made according to the following rules.

- Initialization: List all symbols according to their occurrence frequencies.
- Repeat the following steps until only one symbol remains.
- The parent node is produced by selecting two items with the least occurrence frequencies from the list as child nodes. Here, the value of the parent node is the sum of the child nodes.
- When creating a parent node, a weight of 0 is assigned to the edge of a node with a larger value, and a weight of 1 is assigned to the edge of a node with a smaller value.
- The child node with a larger value is placed on the left.

Figures 4 and 5 show how Huffman encoding is performed according to the above rules, and Table 1 presents the encoded and changed value of each symbol through Huffman encoding. If each item between A and F takes 1 byte, it expresses 1064 bits of data using 309 bits, which shows that the data compression rate is approximately 70%.

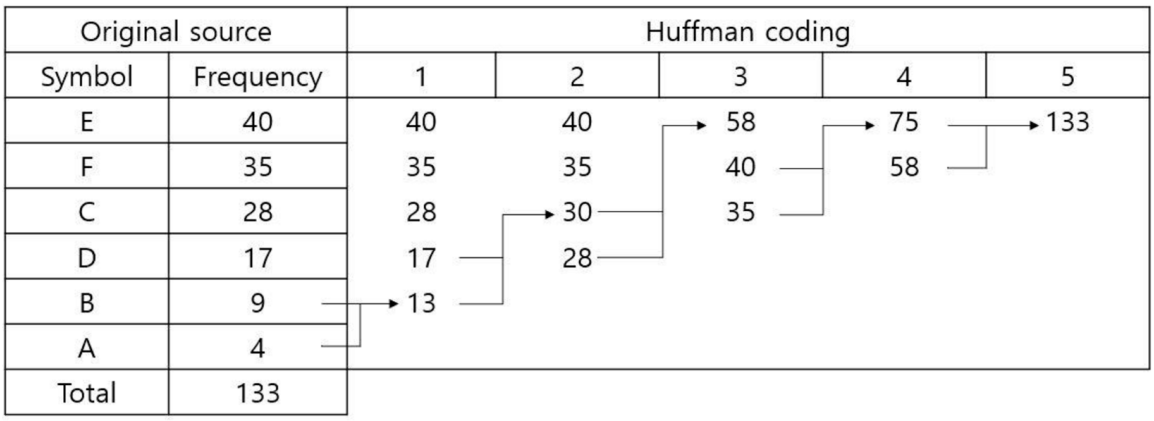


Figure 4. Huffman encoding process.

DC coefficients and AC coefficients that have completed DPCM and run-length encoding, respectively, go through Huffman encoding using different Huffman code tables in the JPEG compression process.

The process of decompressing a JPEG file is basically a reverse order of the compression process. However, it is impossible to restore the part which has lost data in the downsampling process. Moreover, although the matrix in the frequency domain can be reversed to the original image through IDCT (Inverse Discrete Cosine Transform) in the DCT process, it cannot be fully (100%) restored either, because the high-frequency components are removed through the quantization process. In the case of the zigzag scanning and Huffman encoding processes, data are restored by rearranging the stored number of zeros and each element and going through the Huffman decoding process. Data are not lost during these processes.

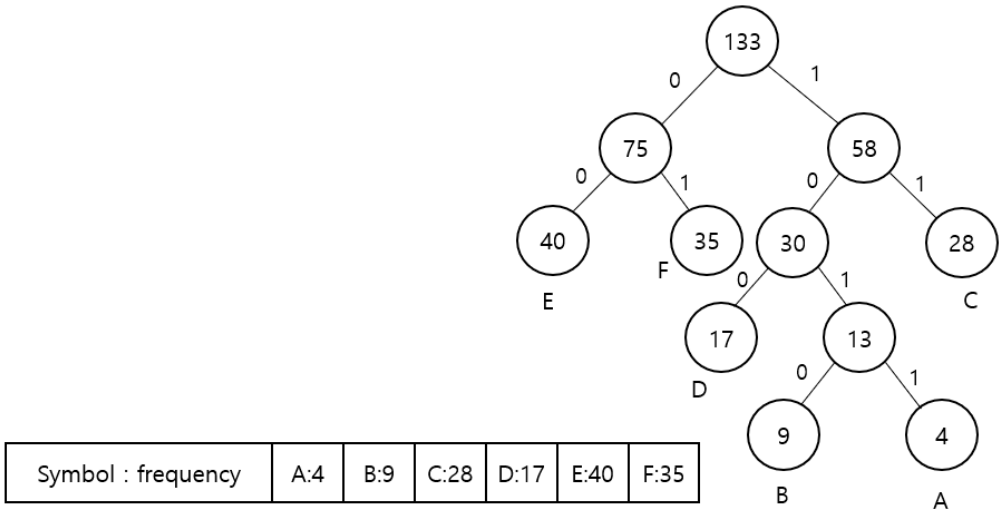


Figure 5. Huffman tree.

Table 1. Huffman encoding result.

Frequency	Symbol	After Coding
40	E	00
35	F	01
28	C	11
17	D	100
9	B	1010
4	A	1011

2.3. JPEG Header

This section reviews the overall structure of a JPEG image created through the JPEG compression process discussed earlier. JPEG images are basically saved in the format of JFIF (JPEG Interchange Format) [14], and the extension is ".JPG" or ".JPEG". JFIF is a minimal format made to be compatible with multiple platforms and applications. There are various extension formats depending on platforms and applications. In some cases, each format additionally includes metadata about the image. The following Figure 6 shows the overall structure of the JFIF file format.

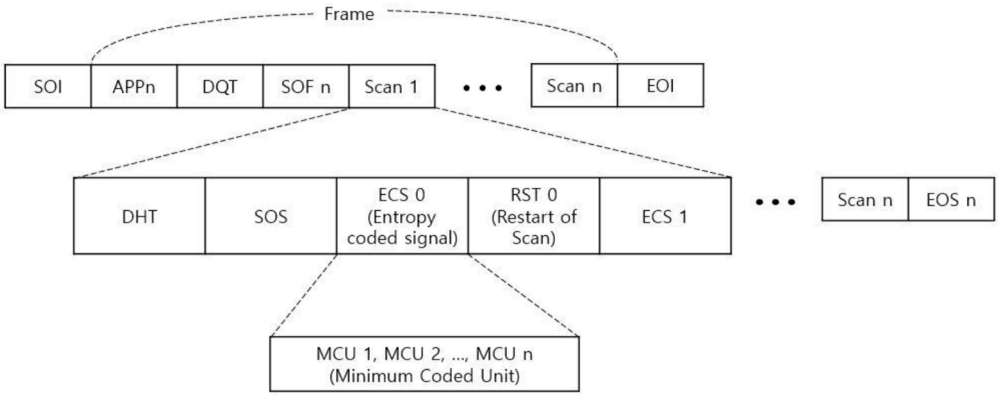


Figure 6. JPEG file structure.

A JPEG file consists of several segments, and each segment is separated by a marker. These markers start with 0xFF, and the type of a marker is identified by the 1 byte immediately after it. If a segment has a variable length, it has payload data defining the

length of the segment after the marker. The following table summarizes commonly used JPEG markers. Further information can be found in the JPEG: still image data compression standard [15].

As shown in Figure 6, the data of the ECS (Entropy Coded Signal) segment is encoded through the Huffman table defined in the DHT segment, and quantization is performed through the quantization table defined in the DQT (Define Quantization Table) segment. If the values of the two tables change, the shape of the image will be different from that of the original image. Therefore, we assessed whether directly encrypting these two tables is suitable for achieving de-identification.

Quantization, as a process, is performed to remove high-frequency components after performing the DCT process. A JPEG header generally has two tables in total, which contain brightness and color difference information, respectively, and they are stored in the DQT segment. The following shows an image after quantization table encryption.

In Figure 7a–d indicate the original image, the quantization table with brightness information, the quantization table with color difference information, and an image of both encrypted quantization tables, respectively. When encrypting the DQT segment, since the low-frequency components are restored through the encrypted table (the high-frequency components cannot be restored because they are 0 and are removed), they have different values from the original values. Therefore, b has a different brightness value with the same color difference value as the original image. Contrarily, c has a different color difference value without changing the brightness of the image.

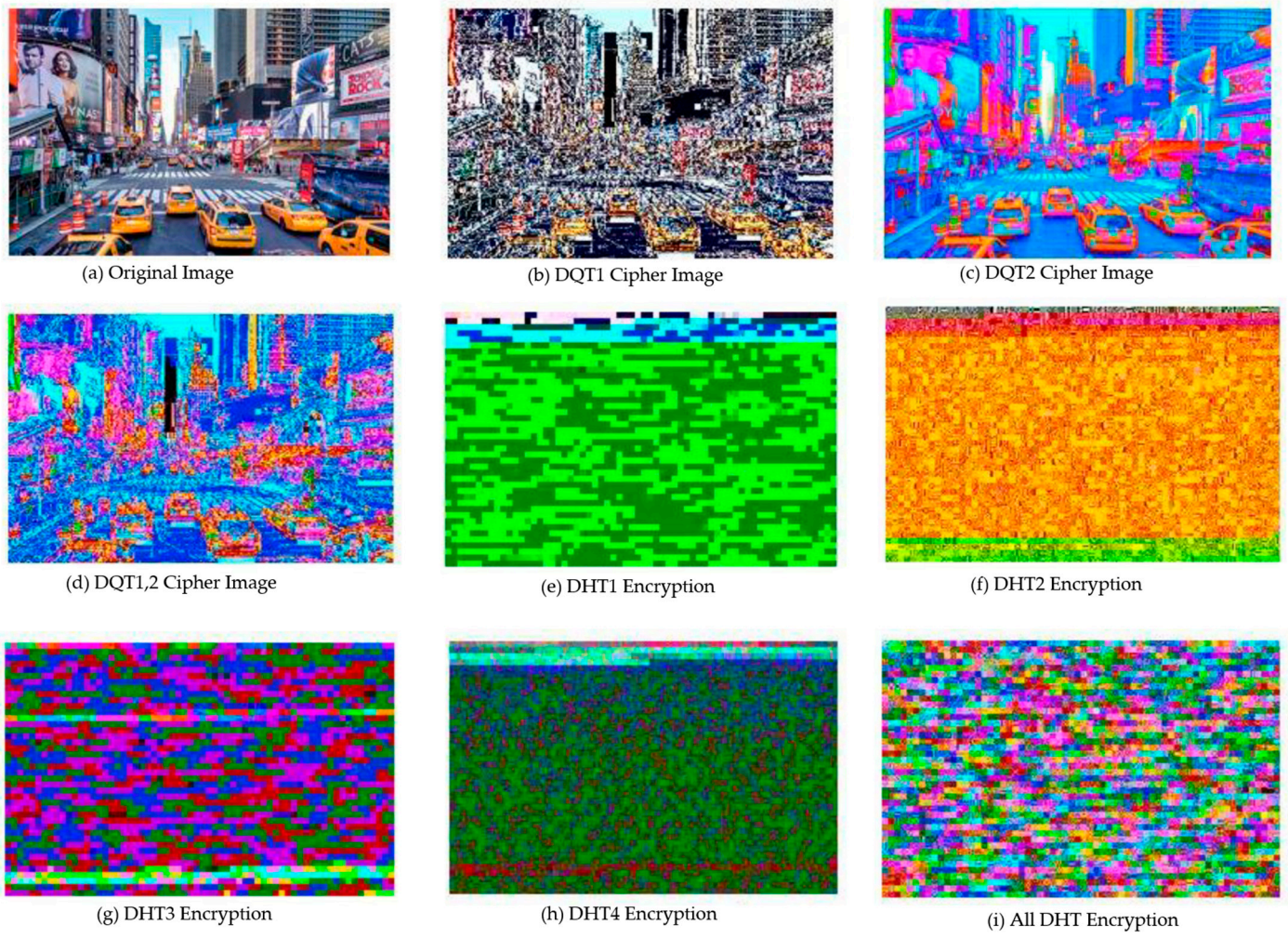


Figure 7. DQT and DHT encoding.

Huffman encoding is a process of compressing image information after conducting a data loss process. If the values of the Huffman table change, all data constituting the image change. It is necessary to understand the structure of the DHT segment for encrypting the Huffman table.

The Huffman table uses a total of four tables (two tables for DC coefficients and two tables for AC coefficients): Y-DC, Y-AC, CbCr-DC, and CbCr-AC tables, respectively [16]. Each table starts with a marker indicating the DHT segment and consists of a field that organizes the length of the DHT segment, table classification information, and the number of symbols constituting the Huffman table and a field that organizes actual symbols. The rest, except for the symbol field, has a fixed size, and the symbol field has a variable length. Figure 7 depicts the encryption of a Huffman table by using the method proposed by this study with reference to the structure of DHT.

The result has completely different values from the original values in all processes (inverse quantization-IDCT-color space conversion), which are the processes of decompressing a JPEG. Therefore, even if only one table is encrypted, the original image cannot be identified. However, since there is a risk of discovering the original values of the encrypted table by referring to the unencrypted table, this study encrypted all Huffman tables.

We choose not to encrypt DQT due to the possibility of extracting information about the original image from the encrypted image, as can be seen in Figure 7b–d. Therefore, DHT encryption was used in this paper, since the original image cannot be identified, as can be seen in Figure 7e–i.

Since it would be possible to deduce the information of the original image from the encrypted image (Figure 7), this study judged that it would be inappropriate to completely de-identify data by encrypting DQT. In the case of DHT encryption, the original image cannot be identified, so this study implemented a model that encrypted DHT to de-identify images.

The proposed method does not assume that one cannot recover the Huffman tables from the compressed data. Our proposed method may not offer perfect security for image security. Given the large amount of data transmitted and received on the Internet, we believe that this method can effectively delay hacking attempts until a desired point in time.

3. Implementation

In this section, we explain the Huffman table in the JPEG header and implement a model that encrypts and de-identifies images. We describe the overall flow of the implemented model, and then we show the results and how to encrypt and decrypt the tables.

3.1. Experiment Environment

The environment used in the experiment is as follows:

- Eclipse IDE 2021-03, JDK 11.0.5
- Intel i7-8700 CPU, 32GB Ram Desktop PC
- Windows 10 professional 64bit
- HxD [17] tool: for read and write binary data of images

3.2. Overall Structure

The following shows the overall flow of the model for de-identifying an image using the DHT encryption process.

- Find the DHT segment by locating the DHT marker in the read JPEG image.
- Check the classification information in the table. Then, if it is a DC table, an algorithm for encrypting the DC table is performed, and if it is an AC table, an algorithm for encrypting the AC table is used.
- After encrypting all four tables following the procedure described above, print the DHT-encrypted JPEG image.

Figures 8 and 9 show the process of encrypting the AC tables and DC tables. An AC table can be encrypted with a general encryption method. However, since the value divided by the DC component is relatively small [18], as can be confirmed in the quantization process, the symbol value of a DC table does not exceed 0x0F. Therefore, it must be encrypted using a method different from that for encrypting an AC table. More precisely, encrypting the DC values will yield a value greater than 0x0F. If the symbol value exceeds 0x0F within the DC values, it will lead to image corruption. Since our objective is to de-identify images without compromising the functionality of the images, we mitigated this risk by expanding the resulting ciphertext. Concretely, since the symbol values of a DC table range between 0x00 and 0x0F, this step was implemented by expanding each byte of the encrypted data in the form of 0x0L 0x0H. The following Figure 9 shows the process of encrypting the AC tables and DC tables.

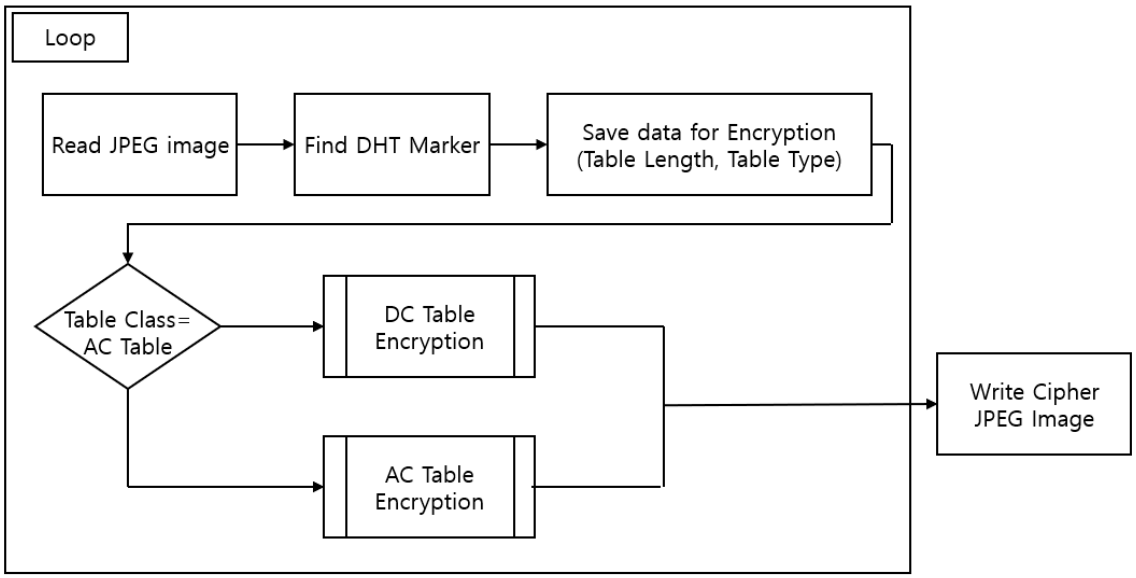


Figure 8. DHT encryption process.

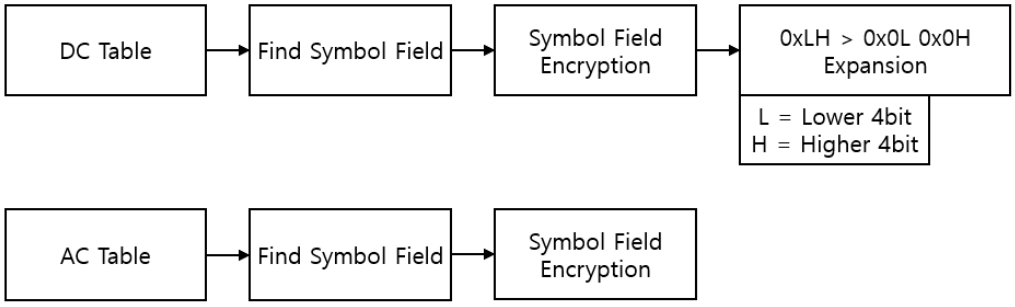


Figure 9. Encryption process of DC tables and AC tables.

Since the symbol values of a DC table range between 0x00 and 0x0F, encryption was implemented by expanding each byte of the encrypted data in the form of 0x0L 0x0H. This paper implemented the de-identification of the image by encrypting the data of the symbol field. The following Tables 2 and 3 compare the data field before DHT encryption and after DHT encryption using the method proposed by this study.

Table 2. Comparison of DHT encryption (before).

Field	Marker	DHT Length	Table Class	Table Number	Huffman Code	Symbol
Length (Byte)	2	2	0.5	0.5	16	No fix Length
Data	0xFF 0xC4	-	0 or 1	0 or 1	...	Original Data

Table 3. Comparison of DHT Encryption (after).

Field	Encrypted Data Length	Marker	DHT Length	Table Class	Table Number	Huffman Code	Symbol
Length (Byte)	1	2	2	0.5	0.5	16	No fix Length
Data		0xFF 0xC4	-	0 or 1	0 or 1	...	Encrypted Data

It is necessary to know the size of the encrypted data to decrypt it back to the original data, because the size of the encrypted data can be equal to or greater than that of the original data. Therefore, this study's process recorded the size of the encrypted data in front of the DHT marker so that it could be referenced when decrypting.

The process of decrypting the encrypted Huffman table is as follows.

- Find the DHT segment by locating the DHT marker in the encrypted JPEG image.
- Perform a decryption algorithm suitable for each table after checking the size information of the encrypted data in front of the DHT marker and the table classification information. Information containing the length of the encrypted data is deleted.
- The four tables are decrypted in the same way as above, and the image is printed.

The Figure 10 shows the flow of the decryption process. The decryption process for the DC tables is different from that of the AC tables. The following Figure 11 shows the decryption process of each table.

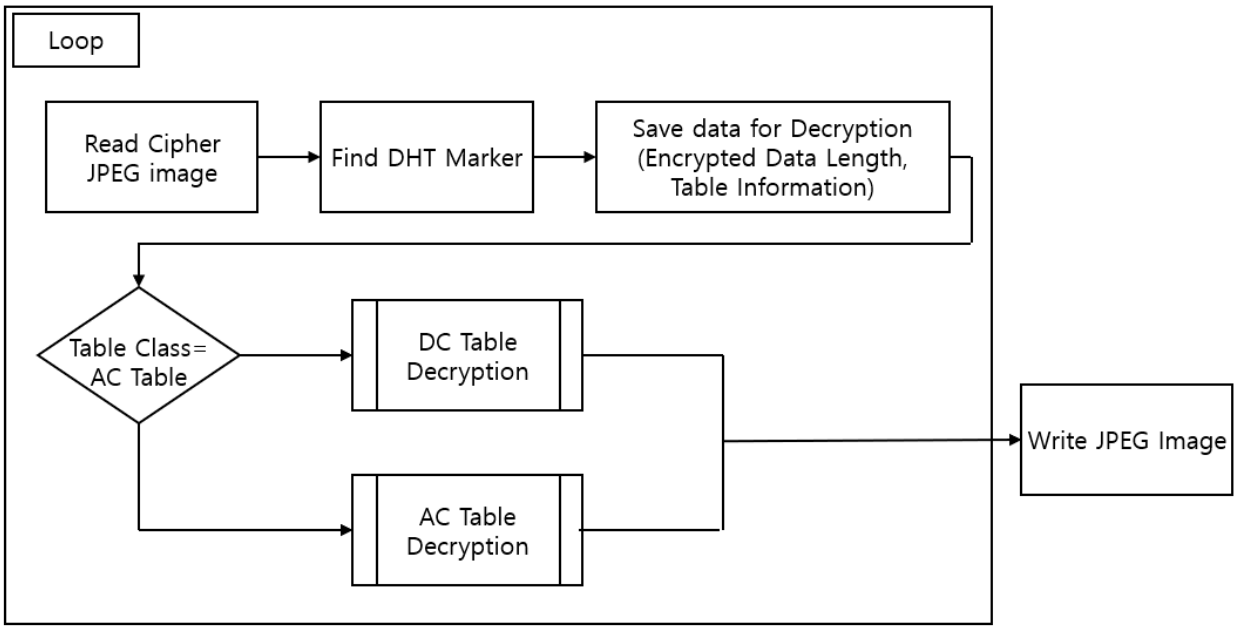


Figure 10. DHT decryption process.

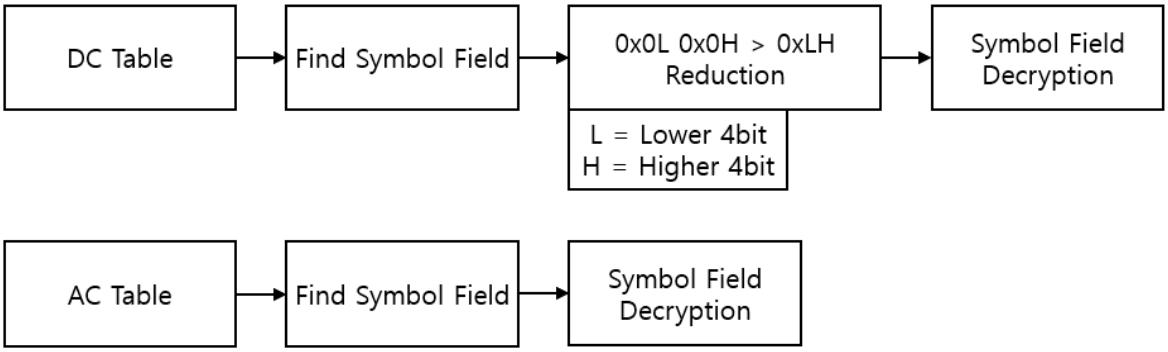


Figure 11. Decryption process of DC tables and AC tables.

Since the symbol of an encrypted DC table is expanded from 1 byte to 2 bytes during the encryption process, it is decrypted while reducing the expanded data back to 0xLH form (1 byte).

Figure 12 shows part of the encryption and decryption codes. These codes are programmed in the Java language. These encryption/decryption codes show our proposed methods.

```java
public byte[] encryptFucDCt(int index, byte[] imageBytes) throws Exception {
    Aes256 aes256 = new Aes256();

    if (index == 1 || index == 3) {
        System.out.println("error!!");
        return imageBytes;
    }

    int[] DHTMarker = this.findIndex(imageBytes);
    int tableLen = (imageBytes[DHTMarker[index] + 3] & 0xFF);
    int startTargetBytes = DHTMarker[index] + 21;
    int endTargetBytes = DHTMarker[index] + tableLen + 2;

    // Encryption
    byte[] originalBytes = Arrays.copyOfRange(imageBytes, startTargetBytes, endTargetBytes);
    byte[] encryptBytes = aes256.encrypt(originalBytes);

    // Extended encryption bytes
    byte[] extendBytes = new byte[encryptBytes.length * 2];
    for (int i = 0; i < encryptBytes.length; i++) {
        extendBytes[i * 2] = (byte) (((encryptBytes[i] & 0xF0)) >> 4);
        extendBytes[(i * 2) + 1] = (byte) ((encryptBytes[i] & 0x0F));
    }

    // Write encrypted Image
    int addedLen = extendBytes.length - originalBytes.length;
    int tempLength = imageBytes.length + addedLen + 1;
    byte[] temp = new byte[tempLength];

    for (int i = 0, j = 0; i < temp.length; i++) {
        if (i < DHTMarker[index]) {
            temp[i] = imageBytes[i];
        } else if (i == DHTMarker[index]) {
            temp[i] = (byte) extendBytes.length;
        } else if (i > DHTMarker[index] && i < startTargetBytes + 1) {
            temp[i] = imageBytes[i - 1];
        } else if (i >= startTargetBytes + 1 && i < startTargetBytes + extendBytes.length + 1) {
            temp[i] = extendBytes[j];
            j++;
        } else {
            temp[i] = imageBytes[i - addedLen - 1];
        }
    }

    return temp;
}
```

(a) art of DC table Encryption Code

```java
public byte[] decryptFucDCt(int index, byte[] encImgBytes) throws Exception {
    Aes256 aes256 = new Aes256();

    if (index == 1 || index == 3) {
        System.out.println("error!!");
        return encImgBytes;
    }

    int[] DHTMarker = this.findIndex(encImgBytes);
    int startTargetBytes = DHTMarker[index] + 21;
    int encBytesLen = (encImgBytes[DHTMarker[index] - 1] & 0xFF);
    int endTargetBytes = startTargetBytes + encBytesLen;

    // Reduction bytes
    byte[] encBytes = Arrays.copyOfRange(encImgBytes, startTargetBytes, endTargetBytes);
    byte[] contractBytes = new byte[encBytesLen / 2];
    for (int i = 0; i < contractBytes.length; i++) {
        contractBytes[i] = (byte) ((encBytes[i * 2] << 4) | encBytes[(i * 2) + 1]);
    }

    // Decryption
    byte[] decBytes = aes256.decrypt(contractBytes);

    int deletedLen = encBytesLen - decBytes.length;
    int tempLen = encImgBytes.length - encBytesLen + decBytes.length - 1;
    byte[] temp = new byte[tempLen];

    for (int i = 0, j = 0; i < temp.length; i++) {
        if (i < DHTMarker[index] - 1) {
            temp[i] = encImgBytes[i];
        } else if (i >= DHTMarker[index] - 1 && i < startTargetBytes - 1) {
            temp[i] = encImgBytes[i + 1];
        } else if (i >= startTargetBytes - 1 && i < startTargetBytes + decBytes.length - 1) {
            temp[i] = decBytes[j];
            j++;
        } else {
            temp[i] = encImgBytes[i + deletedLen + 1];
        }
    }

    return temp;
}
```

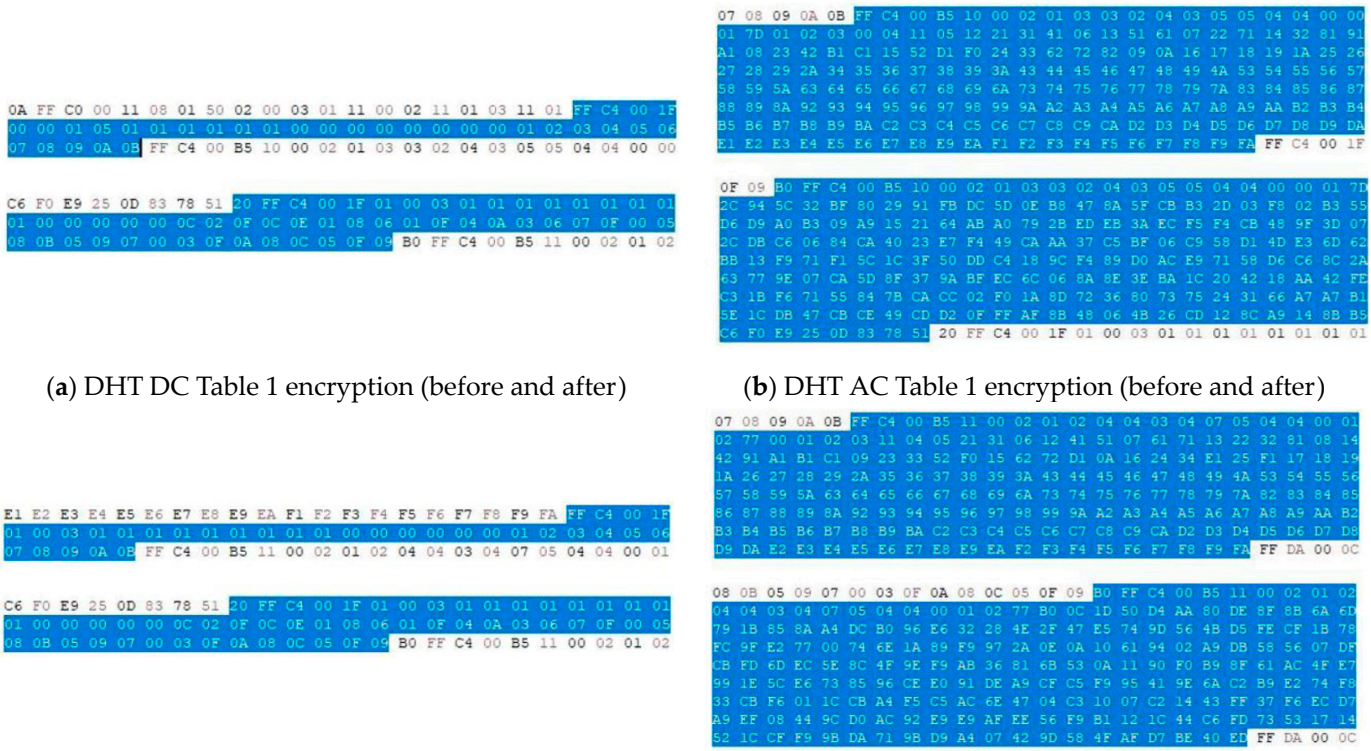
(b) Part of DC table Decryption Code

Figure 12. Part of code for encryption/decryption.

3.3. Result

First, this study compared the Huffman table before encryption and that after encryption by using a program that read images as binary data. Afterward, the encrypted image was examined. The following Figure 13 shows the comparison of each Huffman table before and after encryption.

Figure 14 shows the images after encrypting each table. It is difficult to guess the original image from the encrypted image. In addition, Figure 15 confirms the encryption and decryption results of other images.



(**a**) DHT DC Table 1 encryption (before and after)

(**b**) DHT AC Table 1 encryption (before and after)

(**c**) DHT DC Table 2 encryption (before and before)

(**d**) DHT AC Table 2 encryption (before and after)

Figure 13. Comparison of tables' encryption (before and after).



Figure 14. DHT image compression.

The speed of encryption and that of decryption were calculated by averaging 10,000 runs of the algorithms presented by this study after reading each image. In addition, the following Tables 4 and 5 summarize and compare the sizes of image files after encryption.

Table 4. Speed of encryption and decryption.

JPEG Image Size (Byte)	Encryption Speed (ms)	Decryption Speed (ms)	Encrypted Image Size (Byte)
A (202,692)	2.148	2.207	202,764
B (363,325)	1.301	1.251	363,397
C (399,029)	2.178	2.315	399,097

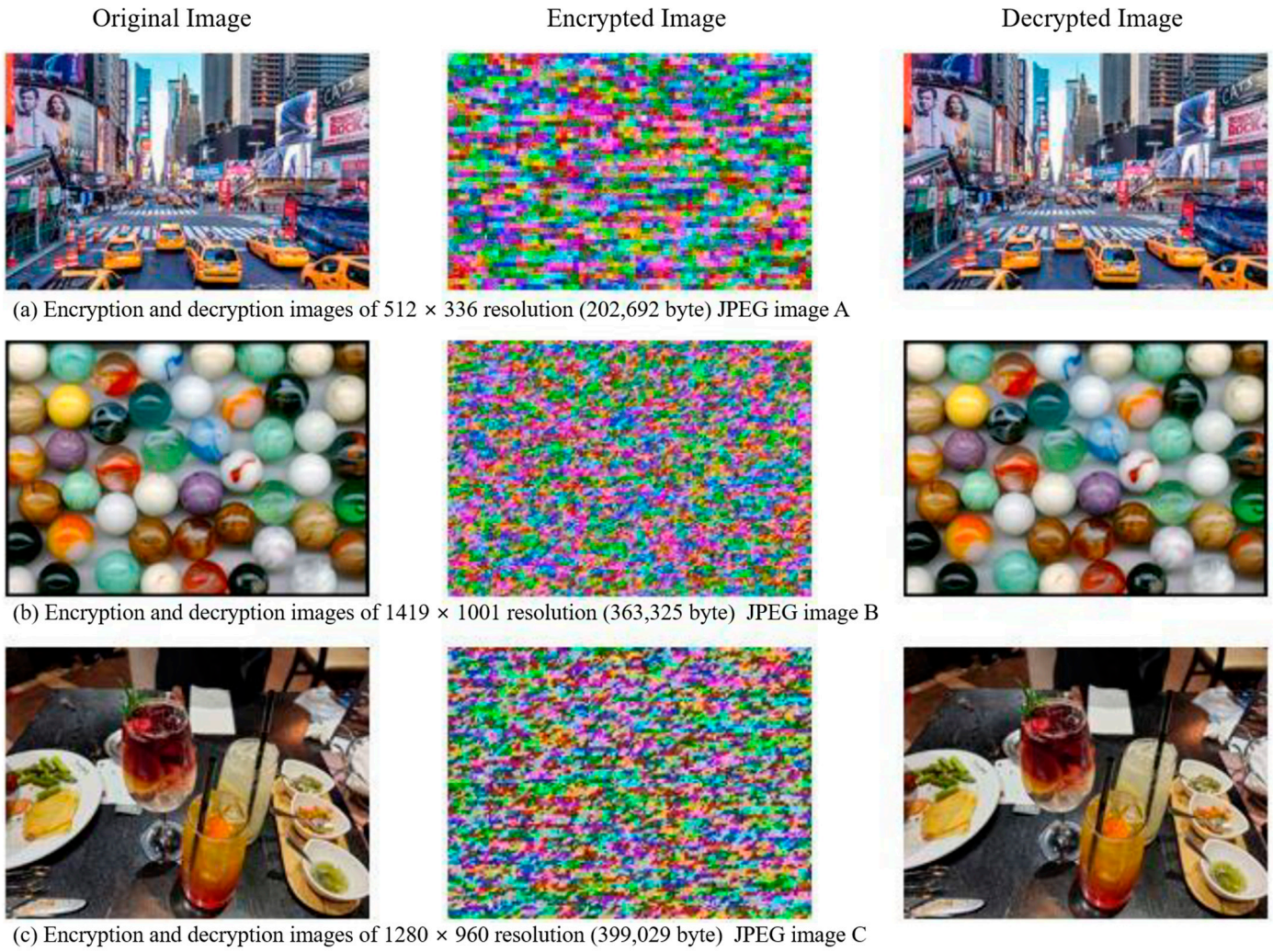


Figure 15. DHT encryption and decryption image results.

Table 5. Comparison with other proposals.

Scheme	Image Size (in Pixel)	Average Encryption
Ref. [2]	256 × 256 200 × 320	31.75 ms 29.25 ms
Ref. [19]	256 × 256 512 × 512 1024 × 1024 2048 × 2048	0.33~0.39 s 0.38~0.40 s 6.26~6.32 s 25.15~25.32 s
Ref. [20]	512 × 512	19.27 ms
Proposed Method	512 × 336 (A) 1419 × 1001 (B) 1280 × 960 (C)	2.148 ms 1.301 ms 2.178 ms

The following Table 5 summarizes the previously studied methods of encrypting images [2] and the encryption times of image encryption using chaotic logistic maps [19]. Furthermore, the method proposed in [20] has excellent security and is based on the Jia chaotic system and a logistic map. Note that the algorithm introduced in [20] prioritizes security over encryption speed. Nevertheless, we can see that is notably slower than our proposed approach.

It was confirmed that encryption and decryption could be conducted quicker than other studies that applied an encryption algorithm to the entire image (Tables 4 and 5). Moreover, the encryption and decryption speed of previous studies tended to be slower when the size of the image was larger (Table 5). However, since this study proposed to encrypt only the Huffman tables, it was confirmed that the speed of encryption was not affected even if the size of the image file increased.

4. Conclusions

Currently, people are becoming more interested in privacy issues. Moreover, in line with this, research on data de-identification is also becoming very important and active. This study evaluated tables that could be encrypted among the segments of the JPEG compression process, JPEG header composition, and JPEG header. The results of this study showed that images could be de-identified by encrypting the data in the DHT segment.

This study implemented an image de-identification model based on the above. The results of this study confirmed that the image could not be identified after encrypting only a minimal portion, compared to previous studies that encrypted the entire image, and also confirmed that the encryption speed and decryption speed were faster and more effective than the results of previous studies. Using this method, it is possible to send and receive data safely by quickly de-identifying important images that should not be exposed. This will also make it difficult for an attacker or intruder to know that the image is important by making it impossible to identify the original image when an intruder is trying to steal image data from an external environment (e.g., a cloud server) or when one is under a snooping attack during Internet communication. Naturally, the user's privacy can be protected. It can be expected to be used in various sectors such as medical services, cloud services, and telecommunications.

However, since only the Huffman table of each JPEG is encrypted, there are still issues to be improved, such as that the scan data of the image is exposed, there is little difference between the tables by image for the DC table of DHT, and it encrypts only the symbol field of the Huffman table. It is planned to study how to encrypt the entire table by extending it to encrypt other fields, except for the field for identifying markers and tables, rather than encrypting only the symbol fields of the Huffman table. It will be also studied whether image file formats other than JPEGs can be de-identified through partial encryption. Our proposed method may not offer perfect security for image security. However. given the large amount of data transmitted and received on the Internet, we believe that this method can effectively delay hacking attempts until a desired point in time. Additionally, we will add experiments to check and supplement the performance compared to recent methods. In addition, the purpose of this paper is to allow encryption to be maintained in real time in scenarios where many image files are transferred. Protecting images permanently seems challenging. Most recent jpeg encryption papers [21,22] encrypt the entire image. Because the goal of this paper is not to target perfect encryption of full jpeg files, it is difficult to directly compare its results with recent studies.

Author Contributions: Conceptualization, H.-S.K., S.C. and S.-R.K.; methodology, H.-S.K., S.C. and S.-R.K.; software, S.C.; validation, H.-S.K. and S.-R.K.; formal analysis, H.-S.K., S.C. and S.-R.K.; investigation, S.C. and H.-S.K.; resources, S.C.; data curation, H.-S.K., S.C. and S.-R.K.; writing—original draft preparation, S.C.; writing—review and editing, H.-S.K. and S.-R.K.; visualization, S.C.; supervision, S.-R.K.; project administration, S.-R.K.; funding acquisition, S.-R.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2016R1D1A1B02011964).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: Author Seongjun Cha is employed by the company INTHESMART (iTS) Corporation. The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Kim, D.-G.; Lee, H. Overview on personal information de-identification based on big data. *Rev. Korean Soc. Internet Inf.* **2015**, *16*, 15–22.
2. Zeghid, M.; Machhout, M.; Khriji, L.; Baganne, A.; Tourki, R. A Modified AES Based Algorithm for Image Encryption. *Int. J. Comput. Sci. Eng.* **2007**, *1*, 745–750.
3. Singh, A.; Agarwal, P.; Chand, M. Image Encryption and Analysis using Dynamic AES. In Proceedings of the IEEE International Conference on Optimization and Applications (ICOA), Kenitra, Morocco, 25–26 April 2019.
4. Daemen, J.; Rijmen, V. *The Design pf Rijndael: The Advanced Encryption Standard (AES)*; Springer: Berlin/Heidelberg, Germany, 2002.
5. Dworkin, M.J.; Barker, E.B.; Nechvatal, J.R.; Foti, J.; Bassham, L.E.; Roback, E.; Dray, J.F., Jr. *Advanced Encryption Standard (AES)*; Federal Inf. Process. Stds. (NIST FIPS)—197; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2001.
6. Ehrsam, W.F.; Meyer, C.H.; Smith, J.L.; Tuchman, W.L. Message Verification and Transmission Error Detection by Block Chaining. U.S. Patent US4074066A, 14 February 1978.
7. Joint Photographic Experts Group. JPEG Homepage. Available online: https://jpeg.org/jpeg/index.html (accessed on 10 November 2021).
8. *ISO/IEC 10918-1*; Information Technology—Digital Compression and Coding of Continuous-Tone Still Images: Requirements and Guidelines. ISO: Geneva, Switzerland, 1994.
9. Wallace, G.K. The JPEG still picture compression standard. *IEEE Trans. Consum. Electron.* **1992**, *38*, 18–34. [CrossRef]
10. Kerr, D.A. Chrominance Subsampling in Digital Images. 2012. Available online: http://www.dougkerr.net/Pumpkin/articles/Subsampling.pdf (accessed on 9 November 2021).
11. Ahmed, N.; Rao, K.R. Discrete Cosine Transform. *IEEE Trans. Comput.* **1974**, *C-23*, 90–93. [CrossRef]
12. Sherlock, B.G. A Model for JPEG Quantization. In Proceedings of the ICSIPNN '94. International Conference on Speech, Image Processing and Neural Networks, Hong Kong, China, 13–16 April 1994.
13. Huffman, D.A. A method for the construction of minimum redundancy codes. *Proc. IRE* **1952**, *40*, 1098–1101. [CrossRef]
14. *Information Technology—Digital Compression and Coding of Continuous-Tone Still Images: JPEG File Interchange Format (JFIF)*; IT-T.871; ITU-T: Geneva, Switzerland, 2011.
15. Pennebaker, W.B.; Mitchell, J.L. *JPEG: Still Image Data Compression Standard*, 1993rd ed.; Springer: Berlin/Heidelberg, Germany, 1993.
16. Razak, S.A. Analysis of DQT and DHT in JPEG Files. *IJITCS* **2013**, *10*, 1–11.
17. Hörz, M. Hexeditor (HxD) Ver. 2.5.0.0. 2021. Available online: https://mh-nexus.de/en/hxd/ (accessed on 1 August 2022).
18. Wu, S.W.; Gersho, A. Rate-constrained picture-adaptive quantization for JPEG baseline coders. In Proceedings of the 1993 IEEE International Conference on Acoustics, Speech, and Signal Processing, Minneapolis, MN, USA, 27–30 April 1993.
19. Pareek, N.K. Image encryption using chaotic logistic map. *Sci. Image Vis. Comput.* **2006**, *24*, 926–934. [CrossRef]
20. Peng, Y.; Fu, C.; Cao, G.; Song, W.; Chen, J.; Sham, C.-W. JPEG-compatible Joint Image Compression and Encryption Algorithm with File Size Preservation. *ACM Trans. Multimed. Comput. Commun. Appl.* **2024**, *20*, 1–20. [CrossRef]
21. Zhu, S.; Deng, X.; Zhang, W.; Zhu, C. Secure image encryption scheme based on a new robust chaotic map and strong S-box. *Math. Comput. Simul.* **2023**, *207*, 323–346. [CrossRef]
22. Yuan, Y.; He, H.; Mao, Y.Y.N.; Chen, F.; Ali, M. JPEG image encryption with grouping coefficients based on entropy coding. *J. Vis. Commun. Image Represent.* **2023**, *97*, 103975. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.