INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2018, VOL. 64, NO. 2, PP. 151–158
Manuscript received September 30, 2017; revised March, 2018. DOI: 10.24425/119363

PixSel: Images as Book Cipher Keys

An Efficient Implementation Using Partial Homophonic Substitution Ciphers

Mykel Shumay and Gautam Srivastava

***Abstract*—In this paper we introduce a novel encryption technique, which we call PixSel. This technique uses images in place of literature as the book cipher's key. Conventional book ciphers possess an unwieldy ciphertext enlargement, creating ciphertexts multiple times the length of the plaintext. As well, there is often the issue of a given book not containing the necessary material for the encipherment of some plaintexts. We sought to rectify these nuisances with PixSel, possessing a typical ciphertext enlargement of merely 1% to 20% for text. Using PixSel, there are also no limitations on encipherable data type, given a suitable image.**

***Keywords*—book ciphers, ciphertext enlargement, ciphertext expansion**

I. Introduction

In this paper, we present a novel take on book ciphers, namely the use of images in place of books as the key; we call this method **PixSel**. Given a workable image, **PixSel** may be used to encrypt any plaintext, without worry of what the plaintext is comprised of.

This paper is organized as follows: The rest of this Section briefly introduces the foundations of homophonic substitution and book ciphers. Section II is dedicated to definitions and algorithms, while Section III covers the experimental design. Section IV discusses the main results and **PixSel**'s characteristics, and the paper concludes in Section V.

A. Homophonic Substitution Cipher

Homophonic substitution was an early attempt to make Frequency Analysis a less powerful method of cryptanalysis. The basic idea behind homophonic substitution is to allocate more than one letter or symbol to the higher frequency letters, and encode a letter with one of its symbols. For example, you might use 6 different symbols to represent e and t, 2 symbols for m and 1 symbol for z. Clearly, this cipher will require an alphabet of more than 26 letters, as each letter needs at least one ciphertext letter, and many need more than this. The standard way to do this is to include the numbers in the ciphertext alphabet, but you can also use a mixture of uppercase, lowercase, upside-down letters, and artistic symbols.

B. Book Ciphers

A book cipher is a homophonic substitution cipher in which the key used is a book or other piece of text. Traditionally, book ciphers work by replacing words in a plaintext with the location of matching words from a book, giving a page, line, and which word along that line. It is essential that all participants have the same edition of the same book.

Problems arise if part of the plaintext does not appear in the book, in which case it cannot be encoded. An alternative approach which gets around this problem is to replace individual letters rather than words. This approach was originally used by George Scovell for the Duke of Wellington's army in some campaigns of the Peninsular War (approximately 1807-1814).

If implemented and used carefully, a book cipher may be very strong as it acts as a homophonic cipher with an extremely large number of equivalents [1]. Although, even with this potential security, the large ciphertext enlargement makes this method largely undesirable, and the needed circulation of the same book to all participants is quite inconvenient.

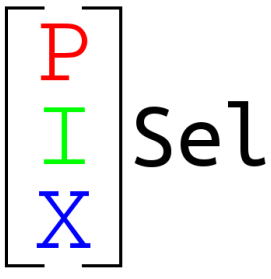
M. Shumay and G. Srivastava are with the Department of Mathematics and Computer Science, Brandon University, Brandon, Canada (email: {shumaymj55, srivastavag}@brandonu.ca).

C. Related Works

In recent years, we have seen some innovative links to the age-old origins of book ciphers. Although not directly connected to the work presented here, they can help motivate and attract what we have accomplished with **PixSel**.

In [2], they focus on the fact that the most efficient attack on symmetric-key systems is an exhaustive key search. They present a novel encryption method with infinite key space by modifying traditional book ciphers. Infinite key space for them can mean an unbounded entropy of the key space, which can frustrate any exhaustive key search. Their approach is the closest found to ours, using multiple common system files as their key, although they were not able to tackle the issue of ciphertext enlargement.

In [3], the authors re-innovate book ciphers by removing some constraints that made them obsolete in the first place. They reduce the size of the formed ciphertext with LZF compression after encoding, and create an encrypted key. Their system also provides an Infinite Key Space similar to [2]. To introduce another level of security, they further hash the encrypted key, and introduce an *Auto-Destruct* functionality to the ciphertext.



We looked to build on the ageless concept of book ciphers with **PixSel**, efficiently using JPEG or PNG images as book cipher keys. At its core, **PixSel** is a novel approach at book ciphers.

II. Algorithm

In this Section we discuss the source of information an image possesses, and how **PixSel** makes use of it.

A. Colour Channels (CC's)

Most JPEG or PNG images are comprised mainly of 3 **Colour Channels** (CC's), these being **RED**, **GREEN**, and **BLUE** (displayed Figure 1). Some images may also have an **ALPHA** (opacity) channel, but its inclusion is uncommon. The colour of any one pixel is a combination of the 3 8-bit values within each CC.

PIXEL	0	1	2	3	4	5	6	7
RED	216	220	202	183	167	140	142	143
GREEN	180	189	176	172	172	174	185	189
BLUE	30	42	45	55	56	54	48	45

Fig. 1. An image's pixels represented by their RGB-values, in a linear manner.

Definition 1 (Ciphertext Enlargement): The percentage of ratio $(ciphertextsize - plaintextsize)/(plaintextsize)$ which describes the increase in length when a message is encrypted.

First, we section each image into many blocks of 253 pixels from left-to-right, top-to-bottom; this size allows for every pixel within a block to be referenced with only one byte, while reserving 3 bytes for use as **Marker Bytes** (II-C).

Definition 2 (Block Address): Each block within the image is addressed by its position within all blocks, from the image's top-left corner to the bottom-right corner. For images smaller than 16 megapixels, each block may be referenced with only 2 bytes, else with 3 bytes for images up to 4.2 gigapixels.

Assume that we have a picture with a width of 100 pixels and height of 10 pixels (Figure 2). For the sake of illustration, let us also stretch the picture vertically. This image is now segmented into linear blocks of 253 pixels.

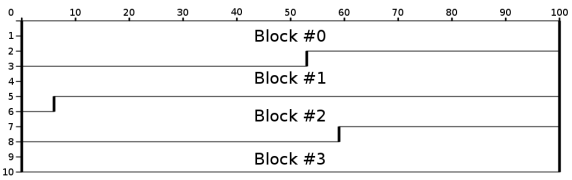


Fig. 2. Block segmentation of an image

To refer to any one pixel, we first refer to the block in which it resides, followed by the pixel's index within that block. Thus, if many consecutively-used pixels happen to be within the same block, it is possible to first refer to the block once, then refer to each individual pixel with only one byte each.

To make use of an image in the same manner as a book in a book cipher, we first need to identify a source of information within the image. This would be the values within each CC. Since each of the 3 CC's hold 8 bits for each pixel, within each pixel there are 3 values from 0 to 255. We disregarded the **ALPHA** channel as its inclusion was detrimental to **PixSel**'s performance due to various characteristics of the channel.

B. Colour Channel (CC) Rotation

The next issue is specifying a pixel's CC for use; instead of explicitly stating the CC, and in turn creating an even larger ciphertext enlargement, the current CC is implicitly defined by the index of the pixel's use within its block's current use. For example, the 0^{th} plaintext byte is searched for within the **RED** channel, the 1^{st} byte is searched for within the **GREEN** channel, and so on (see Figure 3). Upon a change in block, we return to the **RED** channel. For later reference, the **RED** channel may be refered to as the 0^{th} channel, **GREEN** as the 1^{st}, and **BLUE** as the 2^{nd}.

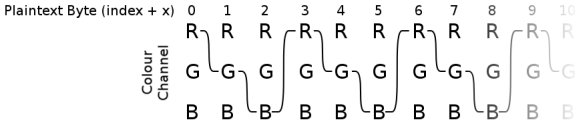


Fig. 3. Current Colour Channel implied by index

C. Marker Bytes

To specify that certain actions are to take place during decryption, 3 bytes were reserved as **Marker Bytes**, as described below.

Since multiple blocks will be made use of, and the same domain of values are used both for block addresses and pixel locations, during decryption one must be made aware of the context of proceeding bytes. For this, each time a new block address is to be defined, a **Block-Termination Marker** of value **255** is placed in the ciphertext.

Instead of strictly sticking to the method of only rotating CC's after each pixel location, it may be beneficial to allow extra rotations between pixel locations; when it is decided to be worth the extra byte, a **Colour Channel Rotation Marker** of value **254** is placed in the ciphertext. When this marker is read in decryption, it signals the CC to rotate to the next available channel.

Finally, it is quite possible to have plaintext bytes that simply are not available within the used image. If this is the case, we print a **Pairing Marker** of value **253**, followed by the locations of two pixels within the previous block whose values add up to the missing value, both within the current CC. Upon reading this marker during decryption, the next two pixels' values are paired together, within the same CC.

D. Block Selection

For a block to be used for n consecutive bytes, with the built-in colour rotation in mind, the 0^{th} plaintext byte must be found in the **RED** (0^{th}) channel, 1^{st} byte in the **GREEN**

(1^{st}) channel, and so on, so that the n^{th} byte is found in the $(n \ mod 3)^{th}$ channel. The use of a block comes to an end once there is no value available within the current channel to correctly encipher the current plaintext byte.

It is very unlikely for any one block to have the ability to encipher the entire plaintext, due to the expected lack of some bytes within each of the CC's. From this, it is clear that each block only has so much potential use at any point in the plaintext, and so the use of many blocks must be combined to encipher the plaintext in its entirety.

The less blocks that are used to encipher a given plaintext, the less block address bytes are required, and so the smaller the ciphertext enlargement will be. Thus, at any point in the plaintext, there is a great emphasis on being efficient with block choice.

Definition 3 ("Optimal" Blocks): For each portion of a plaintext, the block able to encipher the greatest number of consecutive plaintext bytes is picked as the block to choose pixels from for that portion of the plaintext. The address of each optimal block is recorded within the ciphertext when used.

Any block may be used as many times as needed, or not at all. The use of each block, and when it is used, is determined by the encryption algorithm.

E. Pixel Selection

Given some optimal block and the number of bytes that it can encipher, the actual locations of the bytes are chosen at random from all available locations within that block.

F. Augmented Homophonic Substitution Cipher Implementation

Each block acts as 3 separate Homophonic Substitution ciphers, each likely being an entirely different combination of mappings: each block is comprised of three CC's containing the values of 253 pixels, and so has three Homophonic Substitution ciphers of 253 mappings each, with each mapping being to a value from 0 to 255. Since there are less ciphertext "letters" than plaintext values, each block's CC is only a partial cipher.

G. Encryption Steps

Combining everything so far, we create a procedure to encrypt a plaintext through imitation, given some image. The following produces the ciphertext and stores it within the List C.

1) The plaintext's underlying bytes are read and stored.
2) The source image's RGB-values are read and stored.
3) Every block of the image is processed to determine which may encipher the longest consecutive number of plaintext bytes, starting at the index of the next byte to be enciphered. The most optimal block is chosen for use in the next step, for however many consecutive n_0 plaintext bytes it is able to encipher. The optimal block's address is appended to the List C.
    a) For each plaintext byte, a random pixel of the optimal block that holds the correct value is chosen. The CC automatically rotates after each encipherment of a plaintext byte.
    b) If no block is able to encipher any plaintext bytes, find two pixels in the previously-used block that sum to the next plaintext value.
4) Once some n_0 pixel locations within the optimal block are found and recorded in List C, a test is run to determine how manually rotating CC's an extra time at the end of the n_0 bytes will improve the block's usefulness. If this extra n_1 bytes is greater than a set parameter, a **Colour Channel Rotation Marker** is recorded at this mark in C, followed by the n_1 pixel locations. This test is called recursively until the increase in n_k bytes is no longer considered satisfactory.
5) Record a **Block-Termination Marker** at this mark in C. If there is more plaintext to process, go to step 3.
6) Write the List C to a file.

See Appendix A for a pseudocode version.

H. Decryption Steps

All deciphered plaintext values are stored within the List P.

1) The ciphertext is read and stored.
2) The source image's RGB-values are read and stored.
3) The first 2 (or 3, depending on the image size) bytes are read as a block address.
4) Starting at the default CC (**RED**), bytes after this are treated as pixel locations within the given block, and the current CC of that pixel is treated as the plaintext's decrypted data, of which is appended to the List P. The CC rotates after each read pixel location, just as in encryption.
5) If a **Colour Channel Rotation Marker** byte is read, then the CC is rotated once more, and step 4 resumes. If a **Pairing Marker** byte is read, then the current CC's value of the next two pixel locations are summed. Otherwise, if a **Block-Termination Marker** is read, the current block's use has come to an end. Finally, if there is still more ciphertext to process, return to step 3 at the current ciphertext index.
6) Once all of the ciphertext has been processed and looked-up, the resulting plaintext data in List P is written to a file.

See Appendix B for a pseudocode version. The current implementation relies on the receiver to have knowledge of the file type, although this may be automated and stored within the ciphertext with a few extra bytes.

I. Encryption Optimization Techniques

To lower the computational requirements, two simple techniques were implemented to improve the most tasking portions of **PixSel**, discussed below.

1) Elitism: It is clear that most images would likely have a few select portions that are much more suitable than the rest, and so searching every block each time a new block is to be chosen leads to little improvement at great computational cost.

To resolve this, we instead prioritize some small percentage of blocks. These *Elite* blocks are the blocks most likely to

encipher larger amounts of data, as they contain the greatest variety of bytes in all of their CC's. When first looking for the optimal block for a specific segment of plaintext, the most suitable of the *Elite* blocks is chosen. Although, if this optimal *Elite* block does not permit at least a set threshold of bytes, the rest of the image is searched for a more optimal block. This roughly results in $(Elitism\%/100\%)^{\text{th}}$ the amount of computational time required. The ciphertext enlargement efficiency is changed very little, as long as the *Elitism* parameter is set to about 5% or more.

2) Dynamic Programming: It is often the case that a large, complex problem may be solved by first solving many simple, recurring sub-problems, and **PixSel** is able to make use of this idea. With this implemented, performance is improved greatly with only an insignificant increase in memory usage.

J. Greyscale Counterpart

We explored the use of greyscale images for encipherment, which are images that only carry luminous intensity information of the image, or how bright it is at any pixel. Each pixel of an image converted into greyscale would still have its RGB CC's, although each CC would be equal, such that **RED=GREEN=BLUE**. Note that there are other methods of conversion, some of which give differing CC's.

There are two notable characteristics that make greyscale images more effective:

1) Even though the domain of values present within any pixel's CC is equivalent, all 3 CC's of any pixel hold the same value, making the usefulness of an image much less dependent upon the variety of values within each CC, and rather dependent upon the variety overall.
2) Within a small region of an image, it seems more likely that there will be large variations of brightness, rather than a large variation of colour. The usability of a greyscale counterpart is enhanced by this property, and so each block is likely to have a greater variance of values within. This variety is now spread amongst all CC's, reducing the chance to become stuck in a particular CC during a block's use.

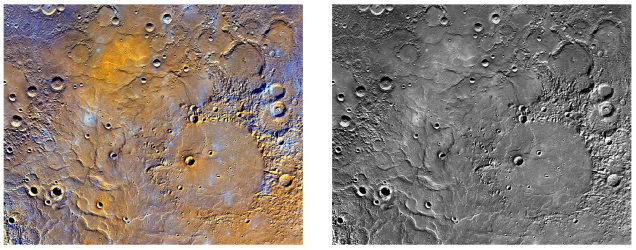
From this, it is clear that the use of a greyscale image results in a more stable encryption process due to a general increase in variety across all CC's. Thus, the efficiency of each block is bound to increase, leading to a smaller ciphertext enlargement.

III. Experimental Design

All tests were run on an Intel i7-2600k at 4.6GHz with 28GB of DDR3 SDRAM on a 64-bit version of Windows 7, with minimal background processes running on the machine. **PixSel** was written in Java 1.8.

To ensure accurate results were gathered and that the machine was "warmed-up" to the task, we performed each function separately a set number of consecutive runs until it was processing at its peak performance, and then took the average of the top 20 runs. The number of runs for each test were made to be at least double the number of runs required to reach the peak performance for the encryption/decryption of any plaintext.

We bencharked **PixSel** using a JPEG image (Figure 4(a)), along with its greyscale counterpart (Figure 4(b)). Both images hold at least one occurrence of each value from 0-255, meaning that no data is enciphered through pairing. Note that **PixSel** functions with either JPEG or PNG source images of up to 4.2 gigapixels in size.



(a) Coloured (b) Greyscale

Fig. 4. Mercury Surface Map, NASA [4]. Colours were added to distinguish between rock types.

Multiple plaintexts were tested upon, varying both in size and file format (Table I). We had tested the same plaintext in different formats specifically to demonstrate how **PixSel** is affected by file type. As well, we had attempted the encipherment of an image, shown in Figure 5. Note that Anne.txt and Anne.docx were simply created from the text within Anne.pdf.

TABLE I
PROPERTY OVERVIEW OF TESTED PLAINTEXTS

File	Format	Size (Bytes)
Lorem Ipsum (392 words) [5]	.txt	2,682
Julius Caesar, Shakespeare[6]	.txt	116,231
Anne of Green Gables	.txt	566,811
Anne of Green Gables	.docx	277,802
Anne of Green Gables [7]	.pdf	1,137,065
House	.jpg	1,574,123

Fig. 5. An abandoned brick house located in southern Manitoba. Image taken by Mykel Shumay.

All tests were set to use the same percentage of *Elitism* (5%), as well as having the same tolerance for what the *Elite* blocks' use resulted in (set to at least 2 bytes); the threshold was set low purposely to prioritize a faster encryption process

TABLE II
PERFORMANCE ANALYSIS OF **PIXSEL** USING MERCURY.JPEG (COLOUR, FIGURE 4(A))

File	Initialization Time (s)	Encryption Time (s)	Decryption Time (s)	Enlargement
Lorem.txt	0.060089	0.017936	0.000119	11.82%
Caesar.txt	0.067426	0.071351	0.001087	18.52%
Anne.txt	0.064075	0.268013	0.004708	16.87%
Anne.docx	0.062106	0.216520	0.003132	39.27%
Anne.pdf	0.063107	0.877533	0.012523	39.13%
House.jpg	0.062399	1.242338	0.016763	38.99%

TABLE III
PERFORMANCE ANALYSIS OF **PIXSEL** USING MERCURY.JPEG (GREYSCALE, FIGURE 4(B))

File	Initialization Time (s)	Encryption Time (s)	Decryption Time (s)	Enlargement
Lorem.txt	0.057347	0.017132	0.000025	0.56%
Caesar.txt	0.059877	0.047504	0.000785	2.89%
Anne.txt	0.060867	0.184572	0.003819	3.17%
Anne.docx	0.061753	0.220708	0.002740	32.25%
Anne.pdf	0.061977	0.823105	0.010846	28.26%
House.jpg	0.062292	1.242453	0.015808	31.09%

over a more efficient ciphertext enlargement. Thus, it is likely that a lesser ciphertext enlargement may be achieved, although the additional computing power needed brings severely diminishing returns.

The source image must be processed once prior to encryption/decryption, and always took significantly less than one tenth of a second (see Table IV). Although this may be a trivial amount of time for larger plaintexts, it could essentially double the encryption time of a small plaintext while using a good image. Note that this does not include the time required to fetch the image from storage, as this may differ greatly from system to system.

TABLE IV
IMAGE PROCESSING ANALYSIS

Image	Processing Time (s)
Mercury.jpeg (Coloured)	0.018414
Mercury.jpeg (Greyscale)	0.018443

The initialization time for encryption is comprised of the needed processing to determine which blocks are *Elite*, along with indexing the contents of each *Elite* block. This has been separated from the encryption time to demonstrate that if this is all done beforehand and the knowledge is used for multiple encipherments, this would be a one-time expense. Note that this initialization time is not present within the decryption process.

IV. RESULTS

The performance of **PixSel** with each source image across all plaintexts is given in this Section. In Tables II and III it can be seen that the initialization time and decryption time were near-trivial, and that the encryption time scaled as expected. It was surprising how much better Figure 4 did across the board once converted to greyscale, both in ciphertext enlargement and computational efficiency.

A. Encryption Benchmarking

PixSel's general behavior when encrypting text documents is a very efficient ciphertext enlargement and reasonable computational requirements. This is less so for the encryption of images, due in part to the great variability that pictures hold, likely with significant jumps in byte-value within each pixel.

As can be seen, greyscale counterparts tend to perform much better in terms of ciphertext enlargement, although, interestingly, sometimes require roughly the same amount of computation as their original counterpart. Before benchmarking, we were expecting to see a close relation between ciphertext enlargement improvements and encryption time, as the greyscale counterparts are using many less blocks, and so require many less searches overall.

It is clear that the performance of our method depends greatly on the image used, as well as the type of plaintext that is being encrypted. Both the ciphertext enlargement and encryption time disparity between an image and its greyscale counterpart seems to be most profound when encrypting text, while only the enlargement experiences significant improvement when encrypting non-text files. Clearly, in certain scenarios, **PixSel** may be very efficient in both computational requirements and ciphertext enlargement.

B. Decryption Benchmarking

As decryption requires little work, it follows that the task would not be computationally heavy or time-intensive; the process consists mainly of reading the image, then looking up the given pixel positions within the image.

C. Lack of Entropy

It is fairly evident by our methods, such as prioritizing the use of the most optimal blocks, along with the inclusion of *Elitism*, that we had prioritized the lessening of the ciphertext enlargement over a balanced frequency distribution. With

some simple frequency analysis it was found that, without superencipherment, **PixSel**'s produced ciphertext was far from randomly distributed values, and does not differ greatly enough between multiple encipherments of the same plaintext with the same image. This is a characteristic that could possibly be revised in the future.

V. Conclusion

A main deterrent to the use of book ciphers is the large ciphertext enlargement, although **PixSel** showed that this could be overcome by using a different approach. **PixSel** performs quite well when encrypting text files with good greyscale images, and is capable of encrypting any data, regardless of type or structure.

In this paper, we explored an alternative implementation of a book cipher that uses images in place of literature as the source material, which we call **PixSel**. From the experimental results, the efficient nature and robustness of **PixSel** is clear. In comparison to the conventional book cipher, the ciphertext enlargement was reduced to trivial amounts, while still maintaining reasonable computation requirements. Although **PixSel** may perform differently depending on the data type, the limitation on encipherable data type was rid of, allowing more general use of this implementation of a book cipher.

References

[1] U. M. Maurer, "Conditionally-perfect secrecy and a provably-secure randomized cipher," *J. Cryptology*, vol. 5, no. 1, pp. 53–66, 1992. [Online]. Available: http://dx.doi.org/10.1007/BF00191321

[2] C. Wang and S. Ju, "A novel method to implement book cipher," *JCP*, vol. 5, no. 11, pp. 1621–1628, 2010. [Online]. Available: http://dx.doi.org/10.4304/jcp.5.11.1621-1628

[3] R. Lele, R. Jainani, V. Mikhelkar, A. Nade, and M. V. Meshram, "The book cipher optimised method to implement encryption and decryption," *Journal of scientific & technology research*, vol. 3, pp. 11–14, 2014.

[4] NASA, "First global topographic model of mercury," 2016. [Online]. Available: \url{https://www.nasa.gov/feature/first-global-topographic-model-of-mercury}

[5] "Lorem ipsum." [Online]. Available: \url{http://www.lipsum.com/}

[6] W. Shakespeare, "The life and death of julius caesar." [Online]. Available: \url{http://shakespeare.mit.edu/julius_caesar/full.html}

[7] L. M. Montgomery, "Anne of green gables." [Online]. Available: \url{http://www.freeclassicebooks.com/Lucy%20Maud%20Montgomery/Anne%20Of%20Green%20Gables.pdf}

APPENDIX A
PIXSEL'S ENCRYPTION PSEUDOCODE

Algorithm 1 PixSel: Encryption

$P \leftarrow$ Plaintext
$C \leftarrow$ new $List$ to store Ciphertext
Read source image
$messageIndex \leftarrow 0$
while $messageIndex < P.length$ **do**
    $maxBytes \leftarrow 0$
    $optimalBlock \leftarrow null$
    for all Blocks B **do**
        $rotIndex \leftarrow 0$ ▷ $rotIndex$ tracks the current CC
        $imitated \leftarrow 0$ ▷ Number of bytes B is able to imitate
        while Any $pixel$'s value within $rotIndex^{\text{th}}$ CC of B = $P[messageIndex + imitated]$ **do**
            $imitated \leftarrow imitated + 1$
            $rotIndex \leftarrow (rotIndex + 1) mod 3$
            Record $pixel$'s index within list L_B
        end while
        if $imitated > maxBytes$ **then**
            $optimalBlock \leftarrow B$
            $maxByes \leftarrow imitated$
        end if
    end for
    if $maxBytes = 0$ **then** ▷ No appropriate value within the image
        Append **Pairing Marker**'s value of 253 to C
        Find 2 pixels in previous block whose sum $= P[messageIndex]$, append indices to C
        $messageIndex \leftarrow messageIndex + 1$
    else
        $messageIndex \leftarrow messageIndex + imitated$
        Append $optimalBlock$'s index to C
        Append each index of $optimalBlock$'s used $pixel$s (within L_B) to C
        $satisfactory \leftarrow true$
        do ▷ Manual rotation test
            $rotIndex \leftarrow (rotIndex + 1) mod 3$
            Continue search for extra n imitatable bytes of P, starting at $messageIndex$
            if $n \geq tolerancePercentage \cdot imitated$ **then**
                Append **Colour Channel Rotation Marker**'s value of 254 to C
                Append each extra index of $optimalBlock$'s used $pixel$s to C
                $messageIndex \leftarrow messageIndex + n$
            else
                $satisfactory \leftarrow false$
            end if
        while $satisfactory$
        Append **Block-Termination Marker**'s value of 255 to C ▷ End of $optimalBlock$
    end if
end while
Write C to a file

APPENDIX B
PIXSEL'S DECRYPTION PSEUDOCODE

Algorithm 2 PixSel: Decryption

$C \leftarrow$ Ciphertext
$P \leftarrow$ new $List$ to store the deciphered Plaintext
Read source image
$index \leftarrow 0$
while $index < C.length$ **do**
    $currentBlock \leftarrow$ next two bytes of C ▷ Or next three bytes, if the image is large
    $index \leftarrow index + 2$ ▷ Or $index \leftarrow index + 3$
    $rotIndex \leftarrow 0$ ▷ $rotIndex$ tracks the current CC
    while $C[index] \neq 255$ **do** ▷ 255 marks the end of a block's use
        if $C[index] = 254$ **then** ▷ Manual CC rotation
            $rotIndex \leftarrow (rotIndex + 1) mod 3$
        else if $C[index] = 253$ **then** ▷ A pair of pixels is to follow
            Sum previous block's $C[index + 1]$ and $C[index + 2]$'s pixel's $rotIndex^{th}$ CC's, append to P
            $index \leftarrow index + 2$
        else
            Append $currentBlock$'s $C[index]^{\text{th}}$ $pixel$'s $rotIndex^{\text{th}}$ CC's value to P
        end if
        $index \leftarrow index + 1$
    end while
    $index \leftarrow index + 1$
end while
Write P to a file