Image Encryption by Pixel Property Separation

Karthik Chandrashekar Iyer and Aravinda Subramanya

Abstract— Pixels in an image are essentially constituted of two properties, position and colour. Pixel Property Separation, a radically different approach for Symmetric-key image encryption, separates these properties to disturb the semantics of the image. The scheme operates in two orthogonal stages each requiring an encryption key. The first stage creates the Position Vector, an ordered set of Pixel Position Information controlled by a set of plaintext dependent Random Permutations. A bitmap flagging the presence of all the 24 bit colours is generated. The second stage randomly positions the image width and height within the ciphertext and finally applies a byte transposition on the ciphertext bytes. The complete set of image properties including width, height and pixel position-colour correlation are obscured, resulting in a practically unbreakable encryption. The orthogonality of the stages acts as an anti-catalyst for cryptanalysis. The information retrieved from compromising a stage is totally independent and cannot be used to derive the other. Classical cryptanalytic techniques demand huge number of attempts, most failing to generate valid encryption information. Plaintext attacks are rendered ineffective due to the dependency of the Random Permutations on the plaintext. Linear and Differential cryptanalysis are highly inefficient due to high Diffusion and Confusion. Although the paper describes the algorithm as applied to images, its applicability is not limited to images only. The cryptographic strength is independent of the nature of the plaintext.

Index Terms— Pixel Property Separation, Image Encryption, Cryptanalytic Error Avalanche Effect, random colour permutation, security, cryptography, pixel position, pixel colour, plaintext attack, confusion, diffusion.

1 Introduction

IMAGE and data security is a major challenge in Storage and Transmission applications. Encryption algorithms for these applications are exposed to various threats and security breaches due to the availability of immensely powerful and inexpensive computational resources. Brute Force and Statistical attacks on the existing cryptographic algorithms is not only possible, but are becoming more practical in the wake of technological advancements like Distributed and Grid Computing. Vast amounts of data can be processed in parallel by agents distributed over the Internet and aid in revealing secure information.

Several data encryption algorithms like *DES* [1], *AES*[1], *IDEA*[1] are being employed for protecting digital information, *chaos based* [5][17], *combinatorial permutation* [13] and *optical techniques* [12] are also proposed for encrypting images. Along with these developments in the security domain, the vulnerability of the algorithms are also being exposed. It is possible to build a machine that can determine the key used for DES encryption at a cost as low as US $10000 [16]. It is also vulnerable to Linear and Differential cryptanalysis or a combination of both. Techniques like the *Side Channel Attack* and several *Cache Timing Attacks* have been developed to compromise AES algorithm and retrieve the encryption key in as less as 65ms with 800 write operations [6]. Chaos based techniques like the *CKBA* are prone to plaintext attacks [7] and algorithms using combinatorial permutations are as strong as the permutation of the least sized block even if they apply multiple permutations over different sized image blocks.

Applications in the Automobile, Medical, Construction and the Fashion industry require designs, scanned data, building plans and blue-prints to be safe-guarded against espionage. Considering the long lifetime of images in the mentioned domains, it is imperative to develop and employ techniques which protect the content throughout their lifetime.

A novel image encryption technique based on *Pixel Property Separation* is proposed in this paper. The pixel position and the colour are separated and encoded using a set of Random Permutations. The algorithm conceals the image colour-position correlation, the colour information and the image size. The concealment of the image size considerably enhances the cryptanalytic complexity. The Ciphertext only attack is practically impossible while the plaintext and the differential attacks fail to reveal useful encryption information. The algorithm is robust against Plaintext attacks due to the utilization of Plaintext Dependent Random Permutations ([18],[19]) to separate pixel properties. The time required for a successful Brute Force Attack is huge attributing to the high key lengths and orthogonality of encryption stages. Hence it is not likely to be broken by brute force methods using any existing technology.

- *K. C. Iyer and A. Subramanya are with the DSP Software Centre, Innovation Center, Business Unit Home, NXP Semiconductors (formerly Philips Semiconductors), Bangalore - 560 045, India.*
E-mail: karthik.iyer@nxp.com, kiyer21@yahoo.com
aravinda.bairy@nxp.com, arubairy@gmail.com

The organization of the paper is as follows. Section 2 describes Pixel Property Separation, section 3 explicates and illustrates the encryption scheme. Section 4 and 5 discuss in detail the various cryptanalytic attacks and the encryption strength. The last section presents the simulation results. A mathematical model for the encryption algorithm has been presented in the appendix.

2 Pixel Property Separation

Pixels in an image are essentially constituted of two properties, pixel position and pixel colour. The pixel position is defined by the (x, y) co-ordinates indicating the horizontal and vertical distance of the pixel from $(0, 0)$ and the colour value can be a RGB colour or a greyscale value. *Pixel Property Separation* is an encoding technique that separates the pixel position and colour and represents them as distinct vectors. The separation process results in two ordered vectors, the Position Vector Pos representing the pixel positions and the Colour Bitmap Vector CBM that represents the colours. The vectors are defined as follows.

1) The Position Vector Pos is an ordered set of pixel co-ordinate positions. This vector bears a position entry for each pixel in the input image. For each (x, y) position entry, the Pos vector also indicates by a flag flg whether an entry is the last entry for that colour. A flag value of 1 indicates that the entry is the last for that colour, a value of 0, otherwise.
2) The Colour Bitmap CBM is an ordered set of bits(flags) that indicate the presence of a colour C in the input image. The number of bits in CBM is equal to the number of colours in the colour system used by the image. For example, if the image uses a 24 bit RGB colour system, then the CBM would be composed of 2^{24} flags indicating the presence of the 2^{24} RGB values. A flag value of 1 indicates the presence of a colour and a value 0, otherwise.

These two vectors in combination, completely represent the image. The following illustration describes the technique.

Consider a system S of 6 colours C_0 through C_5, $S = \{C_0, C_1, C_2, C_3, C_4, C_5\}$. Let *img* be a colour image with width $W = 3$ and height $H = 3$ and be composed of a set of 4 colours, $\{C_0, C_1, C_2, C_4\}$. Figure 1 depicts the image *img*. For a better understanding of the technique, we represent the input image *img* as depicted in Figure 1b where pixels are grouped based on their colour. The input image is initially scanned to group pixels based on their colour. The Position Vector Pos is created as follows. For each

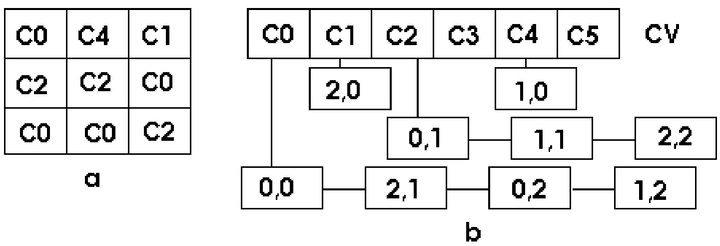

Fig. 1: Input image representation a. Original b. Colour based pixel grouping

colour $C \in S$, starting from C_0 to C_5, the (x, y) positions of the pixels bearing C is augmented to Pos. Flag flg is also entered for each pixel considered. For example, there are 4 pixels bearing the colour C_0. The position of the first pixel bearing C_0 is $(0, 0)$ and it is not the last pixel bearing C_0. Hence the first entry in Pos is $\{(0, 0), 0\}$. Considering the second and third pixels bearing C_0, the next two entries in Pos are $\{(2, 1), 0\}$ and $\{(0, 2), 0\}$. Pixel $(1, 2)$ is the last pixel bearing colour C_0. Hence the Pos entry for that pixel is $\{(1, 2), 1\}$. Hence, for the colour C_0, $Pos = [\{(0,0),0\}, \{(2,1),0\}, \{(0,2),0\}, \{(1,2),1\}]$. Similarly,

1) Considering C_1, $Pos = [\{(0,0),0\}, \{(2,1),0\}, \{(0,2),0\}, \{(1,2),1\}, \{(2,0),1\}]$
2) Considering C_2, $Pos = [\{(0,0),0\}, \{(2,1),0\}, \{(0,2),0\}, \{(1,2),1\}, \{(2,0),1\}, \{(0,1),0\}, \{(1,1),0\}, \{(2,2),1\}]$
3) Considering C_3. Since there are no pixels in *img* bearing C_3, no entries are made for C_3.
4) Considering C_4, $Pos = [\{(0,0),0\}, \{(2,1),0\}, \{(0,2),0\}, \{(1,2),1\}, \{(2,0),1\}, \{(0,1),0\}, \{(1,1),0\}, \{(2,2),1\}, \{(1,0),1\}]$
5) Considering C_5. Since there are no pixels in *img* bearing C_5, no entries are made for C_5.

The Position vector Pos for the image *img* is

$$Pos = [\{(0,0),0\}, \{(2,1),0\}, \{(0,2),0\}, \{(1,2),1\}, \{(2,0),1\}, \{(0,1),0\}, \{(1,1),0\}, \{(2,2),1\}, \{(1,0),1\}]$$

As stated earlier, the Colour Bitmap CBM enters a flag for each colour in the system S indicating the presence of that colour in *img*. Since, the colour C_0 is present in *img*, the first entry in CBM is a '1'. The next two entries in CBM are '1's since both C_1 and C_2 are present. None of the pixels in *img* bear colour

C_3 and hence the CBM entry for C_3 is a '0'. Similarly, the CBM entries corresponding to C_4 and C_5 are '1' and '0'. Hence, $CBM = [1, 1, 1, 0, 1, 0]$

It can be noted that the vectors Pos and CBM together, completely represent the input image *img*. This can be shown by reconstructing *img* using only Pos and CBM. The decoding process is described below.

The first entry of CBM is a '1' and hence colour C_0 is present in *img*. Since the same colour order C_0 through C_5 is used to create both Pos and CBM, the first entry $\{(0, 0), 0\}$ in Pos corresponds to C_0. Hence, the pixel with position $(0, 0)$ bears colour C_0. Hence, *img* $(0, 0) = C_0$. Since, the first Pos entry bears a flg value of '0', it is not the last pixel bearing C_0. Hence, the pixel corresponding to the second entry $\{(2, 1), 0\}$ in Pos also bears colour C_0, *img* $(2, 1) = C_0$. Similarly, corresponding to the third and the fourth entry in Pos, *img* $(0, 2) = C_0$ and *img* $(1, 2) = C_0$. But the flg value of the fourth entry in Pos is '1' meaning that there are no more pixels bearing colour C_0. For the fifth entry in Pos, the next colour in CBM is considered. The second entry in CBM corresponds to colour C_1 and has a value '1'. Hence, for the the fifth entry in Pos, colour C_1 is considered. Similarly, $img(2, 0) = C_1$, $img(0, 1) = C_2$, $img(1, 1) = C_2$, $img(2, 2) = C_2$ and $img(1, 0) = C_4$.

It can be noted that the pixel position information and the colour information are completely separated and have been encoded as two distinct ordered vectors. Though it is straight forward to reconstruct the input image using Pos and CBM, the advantage of such a representation of images (or any form of digital data) is as follows:

1) The Pos vector is composed of numbers from 0 to W and 0 to H. This information is obvious if the image width and height is known.
2) no information regarding the colours is revealed. CBM encodes the colours as a bit sequence
3) the Pixel Position-Colour correlation that defines the semantics of an image, is completely disturbed.

***Digital Representation of* Pos *and* CBM:** Each entry in the Pos vector is composed of three elements. The 'x' and the 'y' co-ordinates of the pixel considered and the flag flg indicating if the pixel is the last entry for that colour. Since the flag flg can take only two values '0' or '1', one digital memory bit is sufficient to represent it. The value of the 'x' co-ordinate defines the relative horizontal distance of a pixel from the origin $(0, 0)$ and can not exceed the image width W. Hence the number of bits necessary to represent the 'x' co-ordinate of any pixel is given by bw.

$$bw = \text{If } (W \leq 2) \text{, then } 1 \text{, else, } (\lceil \log_2 (W) \rceil) \text{ bits}$$

Similarly, the number of bits necessary to represent the 'y' co-ordinate of any pixel is given by bh.

$$bh = \text{If } (H \leq 2) \text{, then } 1 \text{, else, } (\lceil \log_2 (H) \rceil) \text{ bits}$$

Hence, the number of bits used to represent an entry in the Pos vector is $(bw + bh + 1)$ bits. Considering all the $W \times H$ entries in the Pos vector, the number of bytes required to represent the Pos vector is

$$Size\,(Pos) = \lceil ((bw + bh + 1) \times (W \times H)) / 8 \rceil \text{ bytes}$$

For the image *img*, $bw = bh = 2$. Hence, each entry in Pos vector is represented using 5 bits and the entire vector is represented using 45 bits. The Pos vector can be represented as:

$$Pos = [(00000), (10010), (00100), (01101), (10001), (00010), (01010), (10101), (01001)]$$

Finally, the Pos vector is represented as an ordered set of integers by packing 8 bits in sequence. For example, the first integer with value '4' is formed by packing 5 bits of the first Pos entry and 3 bits of the second entry. The second integer with value '136' is formed by packing 2 bits from the second Pos entry, 5 from the third and one bit from the fourth Pos entry. Similarly, the encoding continues until the last bit in Pos. Since the Pos vector is composed of 45 bits, the last 5 bits (last Pos entry) are padded with three '0' bits. The final representation of the Pos vector is given by $Pos = [4, 136, 216, 137, 85, 72]$.

The CBM vector, by definition is a set of 2^{24} flags (considering 24 bit RGB colours), each of which can be represented by a single bit. Since CBM is a sequence of '1's and '0's, it can be coded as a sequence of bytes representing runs of subsequent '1's and '0's. It can be noted that the exact number of bytes required to represent CBM depends on the composition of CBM. The CBM vector for image *img* can be coded as $CBM = [1, 3, 1, 1, 1]$, indicating that the first 3 bits are '1's followed by '0', '1' and '0'. The technique used in this example encodes the first byte with the value of the first bit in CBM. The second byte encodes the run of the bit type represented by the first byte. Subsequent bytes alternatively encode the runs of 1s and 0s (0s and 1s). It shall be noted that the encryption technique is independent of the coding technique used.

This section has clearly described the concept of *Pixel Property Separation* and the representation of images using the vectors Pos and CBM. The next section develops the encryption algorithm based on this technique.

3 Encryption by Pixel Property Separation

The Encryption scheme is a *Secret Key Algorithm* requiring 2 keys . It operates on the input plain image *img* with width W and height H, the two encryption keys $key1$ and $key2$ and generates the encrypted image *img*$'$ with width W' and height H' (representing the ciphertext as img' with $W' \times H'$ pixels is optional, the ciphertext can be represented as a byte sequence). The input plain image uses a colour system S composed of s colours, the number of distinct colours in *img* being $\leq s$. For a 24 bit RGB colour space, S is the set of all colour values from 0 to $\left(2^{24}-1\right)$ and $s = 2^{24}$. The scheme uses the encryption keys to generate a set of Random Permutation RPs to separately encode the pixel position and colour. The composition of the RPs cannot be determined by the keys alone. This is because, the generation of a permutation in any iteration not only depends on the key, but also depends on the results of the previous iteration. Hence, the RPs used in this encryption scheme depend both on the key and the plaintext. This renders the technique robust against Plaintext Attacks.

Encryption by Pixel Property Separation (EA) is completely based on the technique of separating pixel position and colour. To achieve better encryption strength, the basic design (BD) of separating pixel properties, explained in the previous section has been modified. The following enlists the modifications.

1) EA uses NRP number of RPs $(RP_1, RP_2, \ldots, RP_{NRP})$ to create the Pos vector while BD used only one permutation with colour order $[C_0, C_1...C_s]$
2) EA uses one of the s! (2^{24}! for 24 bit RGB colours) random colour permutations composed of all the colours in S as its first permutation. This is used as the first permutation RP_1 in the creation of the Pos vector and is also used to create the CBM. BD uses a known colour order $[C_0, C_1...C_s]$ which is not randomly picked.
3) For any colour $C \in S$ considered for processing in an RP, EA places exactly one pixel's position (x, y) in the Pos vector, while BD placed the positions of all the pixels bearing colour C

At any stage in the encryption scheme, the RPs are used to define a random colour order. The number of elements in the first permutation RP_1 is equal to that of the set S. RP_1 needs to consider all the colours in S as it is also used to generate the vector CBM. RP_1 defines a random colour order for all the colours in S. The size and the composition of the remaining $(NRP-1)$ RPs, RP_2 through RP_{NRP}, depend on the number of colours available for processing at the point of generation of the RP. In the illustration of Section 2, the colour order $[C_0, C_1, C_2, C_3, C_4, C_5]$ can be regarded as RP_1 and it defines an order for all the colours C_0 through C_5. Hence, the size of RP_1 is 6. Applying modification 3 to this illustration would result in the consideration of one pixel each of colours C_0, C_1, C_2 and C_4 in the first iteration using RP_1. It can be noted that the order of colours considered is same as that in RP_1. For the second iteration, colours C_1, C_3, C_4, and C_5 can be discarded as there are no pixels bearing these colours. Hence for RP_2, there remain only two colours C_0 and C_2 that need an order. RP_2 would then be composed of two elements. After the second iteration, there remain three pixels for processing, bearing two distinct colours C_0 and C_2. Hence, the number of elements in RP_3 is 2.

It can be noted that each RP defines a colour order for the pixels considered for processing. However, after the third iteration, there remains only one pixel at position $(1, 2)$ bearing C_0. Since there is only one colour available for processing, there cannot be a colour order defined. Hence, for the image img in Figure 1, three permutations RP_1, RP_2 and RP_3 suffice to create the Pos vector. Hence, the value of NRP for img is 3. The value NRP (Number of Random Permutations) makes sure that there are enough RPs available to define a colour order for pixels, processed at the rate of *'one pixel per colour per RP'*, until there are pixels to be processed in the plaintext bearing at least two colours. The key $key1$ is used to generate the NRP permutations RP_1 to RP_{NRP}. The derivation of NRP for an image can be found in the Appendix.

It can be noted that the value of NRP, the number of elements and the composition of the permutations RP_2 through RP_{NRP} depends on the number of colours available for processing at any point in the encryption process. This in turn depends on the number of pixels bearing each colour $C \in S$ in the plaintext. Hence, the size and the composition of the random permutations used to create the Pos vector depends on both the $key1$ and the plaintext. This greatly increases the cryptanalytic complexity and renders the technique robust against Plaintext Attacks.

The vector CBM is created using the first permutation RP_1, coded and digitally represented as described in the previous section. The ciphertext is formed by concatenating the Pos and the CBM vectors. The ciphertext bytes are finally shuffled using a random permutation. However, before the final shuffle, the encryption technique carries out two steps described below.

If the ciphertext needs to be represented as an image *img′*, the ciphertext bytes needs to be padded to render *img′* as a rectangular image with integral values for W' and H' and with each pixel composed of 3 bytes (considering 2^{24} colours). The values for the padding bytes needs to be randomly chosen and should not follow any pattern. The padding bytes are augmented to the ciphertext bytes before applying the final shuffle. Vector $P = [P_0, P_1, \ldots, P_{p-1}]$ gives the padding bytes where p is the number of padding bytes necessary. It shall be noted that this is an optional step and hence need not be carried out by an implementation. The determination of the number and the values for the padding bytes is not discussed in this paper.

From the previous section, it can be noted that each entry in the Pos vector can be digitally represented using $(bw + bh + 1)$ bits. To correctly decode such a bit-packed Pos vector, the knowledge of bw and bh is necessary. Without these values, it is not possible to discern how many bits are to be considered for the 'x' and the 'y' co-ordinate entries. Hence, the image width W and the height H values need to encoded as part of the ciphertext. The cryptanalytic complexity can be greatly increased if these values are randomly placed within the ciphertext. Since the values of W and H is not known, the cryptanalyst needs to attempt decryption considering all possible combination of bw and bh values to decode the Pos vector. Section 4.2 derives an equation for the number of ciphertext byte combinations that need to be extracted from the ciphertext to list all possible candidates for W and H. Assuming a maximum value of 65535 for W and H, the encryption technique uses two bytes each (W_1, W_2) and (H_1, H_2) to represent W and H respectively, with W_1, H_1 being the lower and W_2, H_2, the higher bytes. The technique randomly places the four size bytes in the ciphertext after padding. It can be noted that the technique is not limited to using two bytes to represent W and H. The following example illustrates the encryption technique.

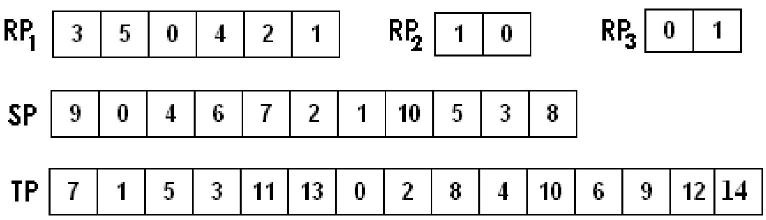

Fig. 2: Plaintext Dependent Random Permutations

The example depicted in Figure 1 has been used to illustrate the technique. Figure 2 depicts the random permutations used to encrypt *img*. For *img*, $bw = bh = 2$, $NRP = 3$, $W_1 = H_1 = 0$ and $W_2 = H_2 = 3$.

The first entry in RP_1 is '3'. But from the colour vector CV (Figure 1), it can be noted that no pixel bears C_3. Hence, there are no Pos entries corresponding to C_3. Since no pixels bear C_5, there are no Pos entries corresponding to the second entry '5'. The third entry in RP_1 corresponds to colour C_0 and hence the first entry in Pos is $\{(0,0), 0\}$. Similarly, the Pos entries corresponding to colours C_4, C_2 and C_1 (in that order) are $\{(1,0), 1\}$, $\{(0,1), 1\}$ and $\{(2,0), 1\}$. Let noc represent the number of distinct colours in *img* (noc = 4 in this example). It can be noted that in this technique, RP_1 processes exactly noc pixels. For RP_2, only C_0 and C_2 remain for processing. The first entry in RP_2 is '1', which picks the second available colour for processing in CV, which is C_2. Hence, the fifth entry in Pos is $\{(1,1), 0\}$. Similarly, after processing all pixels, the Pos vector corresponding to *img* and the permutation set in Figure 2 is given by

$$Pos = [\{(0,0),0\}, \{(1,0),1\}, \{(0,1),1\}, \{(2,0),1\}, \{(1,1),0\}, \{(2,1),0\}, \{(0,2),0\}, \{(2,2),1\}, \{(1,2),1\}] = [2, 71, 21, 72, 149, 104]$$

Vector CBM is created using the colour order defined by RP_1. $CBM = [0,0,1,1,1,1] = [0,2,4]$.
Pos and CBM are concatenated to form the vector PB with pb elements.

$$PB = [Pos, CBM] = [2, 71, 21, 72, 149, 104, 0, 2, 4], \; pb = 9$$

As mentioned earlier, the values W_1, H_1, W_2 and H_2 need to be placed at random positions within the ciphertext. But the size of the ciphertext after placing these bytes would be $pb + 4 = 13$, which is not a multiple of 3. PB needs to be padded with two bytes P_0 and P_1 to make the ciphertext size equal to 15. Let $P_0 = 157$ and $P_1 = 43$.

$$PB' = [PB, P_0, P_1] = [2, 71, 21, 72, 149, 104, 0, 2, 4, 157, 43]$$

To determine the random positions for the 4 size bytes within PB', the key $key2$ is used to generate a Random Permutation SP composed of $(pb + p)$ numbers (11 in this example) with values ranging from 0 to $(pb + p - 1)$. The first four entries of SP are used as byte positions to place W_1, H_1, W_2 and H_2 within PB' to create vector F' with $f = (pb + p + 4)$ elements. Considering SP in Figure 2,

$$F' = [H1, 2, 71, 21, W2, 72, H2, 149, 104, 0, 2, 4, W1, 157, 43] = [0, 2, 71, 21, 3, 72, 3, 149, 104, 0, 2, 4, 0, 157, 43],\ f = 15$$

Finally, the vector F' is transposed using a random permutation TP with f elements with values ranging from 0 to $(f-1)$. Key $key2$ is used to generate TP.

$$F = TP(F') = [F_0, F_1, \ldots, F_{f-1}] = [149, 2, 72, 21, 4, 157, 0, 71, 104, 3, 2, 3, 0, 0, 43]$$

The ciphertext size is $f = 15$ bytes and is composed of $nop = (f/3) = 5$ pixels. Starting from the set $[F_0, F_1, F_2]$, each set of 3 bytes in sequence represents a pixel. The values W' and H' are any two factors of nop such that $W' \times H' = nop$. Assuming $W' = 5$ and $H' = 1$, the cipher image img' is given by,

9765448	1377437	18280	197123	43

4 CRYPTANALYSIS

This section explicates the different techniques for cryptanalysis. The Brute Force and the Plaintext attacks have been analysed. Differential Cryptanalysis has been discussed as part of Section 5.

4.1 Brute Force Attack

In a Brute Force Attack, the algorithm implementation is considered as a black box and all valid key combinations are attempted until the correct key is discovered. The complexity of such an attack depends on the size of the keys used for encryption.

Key Size key1: The maximum size of $key1$, NBK_1, for a given plain image depends on the value of NRP and the number of elements in each of the NRP permutations. The first permutation RP_1 is composed of 2^{24} numbers that can be arranged in $(2^{24}!)$ ways. If noc_i represents the number of colours available for processing for permutation RP_i, then for a given plain image, the maximum number of key bits in $key1$ is $NBK_1 = \log_2 ((\lceil 2^{24}! \rceil) \times (\lceil noc_2! \rceil) \times (\lceil noc_3! \rceil) \times \ldots, (\lceil noc_{NRP}! \rceil))$. However, an implementation can choose NBK_1 independent of the plaintext. The permutation generator generates one of the 2^{NBK_1} possible combinations of NRP permutations.

Key Size key2: Key $key2$ is used to generate SP and TP. Like $key1$, the size of $key2$, NBK_2, depends on the ciphertext size $(f = W' \times H' \times 3)$, which is a variable entity. But, the size of $key2$ can be chosen independent of the ciphertext size, because, though the value of (f) is plaintext/ciphertext dependent, the Random Permutation Generator generates one of the 2^{NBK_2} random permutations of composed of (f) numbers.

In the proposed scheme, it is not possible to individually determine $key1$ or $key2$ during decryption. This is because the correctness of the decryption attempts involving $key1$ or $key2$ cannot be determined until the entire decryption process is complete and the result is verified for correctness. Hence, on average, $((2^{NBK_1+NBK_2})/2)$ attempts are required to successfully compromise the keys.

Also, the presence of any colour in the plaintext is flagged as a bit in the tuple CBM and hence is not known to the cryptanalyst. While decryption, as each element Pos_k is considered, if the pixel $img(w_k, h_k)$ bearing the colour C is the last pixel of that colour (indicated by $flg_k = 1$), then C is discarded for further processing, as there are no more pixels of colour C available. The information as to what colours have to be discarded in RP_n is known only after all the colours available for the previous permutation RP_{n-1} have been considered for processing. Hence, for a given set of keys and a given ciphertext, decryption is a sequential process and can not be parallelized. This greatly increases the complexity of the Brute Force Attack.

4.2 Ciphertext only Attack

This method considers different stages in the decryption process and determines the candidates for each stage without using the encryption keys. The right set of candidates will successfully decrypt the cipher image. The method consists of the following stages.

Reverse Transposition: This is the process of deriving F' from the *transposed encoded byte vector* F. Since the size of F is f bytes, there are $(f!)$ possible arrangements, one of them being F itself. Therefore there are $(f)! - 1$ different possible candidates for F'. Since the values of the bytes in F are just numbers, there is no specific signature or references that can prove the validity of the reverse transposition result as a valid candidate for F'. Hence, only after the entire decryption process that the validity of any reverse transposition result be proved. The average number of complete decryptions to successfully get the correct solution for this stage is $((f! - 1)/2)$.

Derivation of W and H: This step involves the extraction of the randomly placed 4 size bytes W_1, W_2, H_1 and H_2 in F'. Since W and H are unknown, all possible width and height values have to be considered for decryption. There are $\left({}^{f}P_4\right)$ ways of choosing 4 out of a set of f bytes. Since 2 bytes represent W or H, the plaintext can have a maximum of 65536×65536 pixels, which is not the case always. If the plaintext supports a maximum size of 4096×4096, then the most significant byte of the width and the height cannot exceed the *threshold* value of 16. All values that exceed this threshold in the higher byte can be discarded. Also, the set of bytes which result in a value 0 for any of W or H can be discarded. This knowledge assists in eliminating all invalid byte combinations as candidates for this stage. Assuming that there are z bytes with value zero and u bytes with value greater than the threshold and if, M is the no. of combinations such that W or $H = 0$, T is the no. of combinations such that W_2 or $H_2 > t$ and MT is the no. of combinations satisfying both M and T, the number of valid candidates for this stage is given by

$$V = {}^{f}P_4 - M - T + MT^{1} \tag{1}$$

where,

- M = ${}^{z}P_4 + 4\,(f-z)\,({}^{z}P_3) + 2\,({}^{z}P_2)\,\left({}^{(f-z)}P_2\right)$
- T = ${}^{u}P_4 + 4\,(f-u)\,({}^{u}P_3) + 5\,({}^{u}P_2)\,\left({}^{(f-u)}P_2\right) + 2\,(u)\,\left({}^{(f-u)}P_3\right)$
- MT = $2\,(u)\,(f-u)\,({}^{z}P_2) + 2\,({}^{z}P_2)\,({}^{u}P_2)$

MT is added to V because these combinations are internally eliminated twice in the equation, once in M and one more time in T.

Retrieval of vectors Pos and CBM: For a given Reverse Transposition candidate and W and H values, the retrieval of Pos vector involves extracting $(bw + bh + 1)$ bits for each of the $W \times H$ pixels, starting from the first byte of the Reverse Transposition candidate. Since CBM is concatenated to Pos, it can be derived by considering bytes that follow the Pos vector bytes. However, the retrieved Pos and CBM vector candidates has to satisfy the following conditions to be considered as a valid ones.

1) If Pos = (w_k, h_k), $w_k < W$ and $h_k < H$, $\forall\ 0 \leq k < (W \times H)$
2) The extracted and decoded tuple CBM is composed of exactly 2^{24} bits.

Since w_k and h_k values are bit packed and not aligned to byte boundaries, the retrieval of these values from the Pos vector and the verification of the above condition involves huge number of bitwise operations. If the above conditions are not satisfied, then the cryptanalyst can choose another set of 4 size bytes or another Reverse Transposition candidate. For a valid candidate, the number of colours noc is given by the number of '1's in the retrieved CBM. At this point, though the cryptanalyst can verify the candidature of Pos, CBM, W, H and noc, the correctness of these parameters cannot be verified.

Image Reconstruction: The Image Reconstruction process uses a set of random permutations and the vectors Pos and CBM to match each Pos_k with its colour. The number of permutations required and their composition depends on the candidate Pos vector and is different for each decryption attempt. If $dnoc_i$ represent the number of colours available for the i^{th} permutation and $DNRP$, the number of permutations required for a decryption attempt, then the cryptanalyst, for a given Pos vector candidate, needs to attempt a maximum of $\left(\left(\lceil 2^{24}! \rceil\right) \times \left(\lceil dnoc_2! \rceil\right) \times \left(\lceil dnoc_3! \rceil\right) \times \ldots \times \left(\lceil dnoc_{DNRP}! \rceil\right)\right)$ combinations of random permutations to verify if the ciphertext has been successfully decrypted.

It shall be noted that none of the stages can be validated for correctness individually. Rather, for each attempt in each stage, it is necessary to complete the entire decryption process to verify their validity. Hence, the time required for a successful attack is not the sum of the time required for each stage, but their product. Because of the multiplication of time needed for each stage and excessively large number of iterations, it is highly impractical to break this algorithm without necessary information.

4.3 Chosen Plain Text Attack

The goal of Chosen Plain Text Attack (*CPTA*) is to reveal encryption information like the set of permutations RP_1 through RP_{NRP}, permutations TP and SP. The cryptanalyst has the knowledge of the algorithm and is provided with an *Encryptor* that can encrypt any input plaintext with keys $key1$ and $key2$. The cryptanalyst has the ability to fabricate arbitrary plaintexts, which, when encrypted have the potential to reveal encryption information. Though the intention of a typical *CPTA* is to reveal the encryption keys, in the proposed scheme, retrieval of keys $key1$ and $key2$ from RPs, TP and SP is immensely complex. Since the generation of a Random Permutations involve the usage of the keys as seeds of a *Cryptographically Secure Pseudo Random Number Generator*, the process of key generation starting from the permutations is practically impossible.

1. See Appendix for derivation

It can be noted from the previous sections that the permutations RP_2 through RP_{NRP}, TP and SP depend on parameters which are derived based on the properties of the input plaintext. Also, for a given key, the *PRNG* generates different permutations based on the number of elements in the permutation. Hence, these permutations are rendered useless since they cannot be applied for other images. As a result, the only useful encryption information that the *CPTA* can potentially reveal is RP_1 which depends only on the key.

Consider an input plaintext img_1 with $W = H = 1$. The only pixel in the image, identified by $img_1\,(0,0)$, bears the colour C. Let img_1' be the encrypted image with width W' and height H', with each of the $W' \times H'$ pixels bearing any of the 2^{24} colours. For such an image, $NRP = 1$ and Pos = [(0,0,1)] = 00100000 (5 bits 0 padding). Since img_1 is composed of only one colour, the colour bitmap would contain one entry of '1' and huge runs of '0's. Hence the only possible values CBM (digitally encoded - refer Section 2) bytes can take are 0, 1 and 255.

The first step in cryptanalysis is the reverse transposition operation to derive F' from F. It can be noted that there are $((f!) - 1)$ possible candidates for F'. But with the knowledge that $Pos = 00100000$ and that it precedes any other byte in F', all outcomes with the first byte value 00100000 are candidates for F'. Since the first byte is fixed, the number of such outcomes are $(f-1)!$. It is clear from the previous section that it is not possible to discern until the entire decryption process that which of the $(f-1)!$ candidate outcomes is F'. The next step is to pick the 4 size bytes from the ciphertext. Since the positions of the size bytes are unknown, the cryptanalyst has to choose a set of 4 bytes having the values $0, 1, 0, 1$. Though there is a possibility that the cryptanalyst choses an invalid size byte set, the maximum error an invalid size byte choice can introduce is the shift of the only 1 in the CBM, which further results in an incorrect position of C in RP_1. It can be noted that such an error can affect only CBM because bulk of img_1' is CBM with only a byte occupied by Pos. Assuming W_1, W_2, H_1 and H_2 are correctly chosen and removed from the candidate, the cryptanalyst now has the task of deriving the position of C in RP_1, which can be any of the 2^{24} different possible positions.

It is apparent that only when the cryptanalysis results in img_1, that the correctness of the reverse transposition operation, choice of the size bytes and that of RP_1 can be proved. But in this attack, all the 2^{24} choices of RP_1 will result in img_1 rendering this attack useless. The following example explains this phenomenon.

Let one of the reverse transposition candidates take the following form, $F' : [32, 1, 1, 255, 0, 255 \ldots]$, indicating that the Pos value is 32(00100000), the first bit of the CBM is 1, followed by a sequence of 510 0s and so on. Since the first element of CBM is 1, the attempt in which RP_1 has C as its first colour results in img_1. Similarly if, $F' : [32, 0, 1, 1, 255, 0, 255 \ldots]$, usage of a candidate RP_1 having C as its second colour results in img_1. Hence, with the above attack, it is not possible to uniquely determine the position of C in RP_1.

Consider input plaintext img_2 with $W = 2$ and $H = 1$. The two pixels in the image, identified by $img_2\,(0,0)$ and $img_2\,(0,1)$, bear colours $C1$ and $C2$ respectively. $NRP = 1$. Let img_2' be the encrypted image with width W' and height H', each of the $W' \times H'$ pixels bearing any of the 2^{24} colours. Depending on whether $C1$ appears before or after $C2$ in RP_1, the position vector takes the following form.

1) Colour order $C1\ C2$, Pos : [(0,0,1),(0,1,1)] = 00101100 (2 bits 0 padding)
2) Colour order $C2\ C1$, Pos : [(0,1,1),(0,0,1)] = 01100100 (2 bits 0 padding)

Since the cryptanalyst does not have the knowledge of RP_1, all reverse transposition results that have their first byte value either 00101100 or 01100100 become candidates for F'. In this case it is straight forward to determine the relative positions of $C1$ and $C2$. The occurrence of a byte 00101100 reveals that the colour order is $C1\ C2$ and of 01100100, colour order $C2\ C1$. The cryptanalyst may get confused if both the bytes occur in the cipher text img_2'. In general, the cryptanalyst can determine the colour order only if there are no byte sequences in the ciphertext pertaining to other colour orders. In the case of img_2, since the plain text is made of only two colours, there can be only two 1s in CBM, with huge runs of 0s. As a result, the CBM is composed of bytes with values 0, 1 and 255, making the probability of the occurrence of both the above said bytes 00101100 and 01100100, almost 0.

It is impossible to derive the absolute positions of $C1$ and $C2$ because of the reason explained in the previous example, that multiple choices of RP_1 for decryption result in img_2.

Though the previous attack could not reveal the absolute positions of $C1$ and $C2$, it reveals their relative positions. The relative position of a colour $C3$ with respect to $C1$ and $C2$ can be derived if an image img_3 with three pixels bearing $C1$, $C2$ and $C3$ is cryptanalysed. The following maps the various Pos values with the relative positions of the colours, assuming $C2$ occurs before $C1$.

1) Colour order $C3\ C2\ C1$, Pos : 01010011 00010000
2) Colour order $C2\ C3\ C1$, Pos : 00110101 00010000
3) Colour order $C2\ C1\ C3$, Pos : 00110001 01010000

For a given order of $C1$ and $C2$, $C3$ can take three different positions resulting in three different colour patterns and Pos vectors. The cryptanalyst has to search for any of the byte patterns to derive the relative positions of the three colours.

It shall be noted that usage of an image with only $C1$ and $C3$ or $C2$ and $C3$ would have revealed $C3$'s relative position with respect to only $C1$ or $C2$ respectively, while it is important to find the relative position of $C3$ w.r.t both $C1$ and $C2$. Hence, it is necessary to consider all the other $n-1$ colours when the relative position of a colour Cn in the permutation is being determined.

The above process can be extended for all the 2^{24} colours of RP_1, which in totality, reveals the absolute positions of all the colours of RP_1. But, as the number of colours and pixels increase in the plaintext, this attack decays into a *Brute Force Attack*. This is because of the following reasons.

1) As the number of colours and the pixels increase, the entries in Pos for each pixel spans more than a byte increasing the number, and the complexity of a byte pattern search that reveals the colour order.
2) The number of 1s in the CBM increases resulting in non-standard (values other than 0, 1 and 255) byte entries which further confuse the cryptanalyst in uniquely determining the byte patterns.
3) There are n different combinations of byte patterns that needs to be analysed to uniquely determine the relative position of the n^{th} colour w.r.t $n-1$ other colours whose relative positions have already been determined, while it requires to analyse $n!$ combinations of byte patterns, to reveal the relative positions of n colours, if the relative order of any of the $n-1$ colours is not known. The cryptanalyst is forced to cryptanalyse sequentially.
4) There is a high probability that a byte pattern specifying multiple colour orders exist in the ciphertext due to increase in the number of entries of the Pos vector and non-standard byte values in CBM. This confuses the cryptanalyst as to which order to choose to continue cryptanalysis.
5) The cryptanalyst not only has to analyse the ciphertext to reveal the colour order RP_1 used, but has to further verify for byte patterns corresponding to other colour orders to indicate or invalidate their presence.

Hence, as the colours and pixels increase, the cryptanalytic complexity converges to that of a *Brute Force Attack* and in most cases fail to provide valid encryption information due to presence of bytes that specify multiple colour orders. The above technique requires $2^{24}-1$ cryptanalytic attempts to reveal RP_1, each attempt being higher in complexity than the previous one. The number of combinations of byte patterns that needs to be analysed to reveal the colour permutation is given by $A = 2 + 3 + \ldots + 2^{24}$.

The above attack can be extended for m permutations by generating a plaintext where each colour is borne by m pixels. But the analysis is rendered useless since the revealed permutations are plaintext dependent and can not be used for other images.

4.4 Known Plain Text Attack

The goals and limitations of *CPTA* are applicable to *Known Plain Text Attack KPTA*. In a *KPTA* the cryptanalyst has the access to a plaintext image img and its corresponding ciphertext img'. The cryptanalyst has the knowledge of width W and height H, the colour value of $W \times H$ pixels and hence NRP.

Among the $((f!)-1)$ different ways of arranging the ciphertext bytes, the reverse transposition results which satisfy the following conditions, are candidates for the vector PB.

1) The 4 size bytes extracted have values of W_1, W_2, H_1 and H_2.
2) Within the extracted Pos vector, $w_k < W$ and $h_k < H$ of Pos_k, $\forall\ 0 \leq k < (W \times H)$
3) The first noc entries in Pos should provide position information of pixels bearing noc distinct colours
4) The extracted and decoded tuple CBM is composed of exactly 2^{24} bits.
5) The tuple CBM has noc number of 1s.

As indicated earlier, w_k and h_k values are not aligned to byte boundaries. To verify conditions 2 and 3, bits have to be extracted from multiple bytes involving huge number of bitwise operations and integer comparisons. It requires 4 integer comparisons per Pos entry to verify if condition 2 and 3 are satisfied and 1 comparison for the first noc entries to verify condition 3. Since there are $W \times H$ Pos entries, the number of integer comparisons IC is

$$IC = 4\,(W \times H) + noc$$

Verifying if a reverse transposition result is a valid candidate involves huge bit processing and integer comparisons.

Since there can be multiple size byte sets for a reverse transposition result, for a worst case, the above verification has to be carried out for each such size byte choice of all the $(f)!-1$ possibilities. If for a plaintext image, B is the number of ways in which the W_1, W_2, H_1 and H_2 can be chosen, and Δ is the time to verify all the conditions for an attempt, then approximately

$$(((f)! - 1) \times 2B \times \Delta) / 2$$

time units are required to derive the valid candidates for PB. The time Δ is halved since not all attempts require complete verification.

Since in this attack the colour of all the pixels is known, CBM and padding bytes can be discarded. Hence all candidates of PB directly translates to candidates of Pos. Let $n < (W \times H)!$ be the number of candidates for vector Pos. Each of the n candidates for vector Pos result in n different pixel orders and hence n different colour orders for permutations RP_1 through RP_{NRP}. There is no deterministic criteria for accurately deriving the colour orders, because, irrespective of the order of the Pos entries, all the candidates result in correct decryption as the pixel colours are already known. Hence, $KPTA$ results in multiple valid colour orders and fails to generate valid encryption information.

It is seen that the *KPTA* is unable to accurately determine the permutations RP_1 through RP_{NRP} as it results multiple colour orders for a given plain image. Hence, *KPTA* is rendered useless since the very purpose of the attack is defeated. Also, even if the cryptanalyst succeeds in deriving the set of permutations used for encryption, the same set cannot be used to decrypt other images as the permutations are plaintext dependent.

A minor change in the algorithm greatly increases the cryptanalytic complexity. Based on a random bit pattern of $W \times H$ bits, the decision of placing the pixel width or the height as the first entity of each of the $(W \times H)$ Pos entries is made. For example, bit value 1 for k^{th} bit, results in $Pos_k = (h_k, w_k, (1/0))$ and $Pos_k = (w_k, h_k, (1/0))$, for value 0. This enhancement results in huge increase in the cryptanalytic complexity.

5 Strength of the Algorithm

The previous sections describe the number of unit operations and the time required to decrypt the ciphertext based on the availability of plaintext information and resources for cryptanalysis. In the process, they also provide the *Objectival Strength* in terms of mathematical equations. This section provides the *Subjectival Strength* of the algorithm based on the properties of Pixel Property Separation.

Pixel Property Separation: Consider that the cryptanalyst is in possession of $key2$. Then, the reverse transposition and the extraction of the size bytes can be successfully carried out, resulting in the vector PB. The vector PB contains the pixel position information and a combination of 1s and 0s indicating the presence of any of the 2^{24} colours, as part of CBM. Each entry of the pixel position information Pos, is composed of entities to describe only the positions of the pixel but not its colour. The entire Pos vector consists of the position values of all the $W \times H$ pixels in the plaintext. This information is obvious since the cryptanalyst at this stage has the knowledge of W and H, and can derive the positions of all the pixels in the image. Since the vector CBM is composed of bit values, it conveys no information as to the actual colours of the plaintext. Most part of the vector PB contains information which is not of much use to the cryptanalyst. If $key1$ is compromised, the cryptanalyst can only derive RP_1 since it is composed of 2^{24} elements. It is not possible to derive the rest of the permutations as they are plaintext dependent. Without $key2$, the ciphertext can not be reverse transposed. Also, no information regarding the values of the 4 size bytes can be derived. Without the size of the plaintext it is not possible to derive the Pos and CBM vectors, making it impossible to discern the pixel position-colour association.

Plaintext Size Concealment: The size of the plain image is concealed by randomly placing the size bytes within the ciphertext. It is clear from the earlier sections that the cryptanalytic complexity greatly increases because of non-availability and the random placement of the original width and height.

Plaintext Dependent Permutation Generation: It is noted from Section 3 that NRP and the permutations RP_2 through RP_{NRP} are plaintext dependent. The composition of these permutations depend on the number of colours available for processing. This renders the technique robust against plaintext attacks as described in the previous sections.

Cryptanalytic Error Avalanche Effect (CEAE): The $CEAE$ results in a decrypted image with huge errors for a small error during any stage of the decryption process. Since the entries of the vectors Pos and CBM are closely packed, a small error manifests into considerable deviations of the decrypted image from the original plaintext. For example, for a 512×512 image, 9 bits are required to represent any width or height. Figure 3a depicts a byte pattern in Pos of such an image and their usage in creating Pos entries. During decryption, the above byte pattern results in the vector $Pos = [\ldots, (2, 183, 0), (305, 235, 0), \ldots]$. But, if the second byte is missing either due an error in reverse transposition or an error in choosing the size bytes, there is a huge difference in the entries of Pos. Figure 3b illustrates the vector with the

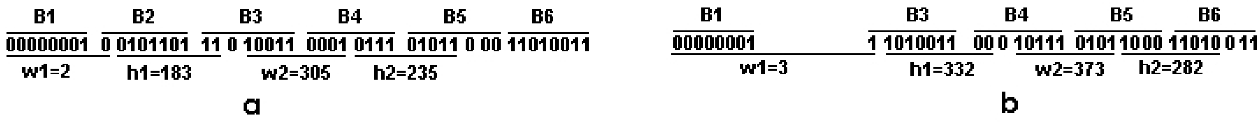

Fig. 3: Cryptanalytic Error Avalanche Effect - a. Original b. Erroneous

byte B2 missing. The erroneous byte pattern results in $Pos = [\ldots, (3, 332, 0), (373, 282, 0), \ldots]$. It shall be noted that with such a decryption error, the number of bits retrieved from a byte and the purpose of the retrieval is disturbed, which continues until all the $W \times H$ entries are retrieved. In the process, some of the bits from CBM end up in the Pos entries, resulting in a huge error both in Pos and in CBM.

Consider the vector, $CBM : [1, 167, 25, 34, 123 \ldots]$. The vector indicates that the first 167 bits in CBM are 1s, followed by runs of 25 0s, 34 1s and 123 0s. If the byte with value 167 is missed due to reasons stated above, the semantics of the vector is completely altered. The modified vector is composed of a first run of 25 1s, followed by 34 0s and 123 1s, indicating the presence of non existing colours and invalidating existing colours, disturbing the colour-position correlation in the decrypted image.

Confusion and Diffusion: The proposed algorithm is characterised by good *Diffusion* of plaintext bits in the ciphertext. If a bit in the plaintext changes, the colour of exactly one pixel changes from C_o to C_n. This replaces an entry in the Pos vector. The position of the new entry depends on whether C_n already exists in the plain image or is a new colour. If C_n is a new colour, then the entry is made within the first noc (number of distinct colours in plain image) entries in Pos. The entry corresponding to C_o is removed. The insertion and removal of entries has a cascading effect resulting in a shift of the subsequent entries. If C_n is not already present in the plaintext, CBM is also altered to accommodate C_n, changing exactly 2 bits. Even if only two bits are altered in CBM, the run composition is altered resulting in a considerable change. Such an effect may induce a change in W' and H'. These bit alterations are randomly dispersed in the entire ciphertext by the final byte transposition. *Strict Plaintext Avalanche Criterion (SPAC)*, for a fixed key to satisfy, each bit of the ciphertext block changes with the probability of one half, whenever any bit of the plaintext block is complemented. The insertion - removal of entries in Pos and the bit run alteration in CBM affect an approximate 50% of bits, satisfying *SPAC*.

Differential Cryptanalysis attempts to develop a pattern or a relationship between unit changes in plaintext and the ciphertext. Higher the diffusion, higher is the complexity of a differential attack. A good measure of diffusion is the *Percentage Inequality PI* of two ciphertexts *img′* and *img″*, the former derived from the original plaintext *img* and the latter by an unit change on *img*. The following equation defines PI,

$$PI = (NDP \times 100) \,/\, (W \times H) \tag{2}$$

where NDP is the number of pixels that differ in *img″* as compared to the corresponding pixels in *img′*. If the value of PI between two ciphertexts is high, then *Diffusion* is said to be high. Also, if this value, for various such ciphertext pairs changes randomly without exhibiting a pattern, a differential attack becomes practically impossible. Figure 4a depicts *PI* values for 20 different ciphertexts measured against

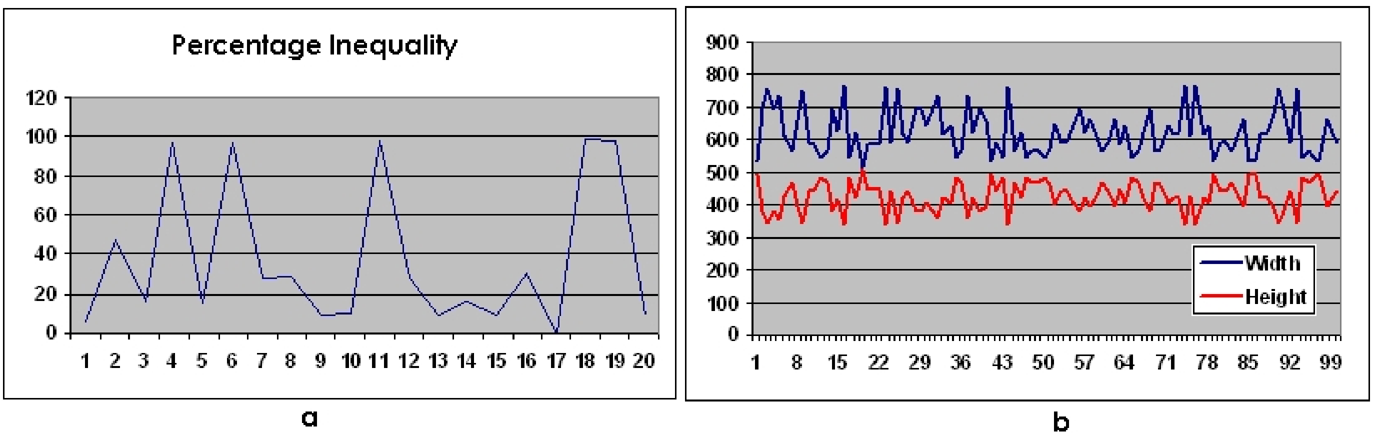

Fig.4: a. Percentage Inequality b. Variation of W' and H' against $key1$ for Lena Image

the plaintext-ciphertext pair in Figure 5a-5b. It shall be noted that the variation of PI ranges from 0.3% to 97% and that it does not follow any pattern.

While Diffusion complicates the relationship between the plaintext and the ciphertext, *Confusion* achieves the same between the ciphertext and the encryption key. In the proposed scheme, the key is not directly operated on the plaintext bits, rather used as seeds to generate random permutations. The random permutations are used as subkeys to arrange pixel position information and generate colour bit map, altering the basic structure of the plaintext. The final random transposition of ciphertext bytes obfuscates the relation between pixel position ordering on $key1$. Cryptographically Secure Random Permutations are characterised by a huge change in the permutation order, for a small change in the seed, complicating the Key-Ciphertext relationship. Also, it has been discussed that retrieval of keys from the permutations RPs, TP and SP is practically impossible. This complicates the key-ciphertext relationship, resulting in high degree of Confusion. *Strict Key Avalanche Criterion (SKAC)*, for a fixed plaintext block, each bit of the ciphertext block changes with a probability of one half when any bit of the key changes. A one bit change in the key, would generate totally unrelated random permutations, resulting in a huge change in the ciphertext, satisfying *SKAC*. As $key1$ changes, bit pattern in the CBM changes altering the content and the size of the ciphertext. This completely alters the values of W' and H'. Figure 4b illustrates the change in W' and H' due to unit changes in $key1$. It can be noted that the change in W' and H' does not follow any pattern.

6 Simulation Results

This section provides the experimental results of a basic-unoptimised implementation of the proposed scheme. The experiments were conducted on a 1.83GHz Intel Centrino Duo Processor with 1GB memory.

Figure 5 depicts the input image 'Lena' and its encrypted form. The values of $key1$ and $key2$ are 0.298760 and 0.514984 respectively. It shall be noted that the encrypted image (b) suggest the change in the width and height of the ciphertext. Figures Rp and Re, Gp and Ge, Bp and Be illustrate the histograms of the R, G and B components respectively of the plaintext and ciphertext. It shall be noted that the histograms of the encrypted image suggest an uniform distribution and significant difference of the RGB components of the ciphertext as compared to the plaintext, a desirable property of any encryption scheme.

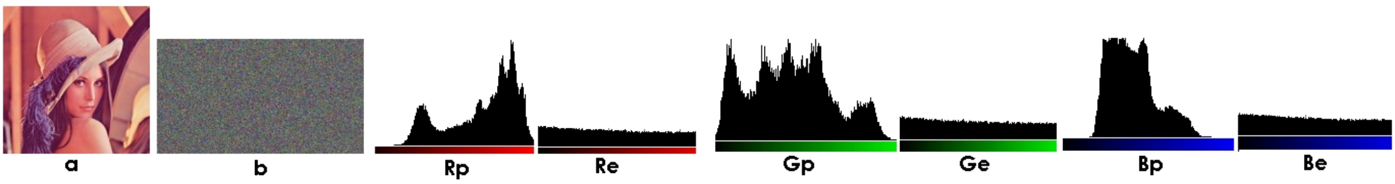

Fig 5: Lena - Plain Image, Cipher Image and RGB Histogram of plain and cipher image

Images of different width and height combinations have been considered for experiments, each being encrypted with a different set of keys. Table 1 presents the plaintext and the ciphertext properties, the encryption time (ET) and the decryption time (DT) in seconds for a selected set of images with different size and NRP requirements.

Img	$W \times H$	$W' \times H'$	NRP	p	ET	DT
1	512×512	535×491	21	0	5.0	4.7
2	640×480	602×507	703	2	5.5	4.8
3	1024×786	1141×658	1991	1	6.4	5.4
4	2816×2112	2841×2291	258	41	19.5	14.8
5	3648×2736	4395×2404	5712	17	22.0	19.1

Table 1 - Encryption - Decryption Time

7 Conclusion and Future Work

In this paper, a new symmetric encryption technique based on Pixel Property Separation has been proposed. The cryptanalytic complexities of various attacks also have been discussed. Together with the objectival strength in terms of mathematical equations, the paper also presents subjectival strength based on the principles of Pixel Property Separation. The results show that the algorithm possesses high security and is characterised by good confusion and diffusion. Plaintext attacks fail to provide valid encryption information and these attacks are rendered useless because of the usage of plaintext dependent Random Permutations. Given its high cryptanalytic complexity, the proposed technique is particularly suitable for applications involving secure storage and transmission of sensitive data with longer lifetime. Also, the algorithm is not limited to images, it can be easily adapted for any kind of digital data.

The flavour of the algorithm specified in this paper assumes a 24 bit RGB colour. Future work includes research on the number of bits used to represent a colour. 24 bit values need not always result in optimal ciphertexts when the algorithm is applied for digital data other than images. Also,the enhancement proposed as part of Section 4 can be analysed to derive the factor by which the cryptanalytic complexity is increased.

APPENDIX

DERIVATION: NRP

NRP can be mathematically defined as follows. The minimum value of NRP is 1. This is because, even if all the pixels bear the same colour (when no colour order can be defined), RP_1 is necessary to create CBM. In case of multicoloured images, If the pixels are grouped based on the colour and sorted on the group size, then NRP is the size of the group second in the list starting from the maximum size. If there are multiple sets with the maximum number of pixels, then NRP is the size of one such set. If $a\,(i)$, $\forall\ 0 \le i < 2^{24}$ RGB colours, denotes the number of pixels of colour i, then NRP is given by the following.

1) $pix_{max} = \max\left\{a\,(0)\,, a\,(1)\,, \ldots, a\left(2^{24}-1\right)\right\}$
2) Multiset $U = \left\{i : a\,(i) = pix_{max}, \forall\ 0 \le i < 2^{24}\right\}$
3) $pix_{smax} = \max\left(\left\{a\,(i)\,, _{\forall\ 0\ \le\ i\ <\ 2^{24}}\right\} - \{pix_{max}\}\right)$
4)

$$NRP = \begin{cases} pix_{max}, size\,(U) \ge 2 \\ pix_{smax}, size\,(U) = 1 \\ 1, otherwise \end{cases}$$

MATHEMATICAL MODEL OF ENCRYPTION ALGORITHM

The encryption algorithm encrypts plain image *img* with width W, height H and $(W \times H) \ge 1$. *img* (x, y), $0 \le x < W$, $0 \le y < H$, be the 24 bit *RGB* colour level of *img* at the position (x, y). The scheme is a 2 key *Secret Key Algorithm*. It operates on the input plain image *img*, the two keys $key1$ and $key2$ and generates the encrypted image img' with $(W' \times H')$ pixels. The proposed encryption scheme is defined as follows. Note that the pixels of the plain image are always scanned from *Left* to *Right* and from *Top* to *Bottom*, as in a *Raster Scan*. Let

1) noc represent the number of distinct colours in plain image *img*
2) function b indicate the presence of any colour C in *img*. Value $b\,(C)$, $\forall\ 0 \le C < 2^{24}$ colour values, is '1' if at least one pixel in *img* bears C, '0', otherwise.
3) function q be such that $q\,(C)$, $\forall\ 0 \le C < 2^{24}$ RGB levels, denotes a number *ne*, such that, at any point in the encryption process, exactly *ne* number of pixels of colour C have been placed in the Pos vector. Note that, $(0 \le q\,(C) \le a\,(C))$ when $(b\,(C) = 1)$ and $(q\,(C) = 0)$ when $(b\,(C) = 0)$
4) function $GetCol$ be such that for any integer $i \ge 0$, $GetCol\,(i)$ returns the $(i+1)^{th}$ colour C_i from the vector CV (depicted in Figure 1), such that there exists atleast one pixel bearing C_i, which is yet to be placed in the Pos vector.
5) function $GetPos$ be such that for any colour $0 \le C < 2^{24}$, $GetPos\,(C, k)$ returns Pos_k, the $(k+1)^{th}$ entry of the Pos vector, composed of (x, y, flg) values of a pixel bearing colour C. This function is defined as follows:

   1. $q\,(C) = q\,(C) + 1$, if $(b\,(C) = 1) \wedge (q\,(C) \le a\,(C))$
   2. $noc = noc - 1$, if $(b\,(C) = 1) \wedge (q\,(C) = a\,(C))$
   3. $Pos_k = \begin{cases} \left(w_{q(C)}, h_{q(C)}, 0\right), if\ \ (b\,(C) = 1) \wedge (q\,(C) < a\,(C)) \\ \left(w_{q(C)}, h_{q(C)}, 1\right), if\ \ (b\,(C) = 1) \wedge (q\,(C) = a\,(C)) \end{cases}$

   where $w_{q(C)}$ and $h_{q(C)}$ represent respectively the width w and the height h of the $q\,(C)^{th}$ pixel such that $img\left(w_{q(C)}, h_{q(C)}\right) = C$

The Encryption Algorithm is defined as follows:

Step 1: Create the Pos vector $[Pos_0, Pos_1, \ldots, Pos_{W \times H-1}]$ by the following three steps. Initially, $k = 0$, $n = 2$. k is incremented after each Pos entry

1. $Pos_k = GetPos\,(CV\,[RP_1\,[j]]\,, k)\,, \forall\ 0\ \le j <\ 2^{24}$, where $RP_1 = RP\left(key1, 0, 2^{24}\right)$
2. $Pos_k = GetPos\,(GetCol\,(RP_n\,[j])\,, k)\,, (\forall\ 0 \le j < noc)$ and $(\forall\ 2 \le n < NRP)$, where any $RP_n = RP\,(key1, n, noc)$
3. $Pos_k = GetPos\,(GetCol\,(0)\,, k)$, $\forall\ k < (W \times H)$

Step 2: Create $CBM = [B_0, B_1, \ldots, B_{2^{24}-1}]$, where any $B_j = b(RP_1[j]), \forall\, 0 \le j < 2^{24}$

Step 3: Assuming Pos and CBM are digitally represented, Create $PB = [Pos, CBM]$. Let pb be the size of vector PB.

Step 4: Let (p) be the number of padding bytes in the encrypted image. The padding bytes are given by the vector $P = [P_0, P_1, \ldots, P_{p-1}]$. Create $PB' = [PB, P_0, P_1, \ldots, P_{p-1}]$. $(pb + p)$ gives the size of PB'

Step 5: Insert W_1, H_1, W_2 and H_2 within vector PB', at positions given by $SP[0]$, $SP[1]$, $SP[2]$ and $SP[3]$ respectively, to create vector F', composed of $(pb + p + 4)$ bytes. $SP = RP(key2, 0, (pb + p))$

Step 6: Shuffle F' using permutation TP to create $F = [F_0, F_1, \ldots, F_{f-1}]$, where any $F_i = F'_{TP[i]}$. Choose W' and H' such that $f = (W' \times H' \times 3) = (pb + p + 4)$. $TP = RP(key2, 1, f)$

Step 7: Create encrypted image img' of size $W' \times H'$,

$$img'(x, y) = [F_i, F_{i+1}, F_{i+2}], \text{ where } i = (x \times W' \times 3) + (y \times 3), \forall\, 0 \le x < W', 0 \le y < H'$$

$img'(x, y)$ is represented as a 24 bit number with F_i being the MSB. Number of pixels in img' is $nop = (W' \times H')/3$.

Step 8: Choose any two integral factors nop_1 and nop_2 of nop such that $(nop_1 \times nop_2) = nop$. Then $W' = nop_1$ and $H' = nop_2$. Note that this step is optional.

DERIVATION: M, T AND MT

Number of invalid size byte sets that can be chosen from a bin of f bytes, of which z bytes carry value 0 and u bytes carry value greater than threshold t is derived as follows.

Let B_n be the n^{th} byte chosen. Note that $B_1 = W_1$, $B_2 = H_2$, $B_3 = W_2$ and $B_4 = H_2$. Let Z represent a byte with value 0, and N, a byte with a non-zero positive value. It shall be noted that $m = (f - z)$ are the number of non-zero bytes available. There are (z) ways to choose the first Z, $(z - 1)$ ways for the second and so on. Similarly (m) ways for the first N, $(m - 1)$ ways for the second and so on.

Since 4 bytes have to be randomly chosen from the bin, there can be 16 distinct possibilities, of which some are invalid. Byte combinations which have both $B_1 = 0$ and $B_3 = 0$ or both $B_2 = 0$ and $B_4 = 0$ result in a value 0 for W or H respectively. Such combinations shall be discarded. Invalid byte combination $B_1B_3B_2B_4$ that result in value W or H = 0 and the number of ways *Num* in which each invalid combination be chosen as a size byte set, has been depicted in the Table 2.

$B_1B_3B_2B_4$	*Num*
$ZZZZ$	$(z)(z-1)(z-2)(z-3)$
$ZZZN$	$(z)(z-1)(z-2)(m)$
$ZZNZ$	$(z)(z-1)(m)(z-2)$
$ZZNN$	$(z)(z-1)(m)(m-1)$
$ZNZZ$	$(z)(m)(z-1)(z-2)$
$NZZZ$	$(m)(z)(z-1)(z-2)$
$NNZZ$	$(m)(m-1)(z)(z-1)$

Table 2. No. of invalid size byte choices with $H = 0$ or $W = 0$

Summing the invalids, the number of 4 byte sets M, resulting in a value 0 for both or any of W and H is

$$M = {}^{z}P_4 + 4(f - z)({}^{z}P_3) + 2({}^{z}P_2)({}^{(f-z)}P_2)$$

On the same lines, the number of 4 byte sets T, resulting in $W_2 > t$ or $H_2 > t$ can be derived. If L represents a byte $B < t$, and G, a byte $B > t$, all byte combinations which have $B_3 > t$ or $B_4 > t$ like *LGGG* and *LLLG* are invalid. Adding all such invalid combinations, the value of T is

$${}^{u}P_4 + 4(f - u)({}^{u}P_3) + 5({}^{u}P_2)({}^{(f-u)}P_2) + 2(u)({}^{(f-u)}P_3)$$

Byte combinations which qualify under both the above mentioned categories of invalid size byte sets, like *ZZLG*, are invalidated twice. The number of such combinations *MT* have to be subtracted once from $M + T$. *MT* is defined by the following equation.

$$MT = 2(u)(f - u)({}^{z}P_2) + 2({}^{z}P_2)({}^{u}P_2)$$

References

[1] W. Stallings, *Cryptography and Network Security principles and practices*, third ed.: Pearson Education, 2003.

[2] D. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Reading, MA: Addison-Wesley, 1998.

[3] N.D. Jorstad, L.T. Smith, Jr., "Cryptographic Algorithm Metrics," *Institute for Defense Analyses Science and Technology Division* , January. 1997.

[4] C. Shannon, "Communication Theory of Secrecy Systems," *Bell Systems Technical Journal*, no. 4, 1949.

[5] J. Cheng; J.I. Guo, "A new chaotic key-based design for image encryption and decryption," *The 2000 IEEE International Symposium on Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva,* vol.4, no. 4, pp. 49 - 52, May. 2000.

[6] D. A. Osvik, A. Shamir, E. Tromer, *Cache Attacks and Countermeasures: the Case of AES*, November, 2005. available at http://theory.csail.mit.edu/t̃romer/papers/cache.pdf.

[7] SJ. Li and X. Zheng, "Cryptanalysis of a chaotic image encryption method," *The 2002 IEEE International Symposium on Circuits and Systems Proceedings 2002. ISCAS 2002 Scottsdale, Arizona*, vol. 1, 2006

[8] R. B. Davies, *Newran02C - a random number generator library*, April, 2006. available at http://www.robertnz.net/.

[9] Anatoliy Kuznetsov, *D-Gap Compression*, 2002. available at http://bmagic.sourceforge.net/dGap.html.

[10] S.K. Sahni, *Data Structures, Algorithms, and Applications in C++,*, Second Edition, McGraw Hill, NY, 2005.

[11] Shiguo Lion, Jinsheng Sun and Zhiquan Wang, "A Novel Image Encryption Scheme Based-on JPEG Encoding," *Proceedings of the Eighth International Conference on Information Visualisation (IV04),* 2004

[12] J.A. Munoz Rodrguez and R. Rodrguez-Vera, "Image encryption based on phase encoding by means of a fringe pattern and computational algorithms," *REVISTA MEXICANA DE FISICA 52 (1),* pp. 53-63, Feb. 2004

[13] A. Mitra, Y. V. Subba Rao and S. R. M. Prasanna, "A New Image Encryption Approach using Combinational Permutation Techniques," *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE,* vol. 1, no. 5, 2006

[14] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions Information Theory,* vol. 22, no. 6, pp. 644-654, Nov. 1976.

[15] P. P. Dang and P. M. Chau, "Image Encryption for Secure Internet Multimedia Applications," *IEEE Transactions Consumer Electronics,* vol. 46, no. 3, pp. 395-403, Aug. 2000.

[16] R. B. Davies, *COPACOBANA, a $10,000 DES cracker based on FPGAs by the Universities of Bochum and Kiel*, December, 2006. available at http://www.copacobana.org/.

[17] A. Fuster and L. J. Garcia, "An Efficient Algorithm to Generate Binary Sequences for Cryptographic Purposes," *Theoretical Computer Science* 259, pp. 679-688, 2001.

[18] Ronald L. Rivest, "The RC5 Encryption Algorithm,", March. 1997.

[19] Min-Sung Koh, Esteban Rodriguez-Marek, Claudio Talarico, *A NOVEL DATA DEPENDENT MULTIMEDIA ENCRYPTION ALGORITHM SECURE AGAINST CHOSEN-PLAINTEXT ATTACKS*, ICME 2007.

Karthik Chandrashekar Iyer received the Bachelor of Engineering degree in Computer Science from the Vishweshwaraya Technological University, Karnataka, India in 2004. Currently, he is a Senior Engineer in the DSP Software Center, NXP Semiconductors (formerly Philips Semiconductors), Bangalore, India. His interests include Embedded Multimedia Systems, Digital Rights Management, Video Post Processing, Cryptography and Network Security.

Aravinda Subramanya received the Bachelor of Engineering degree in Computer Science from the Vishweshwaraya Technological University, Karnataka, India in 2004. Currently, he is a Senior Engineer in the DSP Software Center, NXP Semiconductors (Philips Semiconductors), Bangalore, India. His interests include Audio coding, Embedded Multimedia Systems, Vector Processors, Digital Rights Management, Cryptography and Network Security.