Digital Watermarking through Embedding of Encrypted and Arithmetically Compressed Data into Image using Variable-Length Key

Sabyasachi Samanta [1], Saurabh Dutta [2], Goutam Sanyal[3]

[1]Haldia Institute of Technology, Haldia, WB, INDIA
E-mail id: sabyasachi.smnt@gmail.com
[2]Dr. B. C. Roy Engineering College, Durgapur, WB, INDIA
E-mail id: saurabh.dutta@bcrec.org
[3]National Institute of Technology, Durgapur, WB, INDIA,
E-mail id: nitgsanyal@gmail.com

Abstract:

In this paper, we have encrypted a text to an array of data bits through arithmetic coding technique. For this, we have assigned a unique range for both, a number of characters and groups using those. Using unique range we may assign range only 10 characters. If we want to encrypt a large number of characters, then every character has to assign a range with their group range of hundred, thousand and so on. Long textual message which have to encrypt, is subdivided into a number of groups with few characters. Then the group of characters is encrypted into floating point numbers concurrently to their group range by using arithmetic coding, where they are automatically compressed. Depending on key, the data bits from text are placed to some suitable nonlinear pixel and bit positions about the image. In the proposed technique, the key length and the number of characters for any encryption process is both variable.

Key words:

pixel, invisible digital watermarking, arithmetic coding, symmetric key, nonlinear function.

1. Introduction

Digital Watermarking describes the way or technology by which anybody can hide information, for example a number or text, in digital media, such as images, video or audio. Arithmetic compression technique takes a stream of input symbols and replaces it with a single floating point number less than 1 and greater than or equal to 0. That single number can be uniquely decoded to exact the stream of symbols that went into its assembly. Most computers support floating point numbers of up to 80-bits or so. This means, it's better to finish encoding with 10 or 15 symbols. Also conversion of a floating point number to binary and reverse is maximum time erroneous. Arithmetic coding is best to accomplish using standard 16-bit and 32-bit integer mathematics.

$$\underbrace{11111111}_{\alpha}\quad \underbrace{11011100}_{R}\quad \underbrace{10101011}_{G}\quad \underbrace{11110000}_{B}$$

Figure 1.1 Bit Representation of a 32-bit RGB Pixel

A pixel with 32-bit color depth consists of α value, R (Red), G (Green) and B (Blue) value. α value is the value of opacity. If α is 00000000, the image will be fully transparent. Each of three(R, G & B) 8-bit blocks can range from 00000000 to 11111111(0 to 255) [1] [10] [11].

In this paper, we have proposed a technique, initially to encrypt each and every character of message and from that to a compressed stream of bits. First, we have assembled a table (Table 2.1.1.1) taking a number of characters or symbols available in keyboard or the special symbols as per user's prerequisite. Each character (Ch) is assigned a range (r_c) indicated by high (H_c) and low (L_c) range between 0-1. Then taking a number of characters (maximum 10), a group (G) is defined. Each group is also assigned a range (r_g) indicated by high (H_g) and low (L_g) range. Hence, in this way we can take maximum 100 characters to encrypt. Let if any body wants to encrypt all the characters of ASCII-8 table or if 100 characters are not enough for any encryption process. For that we have defined groups taking hundred characters in one unique range (r_h) and by this we may encrypt thousand of characters. Following the way we may put 1000 characters in one unique range (r_t) and so on. Every time the number of characters increases by 10 times by this process. As a consequence we may encrypt more characters. Then the long textual message is broken into a number of small messages. Every short message(less than or equal to 9 characters) is converted into a number of floating point numbers corresponding to their group range by arithmetic coding technique. The first floating point number is used for character range (i.e. taking r_c) and the next are for hundred and thousand characters group range and so on i.e. in which group range the character belongs to(r_g r_h, $r_{t,\ ......}$). Then we have transformed all the floating point numbers to unsigned long integer numbers (removing the floating point) and it to equivalent binary numbers. Here we have used the variable length key with the key length of 1 to 39 alphanumeric characters. The characters of the key which also be defined with their individual range in encryption table (Table 2.1.1.1). From that alphanumeric key we have generated a four digit key value (K_v). To get it we have applied the similar method as we have broken the long message into groups. From every group we get floating point numbers and from that to long integer numbers depending on number of characters encrypted through the process. Taking the first integer number we have generated a reminder by modulus division with 4-digit maximum number. Add that reminder with the integer number from that group (if any). Again calculate modulo division and continue that process for the integer numbers generated from the group. After calculating reminder from the previous group add it to the integer number of next group, if any. Apply the same method repetitively for the number of groups. If the reminder of modulo division becomes zero finally then take the divisor as key value (K_v). Taking that key value (K_v) we have placed the data bits form the text message in some suitable nonlinear pixel and bit positions. We have positioned the data bits in any one bit of last four significant bit of each R, G and B taking the α value as 255 or as it is in the original image [2] [3] [6] [7] [8] [9].

Example: A text with 24 characters (Let for an encryption process with maximum 100 characters), will be encrypted to an array of 168 (8+2*((2*30) + (1*20))) bits of stream. If we use ASCII-8 (American Standard Code for Information Interchange) to encode 200 (8+192) bits are required. In the above example, we have seen the data is automatically compressed by 32 bits using the encryption technique. An image with 800 X 600 has 2, 40,000 pixels. In our work only we are altering any one bit of last four significant bit of each R, G & B. Here maximum 56 pixels will be affected by this process. If any bit generated from characters become same to the targeted bit of image, then there will be no change.

Section 2 represents the scheme followed in the encryption technique**.** Section 3 represents an implementation of the technique. Section 4 is an analytical discussion on the technique. Section 5 draws a conclusion.

2. The Scheme

This section represents a description of the actual scheme used during implementing "Digital Watermarking through Embedding of Encrypted and Arithmetically Compressed Data into Image using Variable-Length Key" technique. Section 2.1 describes the encryption technique using four algorithms 2.1.1, 2.1.2, 2.1.3 and 2.1.4 while section 2.2 describes the decryption technique using two algorithms 2.2.1 and 2.2.2 [2] [4] [6].

2.1 Encryption of data bits about the image

2.1.1 Assignment of range for individual characters and groups

Step I: Take special characters/symbols or characters available in keyboard.
Step II: Count the number of characters (chlen) and calculate number of groups (nogp=chlen/10) and remaining characters (extch=chlen%10).

a) If (nogp>10) then assign range (r_h) for hundred characters.
b) If (nogp>100) then assign range (r_t) for thousand characters and so on.

Step III: Assign range (r_c) to each characters/symbols indicated by high (H_c) and low (L_c) range between 0 to 1.
Step IV: Taking a set of characters (maximum 10 characters) define a group range (r_g) by high (H_g) and low (L_g) range to each of these. Also assign range (r_h) for hundred characters , if number of character is more than 100. As well assign range for thousand groups (r_t) for more than 1000 characters and so on.

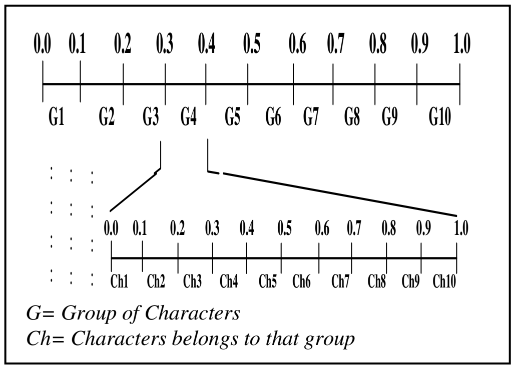
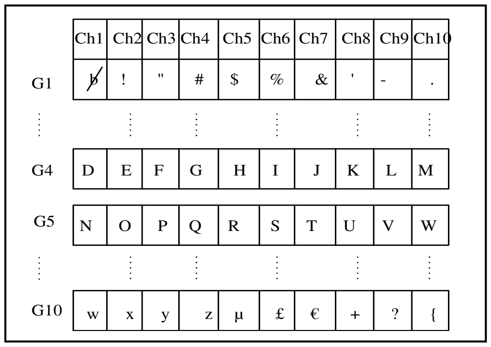


Figure 2.1.1.1 Subdivision of Encoded Characters



	Ch1	Ch2	Ch3	Ch4	Ch5	Ch6	Ch7	Ch8	Ch9	Ch10
G1	b̸	!	"	#	$	%	&	'	-	.
⋮		⋮	⋮		⋮		⋮		⋮	
G4	D	E	F	G	H	I	J	K	L	M
G5	N	O	P	Q	R	S	T	U	V	W
⋮		⋮	⋮		⋮		⋮		⋮	
G10	w	x	y	z	µ	£	€	+	?	{

Figure 2.1.1.2 Characters in Group

Table 2.1.1.1: Characters and Groups with Range

Char acters	**Range for Characters ($L_c \leq r_c < H_c$)**	**Range for Groups ($L_g \leq r_g < H_g$)**	**Range for Outer Groups ($L_t \leq r_t < H_t$)**
#	$0.3 \leq r_c < 0.4$	$0.0 \leq r_g < 0.1$	$0.0 < r_t < 0.1$
&	$0.6 \leq r_c < 0.7$	$0.0 \leq r_g < 0.1$	$0.0 < r_t < 0.1$
A	$0.7 \leq r_c < 0.8$	$0.2 \leq r_g < 0.3$	$0.0 < r_t < 0.1$
B	$0.8 \leq r_c < 0.9$	$0.2 \leq r_g < 0.3$	$0.0 < r_t < 0.1$
E	$0.1 \leq r_c <.0.2$	$0.3 \leq r_g < 0.4$	$0.0 < r_t < 0.1$
I	$0.5 \leq r_c <.0.6$	$0.3 \leq r_g < 0.4$	$0.0 < r_t < 0.1$
L	$0.8 \leq r_c <.0.9$	$0.3 \leq r_g < 0.4$	$0.0 < r_t < 0.1$
N	$0.0 \leq r_c <.0.1$	$0.4 \leq r_g < 0.5$	$0.0 < r_t < 0.1$
O	$0.1 \leq r_c < 0.2$	$0.4 \leq r_g < 0.5$	$0.0 < r_t < 0.1$
R	$0.4 \leq r_c < 0.5$	$0.4 \leq r_g < 0.5$	$0.0 < r_t < 0.1$
:	:	:	:

Step V: If extch =0 then repeat *Step II* to *Step IV* for i= *1* to *nogp*.
Otherwise repeat *Step II* to *Step IV* for i= 1 to (nogp+1).
Step VI: Stop.

2.1.2 Encode message using arithmetic coding and store it to encrypted array as binary values

Step I: Take characters as input from keyboard or special characters (which must be in Table 2.1.1.1).
Step II: Calculate the string length (chlen) from input.
Step III: Convert the length (chlen) into its 8-bit binary equivalent. Store that data bits to earr[bit] as LSB (Least Significant Bit) to earr[1] and MSB (Most Significant Bit) to earr[8] respectively.

Step IV: Calculate number of broken messages n = chlen/9 and remaining characters r =chlen%9.

Step V: Taking the range from Table 2.1.1.1 apply arithmetic coding technique to encode the character set into a single floating point number in between 0 - 1.

```
Set low to 0.0
Set high to 1.0
    While there are still input characters do
     get an input character
     code range = high - low.
     high = low + range*high_ range
     low = low + range*low_ range
     End of While
```

Continue this process to get input (1 to 9) for first n times and (1to r) at (n+1) th time and stop. Output the low.

Step VI: From the Low value and High value convert Low value (low) to an unsigned long integer number removing the floating point.

Step VII: Convert that number into equivalent binary number. For n times if total number of 0 or 1 is less than 30 then left fill with 0's. For the case of r:

a) If ($r \geq 6$ || $r=0$) as in Step VII.

b) If ($r \geq 3$ || $r<6$) and if total number of 0 or 1 is less than 20 then left fill with 0's.

c) If ($r>0$ || $r<3$) and if total number of 0 or 1 is less than 10 then left fill with 0's.

Step VIII: Store the binary values, LSB to earr[9] and rest to earr[bit] respectively.

Step IX: Repeat *Step V* to *Step VIII*

a) If *r=0* then for i= 1 to (m*n) times

Otherwise for i= 1 to (m*(n+1)) times taking ranges (r_c, r_g, r_h, r_t, …) separately and respectively.

Where m= number of simultaneous floating point number configuration for each and every groups.

Step X: Stop.

2.1.3 Selection of the nonlinear pixel positions from the key

Step I: Take key (*K*) input from keyboard.

Step II: Repeat *Step II* to *Step VI* of *Algorithm 2.1.2* to get the floating point number.

Step III: Taking the number getting from character range (r_c) calculate the reminder (r) by modulus division with 4-digit maximum number.

Step IV: Add the reminder (r) with the integer number calculated from group range (r_g) and again calculate the reminder (r) by following *Step III.*

Step V: To calculate through all the group of characters runs through *Step IX* of *algorithm 2.1.2.*

Step VI: Store the final reminder as key value (K_v).

a) If the value of (K_v) is 0 the then takes the value of Kv as 9999.

Step VII: Take the value of *bit* from array e*arr[bit]* to calculate total number of pixels is required as three following data bit replaced in R, G & B of every pixel. So calculate number of pixel p= (ceil (bit /3)).

Step VIII: Take the key value (K_v) and calculate the value of function

$F(x, y) = K_v{}^{p}$ [i.e. POW (K_v, p)].

Step IX: Store the exponential long double values into file one by one.

Step X: Repeat *Step III* to *Step IV* for i= (1 to p) and go to *Step VI.*
Step XI: Read the values as character up to "e" of the every line of the file and store it to another file with out taking the point [.].
Step XII: Modify the value as numeric and store it to an array *arrxyz[p]*
Step XIII: Take most three significant digit to *arrx[p],* next three digits to array *arry[p]* and last significant digit to *arrz[p].*
Step XIV: Repeat *Step VI* to *Step VIII* up to end of the file.
Step XV: Stop.

2.1.4 Replacement of the array elements with R, G & B values of pixels

Step I: Calculate the width (w) and height (h) of the image.
Step II: Set x=*arrx[p]* and *y=arry[p].*
Step III: To select the pixel position into image, compare the value of x and y with the value of w and h (where addressable pixel position is (0, 0) to (w-1, h-1)).

```
   a)   If (x >(w-1)) or (y >(h-1)) then
   Set P (x, y) = P (0+(x %( w-1)), (0 +(y %( h-1)))
        Otherwise Set    P (x, y) = (x, y).
```

Step IV: To select the bit position (b) of selected pixel i.e. with which bit the array data will be replaced. Set *z =arrz[p].*

```
        i)   If (z%4=0) then b=LSB
        ii)  If (z%4=1) then b=2nd LSB
        iii) If (z%4=2) then b=3rd LSB
        Otherwise b=4th LSB of each R, G & B of a pixel.
```

Step V: Verify the pixel or bit positions which previously have used or not about the image.

```
   a)  If ((P(x, y)= (P(x, y)) || P (x, y)= P (x++, y++)) && (b=b++])then
   Set P ((x, y), b) =P (0, h) and b as Step IV.
   Repeat Step V (a) for j=1 to p;
   Repeat Step V (a) for k=j to p.
   Go to Step VI.
```

Step VI: To replace the array elements with the selected bit position of selected pixel and to reform as a pixel

a) After reading the values of R, G & B convert each to its equivalent 8-bit binary values.
b) Replace subsequent element of earr[bit] by following *Step III to Step V.*
c) Taking values of R, G & B switch it to the pixel value and place it to its position of the image (taking α value as before).

Step VII: For replacing the array element to pixels using the above mentioned process starting from the 0th element up to the end of the array.

```
        A) If bit%3 = 0
             Go to Step VIII.
        B) If bit%3 = 1
```

for 0th element to (bit-1)th element of the array repeat *Step VII (A).*For (bit)th element to R, value for G and B will be remain same. And go to S*tep VIII.*

```
        C)  If  bit%3=2
```

for 0th element to (bit-2)th element of the array repeat *Step VII (A).*For (bit-1)th element to R, (bit)th to G and B will be remain same. And go to S*tep VIII.*
Step VIII: Repeat *Step II to Step VII* for i=1 to p.
Step IX: Stop.

2.2 Decryption of the data bits from the image

2.2.1 Retrieving the replaced bits from the encrypted image

Step I: Take the key (K) input as it was at the time of encryption.
Step II: To get the key value (K_v) go through *Step II* to *Step VI* of *algorithm 2.1.3.*
Step III: To get the pixel positions with in the image and bit position in R, G & B of selected pixels run through *Step VII* to *Step XIV* of *Algorithm 2.1.3* and *Step I* to *Step VIII* of *Algorithm 2.1.4.*
Step III: Retrieving the encrypted bits from the selected bit positions of delectated pixels store it to decrypted array from darr[1] to darr[bit] respectively.
Step IV: To get the length repeat *Step II to Step III* for i= 1 to 3 times (as every pixel contain three data bits).
Step V: Taking data bits of darr [1] as LSB and darr [8] as MSB calculate the length (chlen) of message.
Step VI: To find out the total number of bits in decoding array (i.e. value of *bit* in *darr[bit]*), calculate n=chlen/9 and r=chlen%9 (as in *Step IV Algorithm 2.1.2).*
Step VI: Taking the value of n and r calculate bit= (2*(30*n + rbit) +8).
Where *rbit* are calculated as

a) If ($r \geq 6$) then *rbit =30*
b) If($r \geq 3$ || $r<6$) then *rbit =20*
c) If($r>0$ || $r<3$) then *rbit=10 (as in Step VII algorithm 2.1.2).*

Step VII: Now go through *Step VII to Step XIV of Algorithm 2.1.3* and *Step I to Step VIII of Algorithm 2.1.4 for i=4 to p.* Store the data's *darr[10] to darr[bit]* respectively.
Step VIII: Stop.

2.2.2 Decompress array elements to text using decoding algorithm

Step I: To translate the data bits to floating point number

a) If $n \neq 0$, assign the value *darr[8+n*i]* to LSB and *darr[8+30*n]* to MSB respectively and covert it to equivalent decimal number.
   If the total number of digits of that decimal number is less than nine left filling with 0's translate it to nine digits floating point number.
b) If $r \neq 0$, assign the value *darr[(8+30*n) +1]* to LSB and da*rr[(8+30*n)+1)+rbit]* to MSB respectively and covert it to equivalent decimal number.
   If the total number of digits of that decimal number is less than *r* left filling with 0's translate it to *r* digits floating point number.

Step II: Apply the decoding algorithm of arithmetic coding to convert it into individual range.

```
Get encoded number
Do
Find range from the table
Output the range
        Subtract symbol low value from encoded   number
Divide encoded number by range
Until no more symbols or zero.
```

Step III: Comparing the ranges of first time iteration of *Step II* and second time iteration of *Step II* alongside (i.e. comparing the range r_c and the r_g at same time) from the table (Table 2.1.1.1) find out the encoded characters (Ch).

Step IV: Executing the work of *Step (a)* go to *Step (b)*

a) If n≠0 repeat *Step I (a)* to *Step III* for i= 1 to 2 times.

Repeat *Step IV for i=1* to *n* times.

b) If r≠0 repeat *Step I (b) to Step IV* for i=1 to 2 times. And Go to *Step VII.*

Step V: Finally put the characters one by one and assemble the original message.

Step VI: Stop.

3. An implementation

Let the message to be encrypt is NONLINEAR.

So the length of the message

=09(Decimal equivalent)

=00001001(8 Bit Binary equivalent)

All the characters of the message are defined in the encryption table (Table2.1.1.1) with their distinct ranges. And let the table encrypted with 100 characters. Starting from the range 0.0 to 0.1 *Ch1's* (maximum10 characters from *G1* to *G10*) are defined. In that range the first character N is also defined. Applying the technique described in algorithm 2.1.2 we get the codeword for the r_c as,

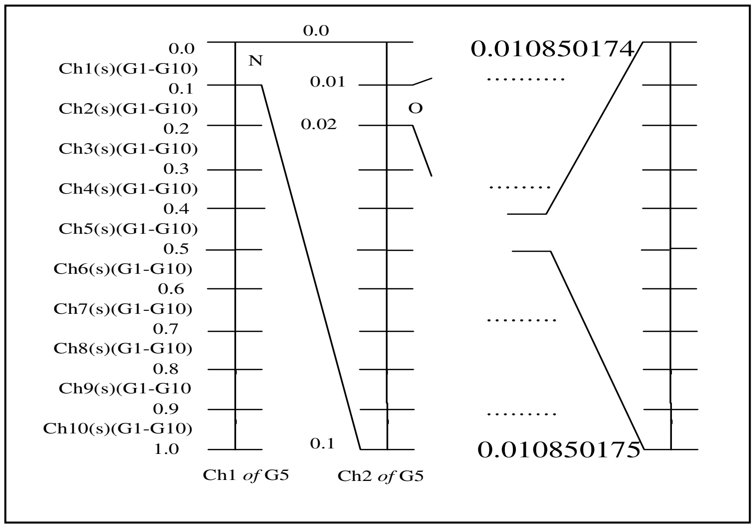


Figure3.1: Generation of Codeword using r_c

Hence we get the codeword for characters0.010850174 ≤codeword_for_Ch< 0.01050175.

And for groups we get 0.444334324 ≤codeword_for_group <0.444334325

Taking the low values for characters

Codeword

=0.010850174(floating point number)

=10850174(integer number removing floating point)

=000000101001011000111101111110 (30 bit binary equivalent)

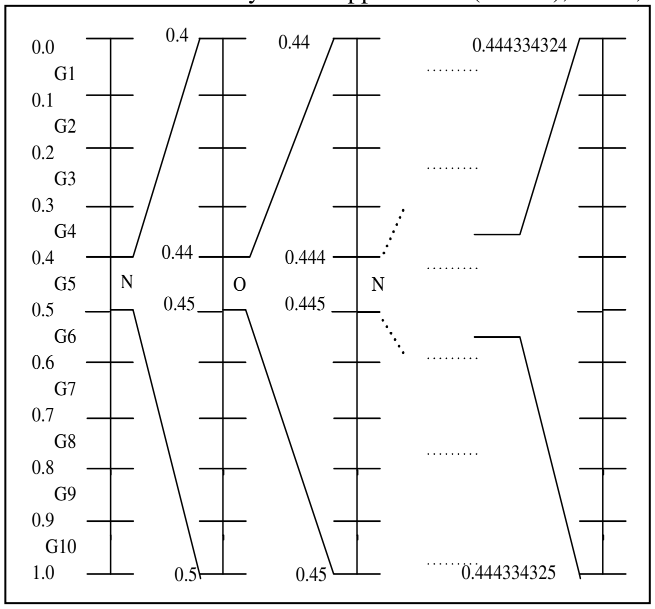


Figure 3.2: Generation of Codeword using r_g

And taking low value for group we get codeword=0.444334324 (floating point number)
=444334324 (integer number removing floating point)
=011010011111000000000011110100(30 bit binary equivalent).
First store the bits form length to encrypted array earr[bit], then store bits of stream from code words respectively as,

bits for length	*bits for character*	*bits for group*
earr[1]=1	earr[9]=0	earr[39]=0
: :	: :	: :
earr[8]=0	earr[38]=0	earr[68]=0

Figure 3.3 Data bits in Encrypted Array

Let the key (K) =6M3U7NU9.
From Key (K), for r_c we get,0.25933635(floating point number)
=25933635 (integer number removing floating point)
For r_g, we get, 0.24152452(floating point number)
=24152452 (integer number removing floating point)
Now the reminder (r) =mod (25933635, 9999) =6228.
Adding r with the second integer we get = (6228+24152452) = 24158680.
Again the reminder (r) = 1096.
As their only 8-alphanumeric characters in key (K) finally we get the key value (K_v) =1096.
The image size= 800 X 600(w x h).

Number of affected pixel (p) = [ceil (68/3)] =23

In the Table-3.1 how the array data's are replaced with R, G & B values in selected nonlinear pixels of an image is described (as described in algorithm 2.1.3 and algorithm 2.1.4).

Table 3.1: Replacement of Data Bits about Image

Key(K),i	Value	Value of pixel P(x,y)	Bit position b= Z%4	Array data to replace
1096,1	1.096000 e+03	P(109,600)	1st LSB	earr[1] earr[2] earr[3]
:	:	:	:	:
1096,5	1.581440e+15	P(158,144)	1st LSB	earr[13] earr[14] earr[15]
:	:	:	:	:
1096,23	8.234605e+69	P(223,005)	2nd LSB	earr[79] earr[80] B as same

Thus we can transmit the encrypted watermarked image through any communication channel. Afterward applying the decryption technique as described in algorithm 2.2.1 and algorithm 2.2.2 we will be able to get back the encrypted message from that watermarked image in the decryption end. Taking character range and group range we get the encrypted characters which are defined in Table 3.2.

Table3.2: Decode into Original Message

Codeword (Using r_c)	Codeword (Using r_g)	Character Range	Group Range	Character
0.010850174	0.444334325	$0.0 \leq r_c < 0.1$	$0.4 \leq r_g < 0.5$	N
0.10850174	0.44334325	$0.1 \leq r_c < 0.2$	$0.4 \leq r_g < 0.5$	O
0.0850174	0.4334325	$0.0 \leq r_c < 0.1$	$0.4 \leq r_g < 0.5$	N
0.850174	0.334325	$0.8 \leq r_c < 0.9$	$0.3 \leq r_g < 0.4$	L
0.50174	0.34325	$0.5 \leq r_c < 0.6$	$0.3 \leq r_g < 0.4$	I
0.0174	0.4325	$0.0 \leq r_c < 0.1$	$0.4 \leq r_g < 0.5$	N
0.174	0.325	$0.1 \leq r_c < 0.2$	$0.3 \leq r_g < 0.4$	E
0.74	0.25	$0.7 \leq r_c < 0.8$	$0.2 \leq r_g < 0.3$	A
0.4	0.5	$0.4 \leq r_c < 0.5$	$0.4 \leq r_g < 0.5$	R
0.0				

Finally, we get the encrypted message "NONLINEAR" after assembling the characters.

4. Analysis

In our process, we have taken a number of characters and grouped them with their distinct range of probabilities, which is unique to sender and receiver. The number of characters is not constant for all instances. Any one may take any number of characters depending on their encryption process. Sender and receiver may also change the order of occurrence and range of probabilities of both the characters and groups from process to process.

Here the key length may vary from 1 to 39 alphanumeric characters. Using this method, only ${}^{95}C_1$ to ${}^{95}C_{39}$ number of key combinations is possible (taking input the characters of key only from keyboard). As we have taken the key value (K_v) from the key as 4 digits maximum number. If anybody wants the value of K_v more than 4 digits, he or she may take more number of alphanumeric characters as key and then more number of key combinations will be possible (but *message_lenght ∞1 / number_of_keydigit.*). If we take maximum 100 characters for our encryption process, using single key we can only replace data bits of maximum 30 characters (as using 4 digits maximum numbers only 77 pixels is truly addressable).

If we apply this technique (as in example Section 1) for 1000 number of characters, to address each and every character of a 1000 character set 10 bit (bit depth) is required. For a text with 24 characters 248 (240+8) bits is required. Using our method also 248 (8+3*((2*30) + (1*20))) bits is required. For 10,000 characters, 14 bit (bit depth) is required. For a text with 24 characters 344 (336+8) bits is required. Using our method also 328 (8+4*((2*30) + (1*20))) bits is required. For 1,00,000 characters, 17 bit (bit depth) is required. For a text with 24 characters 416 (408+8) bits is required. Using our method also 408 (8+5*((2*30) + (1*20))) bits is required.

We see that, almost every time data is automatically compressed by this process. Here it's also remarkable that as number of encrypted characters increases total number of bits to replace about image proportionally increases. If anybody wants to encrypt long message using this technique, he or she can use subset of keys (using characters of key) from the key (K) and may assign them to different regions of image. Binary values generated form both textual information and message length are replaced in different nonlinear places in the image. As bits are placed in any one bit of lower four bits of each R, G & B, the change of color of the modified pixels are invisible to human eye. If size of the text is less, the number of pixels affected from the text will also be less. At that time it will be harder to differentiate the encrypted image from the original image. If the image size is large and number of pixels is less, it will also be harder to differentiate [2] [5] [6].

5. Conclusion

We have formed a table (Table 2.1.1.1) with different ranges of probability for each and every character and groups, which is totally unknown to the others i.e. except from sender and receiver. Also the number of characters may vary from process to process Here we have used the variable length key technique. By this technique a large number of key combinations are possible. Again we generated a key value (K_v) from key (K) by arithmetic coding and our proposed algorithm. Using key value (K_v) and nonlinear function technique we have selected both the pixel positions and bit position where the data will be hidden inside the image. Using our process if we encrypt fewer numbers of characters fewer numbers of pixels will be affected by this process. After all, it produces the similar image to see in necked eye at the time of watermarking. If the key becomes unknown to anybody who wants to attack the information, we think, it will be quite impossible to the attacker to find out the information from the watermarked image.

References:

[1] Sabyasachi Samanta, Shyamalendu Kandar, Saurabh Dutta "Implementing Invisible Digital Watermarking on Image" in 'The Bulletin of Engineering and Science' ISSN: 0974 7176 Vol.3 No.2 September, 2008, pp. 79-82.

[2] Sabyasachi Samanta, Saurabh Dutta "Implementation of Invisible Digital Watermarking on Image Nonlinearly of Arithmetically Compressed Data" in 'IJCSNS International Journal of Computer Science and Network Security, Journal ISSN: 1738- 7906 Vol.10 No.4, April 2010 pp.261-266.

[3] Sabyasachi Samanta, Saurabh Dutta "Implementation of Invisible Digital Watermarking on Image Nonlinearly Encrypted with Galois Field (GF- 256)" in "2010 International Conference on Informatics, Cybernetics, and Computer Applications (ICICCA2010)" July 19-21, 2010, pp. 26-30.

[4] Sabyasachi Samanta, Saurabh Dutta "Implementation of Invisible Digital Watermarking on Image using Permutation of Keys and Image Shares" Special issue of IJCCT, ISSN:0975-7449, Vol. 2, Issue 2, 3, 4; 2010, "International Conference [ICCT-2010]", December 3 – 5, pp. 125-129.

[5] Sabyasachi Samanta, Saurabh Dutta "Implementation of Invisible Digital Watermarking on Image Nonlinearly" ,"International Conference on Computer Applications, 2010 (ICCA2010)" December 24-27, pp.189-195 .

[6] Sabyasachi Samanta, Saurabh Dutta Implementation of Invisible Digital Watermarking on Image Nonlinearly of Arithmetically Compressed Data Encrypted by Variable Length Key", "The Second International Conference on Network & Communications Security " January 2- 4, 2011 pp. 523-534.

[7] Sabyasachi Samanta, Saurabh Dutta, "Implementation of Steganography by Embedding Data encrypted with Galois Field (GF-256) into Image using Variable-length Key", "International Conference on Advanced Computing and Communication Technologies- 2011", January 21-23, 2011, pp.263-267

[8] Pranam Paul, Saurabh Dutta, Anup K. Bhattacharjee, "An Approach to ensure Security through Bit-level Encryption with Possible Lossless Compression" IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.2, February 2008, pp. 291-299.

[9] Moni Noar, Adi Shamir "Visual cryptography" Department of Applied Math and Computer Science Wiazmann Institute, Rehovot.

[10] Mahmoud A. Hassan, and Mohammed A. Khalili "Self Watermarking based on Visual Cryptography" World Academy of Science, Engineering and Technology 8,, 2005.

[11] Azzam Sleit, Adel Abusitta "A Visual Cryptography Based Watermark Technology for Individual and Group Images" Systemics, Cybernetics and Informatics Volume 5, Number 2 pp.-24-32.