IMAGE SCRAMBLING USING QUANTUM DOTBITWISE XOR

Dr. Rafiq Ahmad Khan  
Assistant professor  
GDC Bemina, Kashmir  

Mohd Iqbal Sheikh  
PhD Scholar  
Mewar university  
chittorgarh, India

Abstract: With the rapid development of multimedia technology, the image scrambling for information hiding is severe in today’s world. But, in quantum image processing field, the study on image scrambling is still few. Several quantum image scrambling schemes are in circle but, lot of it is yet to be performed. This paper presents the implementation of XOR quantum dot gate using bitwise operation to scramble an image metric. While the XOR operation has only half chance of outputting false or true (0, 1). XOR by scrambling an image so that image can be hidden immensely to avoid third party intervention.

Keywords: Image scrambling, Quantum image processing bit.

INTRODUCTION

Quantum image processing is attracting more and more attention in recent years, from quantum image representation [1–3], quantum image operation [4–7] to quantum image encryption [8–10]. Image scrambling [11,12] is a basic work of image encryption or information hiding [13]. The image after scrambling removes the correlation of image pixels space, which can make the watermark lose the original information, and then, the watermark information is tucked into the carrier. Thus, even if an attacker extracted carriers from the image, he is almost unable to obtain the original image information in any case. Therefore, scrambling processing for the watermark or information hiding is fairly indispensable in a large sense. The scrambling algorithm mainly includes two categories. Image bit-plane refers to a series of two-value image planes. To begin with, the pixel Values in the image are represented by its corresponding binary values, and then, every single bit of all the pixels will form a two-value image, it is called bit-plane. To be specific, if the image gray value range is [0, 255]. Two-input XOR (exclusive OR) also known as exclusive disjunction is a logical function which gives a high Output only if any one of the two inputs but not both are high. The circuit diagram and the layout of XOR gate is shown in Fig 5(a) and Fig 5(b). The third input line of majority gate 1 is made high and that of majority gate 2 is made low. The output of majority gate 2 is fed into an inverter. Finally, the output from the majority gate 1 and that of the inverter is fed into majority gate 3 whose third input line is made 0. The output of majority gate 3 is the XOR function.

WORKING

FLOWCHART

| Input a matrix  |
| Perform XOR of two characters  |
| Perform XOR into bits  |
| Scramble 5X5 MATRIX  |
| CONVERT INTO ARRAY OF NUMBERS  |
| CONCATENATE ALL THE CHARACTERS  |
| OUTPUT MATRIX  |

Bitwise XOR operation to scramble two character matrices by generating a truth table. I need to perform the operation for four characters where each of them have a bit representation as follow:

<table>
<thead>
<tr>
<th>XOR</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

I need to create a table that’s two characters together which gives the values for all combinations of ing pairs of characters in the following way.

XOR   A G C T  
A   A G C T  
G   G A T C  
C   C T A G  
T   T C G A

To obtain the output, you need to convert each character into its bit representation, the bits, then use the result and conver
it back to where we started. As such:

C = 10
C = 10
G = 01
C XOR G = 10 XOR 01 = 11 --> T

I would ultimately like to apply this rule to scrambling characters in a 5 x 5 matrix. As an example:

A = 'GATT' 'AACT' 'ACAC' 'TTGA' 'GGCT'
   'GCAC' 'TCAT' 'GTTC' 'GCCT' 'TTTA'
   'CTAC' 'AAAA' 'GGGC' 'CCCT' 'TCGT'
   'GTGT' 'GCCG' 'GTTG' 'TTGC' 'ATTA'
   'GCTA' 'TTAC' 'GCCA' 'CCCC' 'TTTC'

B = 'ATAC' 'AAAT' 'AGCT' 'AAGC' 'AAGT'
   'TAGT' 'CAGT' 'AGAT' 'GAAG' 'TCGA'
   'GCTA' 'TTAC' 'GCCA' 'CCCC' 'TTTC'
   'CCGT' 'TCGA' 'GACC' 'GCCC' 'TTCA'
   'TAGG' 'AAGT' 'ATGA' 'AAAG' 'AAGA'

I would like to generate a matrix such that each element of A gets XORed with its corresponding element in B. A XOR B.

For example, considering the first row and first column:
A{1,1} XOR B{1,1} = GATT XOR ATAC = GTTG

First, let’s define the function that takes two 4-character strings and both strings correspond to a letter. We will also need the lookup table using a class where given a letter, we produce a two-bit string. Wewant to convert each letter into its two bit representation, and we need the inverse lookup to do this. After, we XOR the bits individually, then use the forward lookup table to get back to where we started. As such:

function [out] = letterXOR(A,B)

codebook = containers.Map([{'00','11','10','01'},{'A','T','G','C'}]); %// Lookup
invCodebook = containers.Map({'A','T','G','C'},{'00','11','10','01'}); %// Inv-lookup
lettersA = arrayfun(@(x) x, A, 'uni', 0); %// Split up each letter into a cell
lettersB = arrayfun(@(x) x, B, 'uni', 0); %// Split up each letter into a cell
valuesA = values(invCodebook, lettersA); %// Obtain the binary bit strings
valuesB = values(invCodebook, lettersB); %// Convert each letter into a matrix
valuesAMatrix = cellfun(@(x) double(x) -48, valuesA, 'uni', 0);
valuesBMATRIX = cellfun(@(x) double(x) -48, valuesB, 'uni', 0);
% XOR the bits now
XORedBits = arrayfun(@(x) bitxor(valuesAMatrix{x}, valuesBMATRIX{x}), 1:numel(A), 'uni', 0);
%// Convert each bit pair into a string
XORedString = cellfun(@(x) char(x + 48), XORedBits, 'uni', 0);
%// Access lookup, then concatenate as a string
out = cellfun(@(x) codebook(x), XORedString);

Let’s go through the above code slowly. The inputs letterXOR are expected to be character array of letters that are composed of A, T, G, C, and N. We first define the forward and reverse lookups. We then split up each character of the input strings A and B into a cell array of individual characters, as looking up multiple keys in your codebook requires it to be this way.

We then figure out what the bits are for each character in each string. These bits are actually strings, and so what we need to do is convert each string of bits into an array of numbers. We simply cast the string to double and subtract by 48, which is the ASCII code for 0. By converting it, you’ll either get 48 or 49, which is why we need to subtract with 48.

As such, each pair of bits is converted into a single value. This value is the column major index of the element in the matrix. We will use that will XOR each four element stringing your cellarray and writing our final matrix. We will use

to xor the bits. The outputs at this point are still 1 x 2 after this, we concatenate all of the characters together to make the final string for the output. Make sure you save the above in a function. Once we have this, we now simply have to use one call that will XOR each four element stringing your cellarray and writing our final matrix. We will use

... Here is digital image of Charles Babbage:

We use XOR, if we used AND, OR or XOR with the one-time and it’s extremely important to understand that AND has a 75% chance of outputting 0 and a 25% chance of outputting 1. While OR has a 25% chance of outputting 0 and 75% chance of outputting 1. While the XOR operation has a 50% chance of outputting 0 or 1 XOR by encrypting an image. Here is an image of Charles Babbage:

Let’s look at a visual example to see the different scrambling effects of AND vs. OR vs. XOR by encrypting an image... Here is digital image of Charles Babbage:
Know it was Babbage? It's equally likely to be picture of you or anything else another thing to note about XOR versus A ND OR is that it is reversible. The truth table for XOR is:

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

So we know whenever we have 0 as the pad bit, we can leave the bit as it is when Decrypting. When we have 1 as the pad bit, we flip the bit to get the decrypted bit.

CONCLUSION

While the XOR operation has only half chance of outputting false or true (0, 1). XOR by scrambling an image so that image can be hidden immensely to avoid third party intervention. We use XOR, if we used AND, OR or XOR with the one-time and it’s extremely important to understand that AND has a 75% chance of outputting 0 and a 25% chance of outputting a 1. While OR has a 25% chance of outputting 0 and 75% chance of outputting 1. While the XOR operation has a 50% chance of outputting 0 or 1. XOR by encrypting an image.

REFERENCES
