Project on

A Visual Cryptographic Scheme for Multiple Secret-Sharing using Monotonic General Access Structures

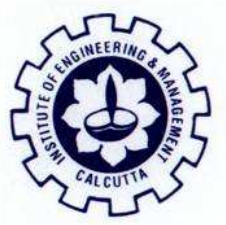
Pritam Bhattacharya

Mrinalkanti Ghosh

Bikash Chandra

Under the able guidance of

Prof. Himadri Nath Saha



Institute of Engineering & Management

To whom it may concern

This is to certify that the project entitled “A Visual Cryptographic Scheme for Multiple Secret-Sharing using Monotonic General Access Structures” was completed and submitted by the following B. Tech. students from the Department of Computer Science & Engineering :

- Pritam Bhattacharya (Roll : 10401061019)
- Mrinalkanti Ghosh (Roll : 10401061025)
- Bikash Chandra (Roll : 10401061013)

The project was carried out in a systematic and procedural manner to the best of our knowledge. It is a bona fide work of the candidates involved and was carried out under the supervision and guidance of Prof. Himadri Nath Saha during the academic year 2009-2010.

Prof. Himadri Nath Saha,
Assistant Professor,
CSE Dept, IEM, Kolkata.

Prof.(Dr.) Debika Bhattacharya,
Head of the Department,
CSE Dept, IEM, Kolkata.

Acknowledgement

The project members would like to acknowledge the invaluable contribution of all those people without whom this project would not have been a success.

First and foremost, we would like to thank our mentor **Prof. (Dr.) Avisek Adhikari**, Assistant Professor, Department of Pure Mathematics, Ballygunge Science College, Calcutta University. Without his continuous supervision and unyielding trust in our abilities, this project would never have seen the light of the day.

We would also like to acknowledge the invaluable suggestions of our guide **Prof. Himadri Nath Saha** regarding the nitty-gritties of project submission and evaluation.

Also, we would like to extend our appreciation to our HOD, **Prof. (Dr.) Debika Bhattacharya,** and the entire faculty of our department.

Finally, we would like to express our heartfelt gratitude to our families for their constant moral support throughout the entire duration of the project.

Project Members

Pritam Bhattacharya (10401061019)

Mrinalkanti Ghosh (10401061025)

Bikash Chandra (10401061013)

[Dept. of Computer Sc. & Engg.]

8th May, 2010

CONTENTS

Abstract……..1
Introduction to Visual Cryptography……2
Share Generation Techniques……4
Extension to Multiple Images……6
System Requirement and Specification……7
Source Code……8
Future Scope……23
Bibliography……24

Abstract

Visual cryptography is a secret-sharing technique where the reconstruction of the secret from the shares distributed among the participants can be done directly by the human visual system, without the need for any explicit computation. In a visual cryptographic scheme involving N participants, a secret image (say S) is used to generate N shadow images, each of them called a share, such that each participant receives a unique share. Then, we define certain subsets of participants, known as qualified sets, who will be allowed to recover the secret. Any other subset of participants, barring the qualified sets, should not be able to recover even partial information about the secret. Ideally, the information content of each individual share should be zero, yet the total information should be recovered whenever the participants belonging to any qualified set (say Q) agree to combine their shares. The visual reconstruction of S is a simple process that consists of photocopying the shares given to the participants belonging to Q onto transparencies, and then stacking them. While defining the qualified sets, we might use simple (K,N)-thresholding, where any K out of the N participants are allowed to reconstruct the secret, or we can define more general access structures, where the cardinality of each qualified set may differ. Furthermore, the access structure might be defined as monotonic, where any superset of any explicitly mentioned qualified set automatically becomes a qualified set, or it might not. This requires only minor variations in the share generation technique using generator matrices.

Introduction to Visual Crytography

Visual cryptography, introduced in 1994 by Moni Naor & Adi Shamir, is a cryptographic technique to encrypt any image in a perfectly secure way such that it can be decoded directly by the human visual system.

To give a proper introduction to the field of visual secret-sharing, a few terms need to be explained first, namely –

- **Secret** : A secret is any piece of information in any medium, that needs to be communicated only to certain intended parties , while ensuring that no other party can lay their hands on it.
- **Participants** : The participants in a secret-sharing scheme refer to the set of all concerned parties within whom the secret has been generated and also needs to be confined.
- **Shares** : Shares are some particular pieces of information derived in some manner from the original secret and distributed among the participants. None of the participants should be able to reconstruct the secret from their share alone, but the shares of certain predefined set of participants, when combined, gives back the original secret to the participants who have agreed to combine their shares. If the set of participants who have combined their shares do not fall within any of the predefined sets, then no information is recovered.
- **Dealer** : Dealer is one of the participants who generates the secret, generates the shares and communicates those shares to the other participants *securely*.
- **Combiner** : Combiner is one of the participants who combines the shares and regenerates the secret. He may be a trusted third party or one of the participants who has been authorized to do so by the participants willing to combine their shares.

- **Access Structure** : The access structure in a secret-sharing scheme is defined to be set of all permissible set of participants who are allowed to regenerate the secret on combining their shares.

  Let the set of participants be $P = \{p_1, p_2, \ldots, p_n\}$. Now, a general access structure G is defined as :

  $G = \{S_1, S_2, \ldots, S_m\}$ where $S_i \in 2^P$ and S_is are permissible sets.

  Eg. – Suppose we have a military organization consisting of 4 generals and 7 corporals who have access to some secret of national importance. Now, keeping the relative importance of the persons concerned, we might define a general access structure where each permissible set may have one of the following compositions :

  - ✓ At least 2 generals
  - ✓ At least 1 general & at least 3 corporals
  - ✓ At least 5 corporals

  Now, the above scheme implies that though any 2 generals by themselves are qualified to reconstruct the secret, even 4 corporals alone will never be allowed to reconstruct the secret from their shares, if at least one general does not agree to co-operate with them.

Share Generation Techniques

In case of secret-sharing schemes for a black and white image, we take black pixel as a 1-bit & white pixel as a 0-bit. Let us assume that the access structure given is :

$$\{ \{P_{11},P_{12},...,P_{1m_1}\} , ... , \{P_{r1},P_{r2},.....,P_{rm_r} \} \}$$

where $\forall i \forall j$, $P_{ij} \in$ Set of participants.

Now we form two systems of linear equations in modulo-2 arithmetic:

$$X_{11}+X_{12}+...+X_{1m_1}=0 \qquad\qquad X_{11}+X_{12}+...+X_{1m_1}=1$$

. . . .

$$X_{r1}+X_{r2}+...+X_{rm_r}=0 \qquad\qquad X_{r1}+X_{r2}+...+X_{rm_r}=1$$

where $\forall i \forall j$ $X_{ij} \in \{0,1\}$ and $\forall i \forall j$ & $\forall a \forall b$ $P_{ij}=P_{ab} \rightarrow X_{ij}=X_{ab}$.

For each system, we represent every solution to it as a column vector and concatenate all these column vectors to form a matrix. Now, we name the matrices formed for the first and the second system of equations as g_0 & g_1 respectively. Now, we form the two sets of matrices G_0 & G_1, where G_i consists of all matrices identical to g_i upto column permutation.

The i^{th} row of any randomly selected generator matrix (G_0 or G_1) indicates the group of pixels to be included in the share of the i^{th} participant for the corresponding message bit (0 or 1) .

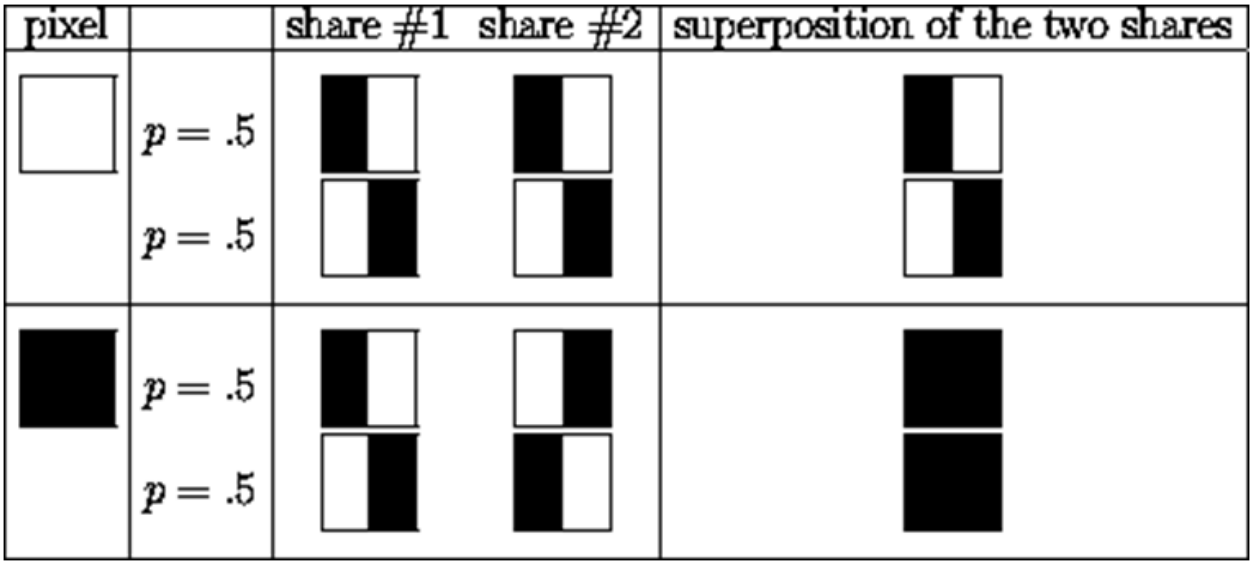
The length of each row in G_0 or G_1, is called the pixel expansion.

Eg. - A very simple scheme where there are 2 participants and both of them have to agree to combine their shares in order to reconstruct the secret leads to the generation of the following sets of matrices :

G_0 = { all column permutation of $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$ }

G_1 = { all column permutation of $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ }

A chart that demonstrates the processes of share generation and secret reconstruction follows :-



The chart above also shows the perfect security of the scheme, since it is impossible to tell by looking at any individual share whether it corresponds to a white pixel or a black pixel, yet it always gives back the correct information when both the shares are combined.

Another interesting point to be noted is that the reconstruction process is perfect for a black pixel, whereas for a white pixel, it never gives back a perfectly white pixel but rather a mixture of a white & a black pixel. This phenomenon leads to a sharp reduction of contrast in the reconstructed image, which simulates the addition of a significant graylevel offset as compared to the original image.

Extension to Multiple Images

The share generation techniques described above can easily be extended to accommodate multiple secrets, with separate access structures for each of them, where it would be enough to distribute a single share to each participant. In this case, we create g_0 & g_1 for each secret and during share generation, for any particular pixel of every image, we choose g_is accordingly and concatenate them to generate G'. Next, we generate a random column permutation of G' and include the i^{th} row of this chosen permutation to the share of the i^{th} participant.

We have seen in the case of share construction with a single secret that a subset of participants who do not form a permissible set can retrieve zero information by combining their shares. Now, consider a situation where a particular set is permissible for only one secret, but is forbidden for every other. If we combine the shares of the members in that particular set, then for other secrets, we can recover zero information (as the set is forbidden for those secrets), whereas the secret for which it is a permissible set can be reconstructed completely. Thus, by stacking the shares, we will get only the intended secret.

System Requirement & Specification

***Programming Language Used** :*

Java (with GUI design in Swing)

***Compiler Used** :*

J2SE v1.6

IDE Used for GUI Design :

NetBeans v6.8

Minimum System Requirement :

As the software is coded in Java it is capable of running on any platform that has adequate hardware support required to install a java run-time environment. However, to ensure smooth operability and reasonable response times, it is recommended that you use a processor with a frequency of at least 1.2 GHz and have at least 256 MB of RAM on your system.

Source Code

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser chooser = new JFileChooser();
   File f;
FileNameExtensionFilter filter = new FileNameExtensionFilter(
   "Any Picture Files", "bmp","jpg","png");
chooser.setFileFilter(filter);
int returnVal = chooser.showOpenDialog(this.getFrame());
if(returnVal == JFileChooser.APPROVE_OPTION) {
  try{
     f= chooser.getSelectedFile();
     img = ImageIO.read(f);
     jLabel8.setText(f.getName());
 }
  catch(Exception e){}
}
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
   n=Integer.parseInt(jTextField2.getText());
   p=Integer.parseInt(jTextField1.getText());
    basis = new int[n][];
   common = new int[n][n];
   for (int i=0;i<n;i++)
   {
      String str= JOptionPane.showInputDialog( this.getComponent(), "Enter elements
      of basis set "+(i+1)+" separated by space :");
      String s[] = str.split(" ");
      int l= s.length;
      basis[count++] = new int[l];
      for (int j = 0; j < s.length; j++) {
            basis[count - 1][j] = Integer.parseInt(s[j]);
   }
   }
```

```
   init_flag(n);
   init_common(n);
   basis_comp();
   pair();
   /*This module solves the system of linear equations and generates the shares*/
create_g();
// Determine the parameters to solve the equations
for (int i = 0; i < n; ++i) {
   if (pairing[i] == 0) {
      // solve a single equation.
      solve_eq1(i);
   }
   else {
      // solve a pair of equations.
      for (int j = i + 1; j < n; ++j) {
         if (pairing[i] == pairing[j]) {
            int x = common[i][j];          // x number of common variables.
            int y = (basis[i].length - x);  // y uncommon variables in the 1st equation.
            int z = (basis[j].length - x);  // z uncommon variables in the 2nd equation.
            solve_eq(x, y, z, i, j);
         }
      }
   }
}
columns0=columns;columns=0;count = 0;  count1 = 0;count2 = -1;  count3 = -1;
System.out.println(G0.length+" "+p+" g "+G0[0].length+" "+columns0);
G00= new int[p][columns0];G01= new int[p][columns0];
for(int i=0;i<p;i++)
   for(int j=0;j<columns0;j++)
   {
      G00[i][j]=G0[i][j];
      G01[i][j]=G1[i][j];
   }
}//GEN-LAST:event_jButton1ActionPerformed

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
out_share();    }//GEN-LAST:event_jButton2ActionPerformed
```

```
   private void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jToggleButton1ActionPerformed
       Decrypt d=new Decrypt(this.getFrame(),true);
       this.getFrame().setVisible(false);
       d.setVisible(true);
       this.getFrame().setVisible(true);
   }//GEN-LAST:event_jToggleButton1ActionPerformed

   private void jButton4ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton4ActionPerformed
JFileChooser chooser = new JFileChooser();
       File f;
   FileNameExtensionFilter filter = new FileNameExtensionFilter(
       "Any Picture Files", "bmp","jpg","png");
   chooser.setFileFilter(filter);
   int returnVal = chooser.showOpenDialog(this.getFrame());
   if(returnVal == JFileChooser.APPROVE_OPTION) {
     try{
         f= chooser.getSelectedFile();
         img1 = ImageIO.read(f);
         jLabel7.setText(f.getName());
      }      catch(Exception e){}    }
   }//GEN-LAST:event_jButton4ActionPerformed

   private void jButton5ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton5ActionPerformed
       n=Integer.parseInt(jTextField4.getText());
       p=Integer.parseInt(jTextField1.getText());
        basis = new int[n][];
       common = new int[n][n];
       for (int i=0;i<n;i++)
       {
          String str= JOptionPane.showInputDialog( this.getComponent(), "Enter elements
          of basis set "+(i+1)+" separated by space :");
          String s[] = str.split(" ");
          int l= s.length;
          basis[count++] = new int[l];
```

```
      for (int j = 0; j < s.length; j++) {
         {
            basis[count - 1][j] = Integer.parseInt(s[j]);
         }
   }
   } // TODO add your handling code here:
   init_flag(n);
   init_common(n);
   basis_comp();
   pair();
   /*This module solves the system of linear equations and generates the shares*/
create_g();
// Determine the parameters to solve the equations
for (int i = 0; i < n; ++i) {
   if (pairing[i] == 0) {
      // solve a single equation.
      solve_eq1(i);
   }
   else {
      // solve a pair of equations.
      for (int j = i + 1; j < n; ++j) {
         if (pairing[i] == pairing[j]) {
            int x = common[i][j];         // x number of common variables.
            int y = (basis[i].length - x);  // y uncommon variables in the first equation.
            int z = (basis[j].length - x);  // z uncommon variables in the second equation.
            solve_eq(x, y, z, i, j);
         }
      }
   }
}
columns1=columns;columns=0;count = 0;  count1 = 0;count2 = -1;  count3 = -1;
G10= new int[p][columns1];G11= new int[p][columns1];
for(int i=0;i<p;i++)
   for(int j=0;j<columns1;j++)
      G10[i][j]=G0[i][j];G11[i][j]=G1[i][j];
}//GEN-LAST:event_jButton5ActionPerformed
```

```
   private void jButton8ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton8ActionPerformed
      showAboutBox();// TODO add your handling code here:
   }//GEN-LAST:event_jButton8ActionPerformed

   private void jButton6ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton6ActionPerformed
       try{
           ImageIcon icon=new ImageIcon(img);
            System.out.println("icon="+icon.getIconHeight());
         JOptionPane.showMessageDialog(this.getComponent(),"","Image1",JOptionPane.
         /INFORMATION_MESSAGE, icon);
           }
       catch(Exception e){
    JOptionPane.showMessageDialog(this.getComponent(), "Unable to display image");
       }
   }//GEN-LAST:event_jButton6ActionPerformed

   private void jButton7ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton7ActionPerformed
      try{
           ImageIcon icon=new ImageIcon(img1);
           System.out.println("icon="+icon.getIconHeight());
         JOptionPane.showMessageDialog(this.getComponent(),null,"Image2",
         JOptionPane.INFORMATION_MESSAGE, icon);
           }
       catch(Exception e)
       {
      JOptionPane.showMessageDialog(this.getComponent(), "Unable to display image");
       }
   }//GEN-LAST:event_jButton7ActionPerformed

   BufferedImage img,img1;
   int[][] basis;
   int p,n;
   private JDialog aboutBox;
   /* Global Variables Used */
```

```
public int temp1,  count = 0,  count1 = 0,  columns = 0,columns0,columns1,
      count2 = -1,  count3 = -1;
/* p: no. of participants
 * n: no. of minimal qualified sets
 * columns: no. of columns in the generating matrices
 */
public int[][]  common;
/* basis[][]: stores the minimal qualified set of participants
 * common[][]: stores the number of common paricipants in every pair
 */
int[][] G0, G1,G00,G01,G10,G11;
/* G0[][], G1[][]: Generating matrices
 */
public String secretKey = "",  output,  input = "";
/* scretKey: stores the secret key string
 * input: stores the binary input stream
 */
public int[] pairing,  flag2;

public void init_flag(int len) {
   pairing = new int[n];
   flag2 = new int[n];
   for (int i = 0; i < len; ++i) {
      pairing[i] = 0;
      flag2[i] = 0;
   }
}

 public void init_common(int len) {
   for (int i = 0; i < len; ++i) {
      for (int j = 0; j < len; ++j) {
         common[i][j] = -1;
      }
   }
}

public int compare(int row1, int row2) {
```

```
    int com = 0;
    for (int i = 0; i < basis[row1].length; ++i) {
        int x = basis[row1][i];
        for (int j = 0; j < basis[row2].length; ++j) {
            if (x == basis[row2][j]) {
                com++;
            }
        }
    }
    return (com);
}

 public void basis_comp() {
    for (int i = 0; i < n; ++i) {
        for (int j = (i + 1); j < n; ++j) {
            common[i][j] = compare(i, j);
        }
    }
}

 public void pair() {
    int i, j;
    if ((n % 2) > 0) {
        j = (n - 1) / 2;
    } else {
        j = n / 2;
    }
    for (i = 0; i < j; ++i) {
        max_common();
    }
}

 public void max_common() {
    int max = 0;
    int row1 = -1, col1 = -1;

    for (int i = 0; i < n; ++i) {
```

```
      for (int j = (i + 1); j < n; ++j) {
         // If either set has already been paired, ignore it
         if (pairing[i] != 0 || pairing[j] != 0) {
            continue;
         }
       // Determine the row and column index of the next greatest element
         if (common[i][j] > max) {
            max = common[i][j];
            row1 = i;
            col1 = j;
         }
      }
   }
   // Pair the basis sets having maximum intersection.
   if (row1 != -1 && col1 != -1) {
      pairing[row1] = pairing[col1] = ++count1;
   }
}
//Method to create and initialize the generating matrices.
public void create_g() {
   int sum = 0, com, x, y;
   for (int i = 0; i < n; ++i) {
      if (pairing[i] == 0) {
         x = (basis[i].length) - 1;
         columns += Math.pow(2, x);
      }
      for (int j = (i + 1); j < n; ++j) {
         if (pairing[j] == 0) {          continue;          }
         if (pairing[i] == pairing[j]) {
            com = compare(i, j);
            x = basis[i].length;
            y = basis[j].length;
            sum = (x + y) - (com + 2);
            columns += Math.pow(2, sum);
         }
      }
   }
```

```
   //To initialize the generating matrices
   G0 = new int[p][columns];
   G1 = new int[p][columns];
   for (int i = 0; i < p; ++i) {
      for (int j = 0; j < columns; ++j) {
         G0[i][j] = 0;
         G1[i][j] = 0;
      }
   }
}

 public int num_XOR(int num) {
   int mask = 1, acc = 0;
   for (int i = 0; i < 8; ++i) {
      int M = mask & num;
      acc = acc ^ M;
      num = num >>> 1;
   }
   return (acc);
}

//Methods to update the elements of the generating matrices.
public void update_G0(int v1, int v2, int v3, int row1, int row2, int c2) {
   char mask = 1, flag = 0;
   for (int i = 0; i < basis[row1].length; ++i) {
      int X = basis[row1][i];
      System.out.println(X);
      flag = 0;
      for (int j = 0; j < basis[row2].length; ++j) {
         if (X == basis[row2][j]) {           flag = 1;              }
      }

      if (flag == 1) {    //it is a common element between row1 and row2
         short M = (short) (mask & v1);
         G0[X - 1][c2] = M;
         v1 = v1 >>> 1;
      } else {
```

```
            short M = (short) (mask & v2);
            G0[X - 1][c2] = M;
            v2 = v2 >>> 1;
        }
    }
    for (int i = 0; i < basis[row2].length; ++i) {
        int X = basis[row2][i];
        flag = 0;
        for (int j = 0; j < basis[row1].length; ++j) {
            if (X == basis[row1][j]) {
                flag = 1;
            }
        }
        if (flag == 0) {    //its an extra variable of the second equation
            short M = (short) (mask & v3);
            G0[X - 1][c2] = M;
            v3 = v3 >>> 1;
        }
    }
}

// Method to update the Generating matrix
public void update_G1(int v1, int v2, int v3, int row1, int row2, int c3) {
    int mask = 1, flag = 0;
    for (int i = 0; i < basis[row1].length; ++i) {
        int X = basis[row1][i];
        flag = 0;
        for (int j = 0; j < basis[row2].length; ++j) {
            if (X == basis[row2][j]) {
                flag = 1;
            }
        }
        if (flag == 1) {    //it is a common element between row1 and row2
            short M = (short) (mask & v1);
            G1[X - 1][c3] = M;
            v1 = v1 >>> 1;
        } else {
```

```
            short M = (short) (mask & v2);
            G1[X - 1][c3] = M;
            v2 = v2 >>> 1;
        }
    }
    for (int i = 0; i < basis[row2].length; ++i) {
        int X = basis[row2][i];
        flag = 0;
        for (int j = 0; j < basis[row1].length; ++j) {
            if (X == basis[row1][j]) {
                flag = 1;
            }
        }
        if (flag == 0) {    //its an extra variable of the second equation
            short M = (short) (mask & v3);
            G1[X - 1][c3] = M;
            v3 = v3 >>> 1;
        }
    }
}

//Method to solve the systems of linear equations.
public void solve_eq(int com_var, int ex_var1, int ex_var2, int row1, int row2) {
    for (int i = 0; i < Math.pow(2, com_var); ++i) {
        for (int j = 0; j < Math.pow(2, ex_var1); ++j) {
            for (int k = 0; k < Math.pow(2, ex_var2); ++k) {
                if (num_XOR(i) == num_XOR(j) && num_XOR(i) == num_XOR(k)) {
        /*Condition for solution G0*/
                    count2++;
                    update_G0(i, j, k, row1, row2, count2);
                }
                if (num_XOR(i) != num_XOR(j) && num_XOR(i) != num_XOR(k)) {
        /*Condition for solution G1*/
                    count3++;
                    update_G1(i, j, k, row1, row2, count3);
                }
            }
```

```
        }
    }
}

// Method to solve single linear equation
public void solve_eq1(int row) {
    System.out.println("solve eq has been called !!  " + row);
    for (int i = 0; i < Math.pow(2, basis[row].length); ++i) {
        if (num_XOR(i) == 0) {
            count2++;
            update1_G0(i, row, count2);
        } else {
            count3++;
            update1_G1(i, row, count3);
        }
    }
}

// Method to update generating matrices when no participants are in common
public void update1_G0(int var1, int row, int c2) {
    int mask = 1;
    for (int i = 0; i < basis[row].length; ++i) {
        short M = (short) (mask & var1);
        int a = basis[row][i];
        G0[a - 1][c2] = M;
        var1 = var1 >>> 1;
    }
}

// Method to update generating matrices when no participants are in common
public void update1_G1(int var1, int row, int c3) {
    int mask = 1;
    for (int i = 0; i < basis[row].length; ++i) {
        short M = (short) (mask & var1);
        G1[basis[row][i] - 1][c3] = M;
        var1 = var1 >>> 1;
    }
```

```
}

int[][] randompix(int[][] G)
{
   Random r=new Random();
   int i,j,k,temp;
   for(j=0;j<G[0].length;j++)
   {
      k= r.nextInt(G[0].length);
      for(i=0;i<p;i++)
      {
         temp=G[i][k];
         G[i][k]=G[i][j];
         G[i][j]=temp;
      }
   }
   return G;
}

//Method to calculate the shares.
public void out_share()
{
   int w,h;
   int w0=img.getWidth();
   int h0=img.getHeight();
   int he0=(int)Math.sqrt(columns0);
   int ve0=columns0/he0;
   int g0[][]=new int[p][columns0];
   int g1[][]=new int[p][columns1];
   int w1=img1.getWidth();
   int h1=img1.getHeight();
   int he1=(int)Math.sqrt(columns1);
   int ve1=columns1/he1;
   if(w0*he0 > w1*he1)
      w=2*w0*he0;
   else
      w=2*w1*he1;
```

```
if(h0*ve0 > h1*ve1)
   h=h0*ve0;
else
   h=h1*ve1;
BufferedImage image[]=new BufferedImage[p];
for(int i=0;i<p;i++)
   image[i]=new BufferedImage(w,h,BufferedImage.TYPE_INT_RGB);
for(int j=0;j< w0; j++)
   for(int k=0; k<h0; k++)
     {
       int c=img.getRGB(j, k);
       int r=0xff & (c >> 16);
       int g=0xff & (c >> 8);
       int b=0xff & (c );
       int pix=((r+g+b)/3)/128;
       if(pix == 0)
          g0=randompix(G01);
       else
          g0=randompix(G00);
       for(int i=0;i<p;i++)
       {
          int col=0;
          for(int l=0;l<he0;l++)
           {
              for(int m=0;m<ve0;m++)
              if(g0[i][col++]==0)
                 image[i].setRGB( 2*(j*he0+l)+1, k*ve0+m ,-1);
              else
                 image[i].setRGB( 2*(j*he0+l)+1, k*ve0+m, -16777216);
           }
       }
     }

for(int j=0;j< w1; j++)
   for(int k=0; k<h1; k++)
     {
       int c=img1.getRGB(j, k);
```

```
            int r=0xff & (c >> 16);
            int g=0xff & (c >> 8);
            int b=0xff & (c );
            int pix=((r+g+b)/3)/128;
            if(pix == 0)
               g1=randompix(G11);
            else
               g1=randompix(G10);
            for(int i=0;i<p;i++)
            {
               int col=0;
               for(int l=0;l<he1;l++)
                {
                   for(int m=0;m<ve1;m++)
                   if(g1[i][col++]==0)
                      image[i].setRGB( 2*(j*he1+l), k*ve1+m ,-1);
                   else
                      image[i].setRGB( 2*(j*he1+l), k*ve1+m, -16777216);
                }
            }
          }
      JFileChooser chooser = new JFileChooser();
      for(int i=0;i<p;i++){
         FileNameExtensionFilter filter = new FileNameExtensionFilter("File for Share
        "+(i+1), "png");
         chooser.setFileFilter(filter);
                  if (chooser.showSaveDialog(this.getFrame()) ==
                  JFileChooser.APPROVE_OPTION && chooser.getSelectedFile() != null) {
               File f = chooser.getSelectedFile();
               if (!f.getPath().endsWith(".png")) {
                  f = new File(f.getPath() + ".png");         }
          try{              ImageIO.write(image[i],"png", f);              }
         catch(IOException e){}
      }
      }
   }
}
```

Future Scope

The project holds considerable future scope in the following areas :-

- Reduction of Pixel Expansion –

  The pixel expansion during share generation can be reduced significantly by migrating towards probabilistic schemes, while taking care to ensure that the chances of error in a pixel of the reconstructed image lies below an acceptable threshold.

- Contrast Enhancement in the Reconstructed Image –

  The reconstructed image can be made to look more like the original image by enhancing its contrast through suitable image processing techniques if the regeneration is carried out computationally instead of stacking transparencies (which is done in traditional visual cryptography) .

- Incorporating Similar Schemes for Polychromatic Images –

  As of now, extending the visual secret-sharing schemes to make it work with polychromatic images presents significant challenges, especially due to the fact that the contrast of the reconstructed image reduces quite drastically as the number of colour components increases.

Bibliography

- Visual Cryptography
  - Adi Shamir & Moni Naor [EUROCRYPT '94]

- Visual Cryptography for General Access Structures
  - Giuseppe Ateniese, Carlo Blundo, Alfredo De Santis, and Douglas R. Stinson [ICALP ’96]

- A New Black and White Visual Cryptographic Scheme for General Access Structures
  - Avishek Adhikari, Tridib Kumar Dutta and Bimal Roy [INDOCRYPT 2004, LNCS 3348, pp. 399–413, 2004. Springer-Verlag Berlin Heidelberg 2004]