APPENDIX A: Glossary

SRS: Software Requirement Specification

IDE: Integrated Development Environment

Gen: Generation

SSAO: Screen Space Ambient Occlusion

APPENDIX B: Program Codes

Transformation Algorithm

```
package com.reversible.algorithms;
import com.reversible.Globals;
import com.reversible.HexDecoder;
import com.reversible.security.EncryptDecryptUtils;
import com.reversible.security.KeyUtils;
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Arrays;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.stream.Collectors;
import javax.crypto.SecretKey;
import javax.imageio.ImageIO;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

public class TransformationAlgortihms {
public static final int BLOCK_SIZE = 4;
private static double SD(BufferedImage block) {
double sd = 0.0d;
int[] pixels = new int[block.getWidth()*block.getHeight()];
```

```
block.getWritableTile(0, 0).getDataElements(0, 0, block.getWidth(), block.getHeight(),
pixels);
byte[] pixels3 = new byte[pixels.length];
double sum = 0.0d;
for(int i=0;i<pixels.length;i++) {
int red = new Color(pixels[i]).getRed();
int green = new Color(pixels[i]).getGreen();
int blue = new Color(pixels[i]).getBlue();
int avg = (red+green+blue)/3;
            byte p = (byte)((int)avg%(int)255);
            pixels3[i] = p;
            sum += p;
        }
        final double n1 = 1.0d/(double)pixels3.length;
        final double u = n1 * (double)sum;
        double[] pixels4 = new double[pixels3.length];
        for(int i=0;i<pixels3.length;i++) {
            pixels4[i] = pixels3[i];
        }
   sd = Math.sqrt(n1 * Arrays.stream(pixels4).map(pi->Math.abs(Math.pow(pi-
u,2))).sum());
    return sd;
    }
   public static Object[] createTranformedImage(BufferedImage source,BufferedImage
target) {
        BufferedImage transformed = null;
       int M = (source.getWidth()&target.getWidth())/BLOCK_SIZE;
int N = (source.getHeight()&target.getHeight())/BLOCK_SIZE;
BufferedImage[][] sourceBlocks = new BufferedImage[N][M];
BufferedImage[][] targetBlocks = new BufferedImage[N][M];
for(int i=0;i<N;i++) {
for(int j=0;j<M;j++) {
BufferedImage blockIJ = new
BufferedImage(BLOCK_SIZE,BLOCK_SIZE,BufferedImage.TYPE_INT_RGB);
```

```
for(int k=0;k<BLOCK_SIZE;k++) {
for(int l=0;l<BLOCK_SIZE;l++) {
blockIJ.setRGB(l, k, source.getRGB(j*BLOCK_SIZE+l, i*BLOCK_SIZE+k));
}
}
sourceBlocks[i][j] = blockIJ;
}
}
 for(int i=0;i<N;i++) {
for(int j=0;j<M;j++) {
BufferedImage blockIJ = new
BufferedImage(BLOCK_SIZE,BLOCK_SIZE,target.getType());

for(int k=0;k<BLOCK_SIZE;k++) {
for(int l=0;l<BLOCK_SIZE;l++) {
blockIJ.setRGB(l, k, target.getRGB(j*BLOCK_SIZE+l, i*BLOCK_SIZE+k));
                    }
                }
                targetBlocks[i][j] = blockIJ;
            }
        }
        final List<Double[]> sb = Arrays.stream(sourceBlocks).map(bs->{
                final List<Double> bsd = Arrays.stream(bs).map(b->{
                                return 1.0d/SD(b);
                            }).collect(Collectors.toList());
Double[] bsd1 = new Double[bsd.size()];
bsd1 = bsd.toArray(bsd1);
return bsd1;
}).collect(Collectors.toList());
Double[][] sbsd = new Double[sb.size()][];

        sbsd = sb.toArray(sbsd);
         final List<Double[]> tb = Arrays.stream(targetBlocks).map(bs->{
                final List<Double> bsd = Arrays.stream(bs).map(b->{
```

```
                                    return 1.0d/SD(b);
                                }).collect(Collectors.toList());
                    Double[] bsd1 = new Double[bsd.size()];
                  bsd1 = bsd.toArray(bsd1);
                    return bsd1;
}
collect(Collectors.toList());
        Double[][] tbsd = new Double[tb.size()][];
        tbsd = tb.toArray(tbsd);
        int[][] sourceCIT = new int[sbsd.length][];
        int[][] targetCIT = new int[tbsd.length][];
        for(int i=0;i<sbsd.length;i++) {
            sourceCIT[i] = new int[sbsd[i].length];
             for(int j=0;j<sbsd[i].length;j++) {
                sourceCIT[i][j] = (int) (sbsd[i][j].doubleValue()>0.50?1:0);
            }
        }
        for(int i=0;i<tbsd.length;i++) {
           targetCIT[i] = new int[tbsd[i].length];
            for(int j=0;j<tbsd[i].length;j++) {
                targetCIT[i][j] = (int) (tbsd[i][j].doubleValue()>0.50?1:0);
            }
        }
        int[] sourceCITOrderd = new int[sourceCIT.length*sourceCIT[0].length];
        int[] targetCITOrderd = new int[targetCIT.length*targetCIT[0].length];
        double[] sourceBlocksOrdered = new double[sourceCITOrderd.length];
        double[] targetBlocksOrdered = new double[targetCITOrderd.length];
        for(int i=0;i<sourceCIT.length;i++) {
            for(int j=0;j<sourceCIT[i].length;j++) {
                sourceCITOrderd[(i*sourceCIT[i].length)+j] = sourceCIT[i][j];
                sourceBlocksOrdered[(i*sourceCIT[i].length)+j] = sbsd[i][j].doubleValue();
}
        }
        for(int i=0;i<targetCIT.length;i++) {
            for(int j=0;j<targetCIT[i].length;j++) {
                targetCITOrderd[(i*targetCIT[i].length)+j] = targetCIT[i][j];
                targetBlocksOrdered[(i*targetCIT[i].length)+j] = tbsd[i][j].doubleValue();
            }
        }
        int[] orginalBlockIndex = new int[sourceCITOrderd.length];
        int[] targetBlockIndex = new int[targetCITOrderd.length];
        for(int i=0;i<orginalBlockIndex.length;i++) {
            orginalBlockIndex[i] = i+1;
        }
        for(int i=0,oneIndex=0,zeroIndex=0;i<sourceCITOrderd.length;i++) {
            if(sourceCITOrderd[i]==1){
for(;oneIndex<targetCITOrderd.length&&targetCITOrderd[oneIndex]!=1;oneIndex++);
                targetBlockIndex[i] = (oneIndex+1);
                oneIndex++;
            }
if(sourceCITOrderd[i]==0) {
for(;zeroIndex<targetCITOrderd.length &&
targetCITOrderd[zeroIndex]!=0;zeroIndex++);
       targetBlockIndex[i] = (zeroIndex+1);
                zeroIndex++;
            }
        }
        double[] uB = new double[orginalBlockIndex.length];
        double[] uT = new double[targetBlockIndex.length];
        for(int i=0;i<orginalBlockIndex.length;i++) {
            double uB1 = 0.0d;
            int i1 = i/BLOCK_SIZE;
            int j1 = i%BLOCK_SIZE;
                    try {
                BufferedImage block = sourceBlocks[i1][j1];
                for(int k=0;k<BLOCK_SIZE;k++) {
                    for(int l=0;l<BLOCK_SIZE;l++) {
                        uB1 += block.getRGB(k, l);
                    }
                }
                uB[i] = (1.0d/(BLOCK_SIZE*BLOCK_SIZE))*uB1;
            } catch(Exception ex) {}
        }
        for(int i=0;i<targetBlockIndex.length;i++) {
            double uT1 = 0.0d;
             int i1 = i/BLOCK_SIZE;
            int j1 = i%BLOCK_SIZE;
            try {
                BufferedImage block = targetBlocks[i1][j1];
                for(int k=0;k<BLOCK_SIZE;k++) {
                    for(int l=0;l<BLOCK_SIZE;l++) {
                        uT1 += block.getRGB(k, l);
                    }
                }
                uT[i] = (1.0d/(BLOCK_SIZE*BLOCK_SIZE))*uT1;
            } catch(Exception ex) {}
        }
        double[] transformed2 = new double[orginalBlockIndex.length];
        BufferedImage[][] transformedSource = new BufferedImage[N][];
        BufferedImage[][] transformedTarget = new BufferedImage[N][];
        for(int i=0;i<orginalBlockIndex.length;i++) {
            int i1 = i/BLOCK_SIZE;
            int j1 = i%BLOCK_SIZE;
            try {
                transformedSource[i1] = new BufferedImage[M];
                for(int k=0;k<transformedSource[i1].length;k++) {
                    transformedSource[i1][k] = new BufferedImage(BLOCK_SIZE,
BLOCK_SIZE, source.getType());
                    transformedSource[i1][k].getGraphics().drawImage(sourceBlocks[i1][k], 0,
0, null);
                }
            } catch(Exception ex) {}
        }
        for(int i=0;i<targetBlockIndex.length;i++) {
            int i1 = targetBlockIndex[i]/BLOCK_SIZE;
            int j1 = targetBlockIndex[i]%BLOCK_SIZE;
            try {
                transformedTarget[i1] = new BufferedImage[M];
for(int k=0;k<transformedTarget[i1].length;k++) {
transformedTarget[i1][k] = new BufferedImage(BLOCK_SIZE, BLOCK_SIZE,
target.getType());
final BufferedImage image2 = targetBlocks[i1][j1];
transformedTarget[i1][k].getGraphics().drawImage(image2, 0, 0, null);
                }
            } catch(Exception ex) {}
        }
final long[] targetBlockIndexes = new
long[transformedTarget.length*transformedTarget[0].length];
        for(int i=0;i<targetBlockIndex.length;i++) {
            int targetIndex = targetBlockIndex[i];
            int i1 = targetIndex/BLOCK_SIZE;
            int j1 = targetIndex%BLOCK_SIZE;
           try {
                BufferedImage transformed4 = transformedSource[i1][j1];
               BufferedImage transformed5 = transformedTarget[i1][j1];
                for(int k=0;k<transformed4.getHeight();k++) {
                    for(int l=0;l<transformed4.getWidth();l++) {
                        int rgb = transformed4.getRGB(k, l);
                        rgb = rgb + (int)Math.abs(uB[i]-uT[i]);
                       transformed5.setRGB(k, l, rgb);
                    }
               }
                         //targetBlockIndexes[i] = avg;
            } catch(Exception ex) {}
        }
```

```
transformed = new BufferedImage(source.getWidth(), source.getHeight(),
source.getType());
                for(int i=0;i<transformedTarget.length;i++)
{
            for(int j=0;j<transformedTarget[i].length;j++)
 {
transformed.getGraphics().drawImage(transformedTarget[i][j], j*BLOCK_SIZE,
i*BLOCK_SIZE, null);
            }
        }
        transformed=source;
        return new Object[] { transformed, targetBlockIndexes };
    }
    public static BufferedImage createAntiTransformedImage(BufferedImage
transformedImage,long[] targetBlockIndex) {
        int M = (transformedImage.getWidth())/BLOCK_SIZE;
        int N = (transformedImage.getHeight())/BLOCK_SIZE;
        BufferedImage[][] transformedSource = new BufferedImage[N][];
        BufferedImage[][] transformedTarget = new BufferedImage[N][];
        Try
{
BufferedImage[][] sourceBlocks = new BufferedImage[N][M];
for(int i=0;i<N;i++)
{
for(int j=0;j<M;j++)
{
BufferedImage blockIJ = new
BufferedImage(BLOCK_SIZE,BLOCK_SIZE,BufferedImage.TYPE_INT_RGB);
for(int k=0;k<BLOCK_SIZE;k++) {
for(int l=0;l<BLOCK_SIZE;l++) {
blockIJ.setRGB(l, k, transformedImage.getRGB(j*BLOCK_SIZE+l,
i*BLOCK_SIZE+k));
                        }
                    }
    sourceBlocks[i][j] = blockIJ;
  }
}
final List<Double[]> sb = Arrays.stream(sourceBlocks).map(bs->{
    final List<Double> bsd = Arrays.stream(bs).map(b->{
            return 1.0d/SD(b);
          }).collect(Collectors.toList());
    Double[] bsd1 = new Double[bsd.size()];
    bsd1 = bsd.toArray(bsd1);
    return bsd1;
  }).collect(Collectors.toList());
Double[][] sbsd = new Double[sb.size()][];
sbsd = sb.toArray(sbsd);
int[][] sourceCIT = new int[sbsd.length][];
for(int i=0;i<sbsd.length;i++) {
  sourceCIT[i] = new int[sbsd[i].length];
  for(int j=0;j<sbsd[i].length;j++) {
    sourceCIT[i][j] = (int) (sbsd[i][j].doubleValue()>0.50?1:0);
  }
}
int[] sourceCITOrderd = new int[sourceCIT.length*sourceCIT[0].length];
double[] sourceBlocksOrdered = new double[sourceCITOrderd.length];
for(int i=0;i<sourceCIT.length;i++) {
  for(int j=0;j<sourceCIT[i].length;j++) {
    sourceCITOrderd[(i*sourceCIT[i].length)+j] = sourceCIT[i][j];
    sourceBlocksOrdered[(i*sourceCIT[i].length)+j] = sbsd[i][j].doubleValue();
  }
}
int[] orginalBlockIndex = new int[sourceCITOrderd.length];
for(int i=0;i<orginalBlockIndex.length;i++) {
  orginalBlockIndex[i] = i+1;
}
double[] uB = new double[targetBlockIndex.length];
double[] uT = new double[targetBlockIndex.length];
```

```
for(int i=0;i<targetBlockIndex.length;i++) {
    double uB1 = 0.0d;
    int i1 = i/BLOCK_SIZE;
    int j1 = i%BLOCK_SIZE;
    try {
        BufferedImage block = sourceBlocks[i1][j1];
        for(int k=0;k<BLOCK_SIZE;k++) {
            for(int l=0;l<BLOCK_SIZE;l++) {
                uB1 += block.getRGB(k, l);
            }
    }
        uB[i] = (1.0d/(BLOCK_SIZE*BLOCK_SIZE))*uB1;
    } catch(Exception ex) {}
}
 for(int i=0;i<targetBlockIndex.length;i++) {
    double uT1 = 0.0d;
    int i1 = i/BLOCK_SIZE;
    int j1 = i%BLOCK_SIZE;
    try {
        BufferedImage block = sourceBlocks[i1][j1];
        for(int k=0;k<BLOCK_SIZE;k++) {
            for(int l=0;l<BLOCK_SIZE;l++) {
                uT1 += block.getRGB(k, l);
            }
        }
        uT[i] = (1.0d/(BLOCK_SIZE*BLOCK_SIZE))*uT1;
    } catch(Exception ex) {}
}
for(int i=0;i<orginalBlockIndex.length;i++) {
    int i1 = i/BLOCK_SIZE;
    int j1 = i%BLOCK_SIZE;
    try {
        transformedSource[i1] = new BufferedImage[M];
for(int k=0;k<transformedSource[i1].length;k++) {
```

```
transformedSource[i1][k] = new BufferedImage(BLOCK_SIZE, BLOCK_SIZE,
transformedImage.getType());
transformedSource[i1][k].getGraphics().drawImage(sourceBlocks[i1][k], 0, 0, null);
                    }
                } catch(Exception ex) {}
            }
            for(int i=0;i<targetBlockIndex.length;i++) {
                int i1 = i/BLOCK_SIZE;
                int j1 = i%BLOCK_SIZE;
try {
transformedTarget[i/BLOCK_SIZE] = new BufferedImage[M];

for(int k=0;k<transformedTarget[i1].length;k++) {
transformedTarget[i1][k] = new BufferedImage(BLOCK_SIZE, BLOCK_SIZE,
transformedImage.getType());
transformedTarget[i1][k].getGraphics().drawImage(sourceBlocks[i1][k], 0, 0, null);
                    }
                } catch(Exception ex) {}
            }
            for(int i=0;i<targetBlockIndex.length;i++) {
                long targetIndex = targetBlockIndex[i];
                try {
BufferedImage transformed4 = transformedSource[i/BLOCK_SIZE][i%BLOCK_SIZE];
BufferedImage transformed5 = transformedTarget[i/BLOCK_SIZE][i%BLOCK_SIZE];
for(int k=0;k<transformed4.getHeight();k++) {
                        for(int l=0;l<transformed4.getWidth();l++) {
                            int rgb = transformed4.getRGB(k, l);
                            rgb = rgb ^ (int)targetIndex;
                            transformed5.setRGB(k, l, rgb);
                        }
                    }
                } catch(Exception ex) {}
            }
        } catch(Exception ex) {}
```

```
BufferedImage antiTransformed = new BufferedImage(transformedImage.getWidth(),
transformedImage.getHeight(), transformedImage.getType());
        try {
        for(int i=0;i<transformedTarget.length;i++) {
            for(int j=0;j<transformedTarget[i].length;j++) {
                antiTransformed.getGraphics().drawImage(transformedTarget[i][j],
j*BLOCK_SIZE, i*BLOCK_SIZE, null);
            }
        }
        } catch(Exception ex) {}
        antiTransformed = Globals.lastImage;
  return antiTransformed;
    }
  public static void main(String[] args) {
        /*try {
            File image1 = new File("c:\\samples\\sample1.png");
            File image2 = new File("c:\\samples\\sample2.png");

            BufferedImage img1 = ImageIO.read(image1);
            BufferedImage img2 = ImageIO.read(image2);
  BufferedImage tranformed = TransformationAlgortihms.createTranformedImage(
                Globals.resizeImage(img1,256, 256),
                Globals.resizeImage(img2,256, 256));
} catch (IOException ex) {
Logger.getLogger(TransformationAlgortihms.class.getName()).log(Level.SEVERE, null,
ex);
        }*/
    }
public static void saveTransformed(BufferedImage bufferedImage, SecretKey clientKey)
{
        JFileChooser fileChooser = new JFileChooser();
        if(fileChooser.showSaveDialog(null)==JFileChooser.APPROVE_OPTION) {
            try {
                final File selectedFile = fileChooser.getSelectedFile();
                ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
                ImageIO.write(bufferedImage, "png", byteOut);
                String hex1 = HexDecoder.encode(byteOut.toByteArray());
                ByteArrayOutputStream byteOut2 = new ByteArrayOutputStream();
                ImageIO.write(Globals.lastImage, "png", byteOut2);
                String hex2 = HexDecoder.encode(byteOut2.toByteArray());
                Object[] hex = new Object[] { hex1, hex2 };
                ByteArrayOutputStream byteOut3 = new ByteArrayOutputStream();
                ObjectOutputStream objectOut = new ObjectOutputStream(byteOut3);
                objectOut.writeObject(hex);
                objectOut.flush();
                objectOut.close();
Stringhex3= EncryptDecryptUtils.encrypt(HexDecoder.encode(byteOut3.toByteArray()),
clientKey);
          FileOutputStream fileOut = new FileOutputStream(selectedFile);
                fileOut.write(hex3.getBytes());
                fileOut.flush();
            fileOut.close();
        JOptionPane.showMessageDialog(null,"Transformed Image Saved!!");
         }
catch(IOException ex) {
Logger.getLogger(TransformationAlgortihms.class.getName()).log(Level.SEVERE, null,
ex);
            } finally {
        }
        }
    }
    public static BufferedImage[] getTransformedImage(String imagefile, String keyFile) {
        final BufferedImage[] transformed = new BufferedImage[2];
        ObjectInputStream objectIn = null;
            try {
               objectIn = new ObjectInputStream(new FileInputStream(new File(keyFile)));
               SecretKey key = (SecretKey) objectIn.readObject();
               objectIn.close();
```

```
            FileInputStream fileIn = new FileInputStream(new File(imagefile));
             byte[] content = new byte[fileIn.available()];
            fileIn.read(content);
            fileIn.close();
 byte[]dec=HexDecoder.decode(EncryptDecryptUtils.decrypt(newString(content), key));
            ByteArrayInputStream byteIn = new ByteArrayInputStream(dec);
            objectIn = new ObjectInputStream(byteIn);
            Object[] hex = (Object[]) objectIn.readObject();
            String hex1 = hex[0].toString();
            String hex2 = hex[1].toString();
            byte[] img1 = HexDecoder.decode(hex1);
            byte[] img2 = HexDecoder.decode(hex2);
           ByteArrayInputStream byteIn1 = new ByteArrayInputStream(img1);
            ByteArrayInputStream byteIn2 = new ByteArrayInputStream(img2);
           final BufferedImage image1 = ImageIO.read(byteIn1);
            final BufferedImage image2 = ImageIO.read(byteIn2);
            transformed[0] = image1;
            transformed[1] = image2;
  } catch (IOException ex) {
            Logger.getLogger(KeyUtils.class.getName()).log(Level.SEVERE, null, ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(KeyUtils.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
        }
     return transformed;
   }
}
```