جامعة تشرين

كلية الهندسة المعلوماتية

قسم البرمجيات ونظم المعلومات

السنة الخامسة

تشفير البيانات المخفية

(مشروع تخرج)

اعداد الطلاب

علي رضوان براءة ناصيف محمد الديك

اشراف

الدكتور بسيم برهوم

2019/2020

مقدمة:

بعد دخول التكنولوجيا إلى كافة نواحي الحياة العملية ،حيث يتم توليد المزيد من البيانات في المجالات الطبية والتجارية والعسكرية، والتي قد تتضمن بعض المعلومات الحساسة فكان لابد من وجود ما يحمي هذه البيانات مثل النصوص والصور والأصوات و...، لذلك أصبح الأمان والخصوصية مهمين. ويعد التشفير أحد أهم الوسائل المستخدمة لتوفير بيئة آمنة لتبادل المعلومات .

يحيط بنا علم التشفير من كل جانب، بدايةً من ماكينات الصراف الآلي والهواتف المحمولة والإنترنت، ومرورًا بنظم الأمن التي تحمي الخصوصية في العمل، وانتهاءً بالشيفرات المدنية والعسكرية، فعملية التشفير هي عملية تحويل المعلومات بحيث تصبح غير مقروءة لأحد باستثناء من يملك مفتاح خاص لإعادة تحويل النص المشفر إلى نص مقروء، عملية التحويل هذه تتم عن طريق مفتاح التشفير، نتيجة عملية التشفير تصبح المعلومات مشفرة وغير متاحة لأي أحد.ويمكن الاستفادة من عملية التشفير أيضا في عمليات تراسل البيانات الهامة في مختلف الأغراض، بالتالي عملية التشفير وارتباطها بالأمان أصبحت من الضرورات الأساسية التي تستخدم في الوقت الحالي والمستقبل.

يقدم هذا المشروع دراسة لخوارزمية الإخفاء ضمن البتات *LSB (Least Significant Bit)* ، ودراسة لخوارزمية *Frame* وهي للإخفاء على شكل إطار للصورة، مع الاستفادة من خوارزميتي الإخفاء في إخفاء نصوص مدخلة من قبل المستخدم ضمن بتات الصور أو على شكل إطار، ودراسة عن خوارزمية التشفير المتناظر *RC6* وشرح عن عمليات التشفير وفك التشفير فيها مع الاستفادة من هذه الخوارزمية لتشفير الصور بأحجام مختلفة ، ودراسة لخوارزمية *RSA* والاستفادة منها في تشفير مفتاح خوارزمية التشفير *RC6* . ثم إرسالها الى طرف آخر حيث يقوم بفك تشفير الصورة ثم إظهار الملف المخفي فيها. كما قمنا بتصميم برنامج يحقق عمليات التشفير وفك التشفير بلغة java من خلال واجهة بسيطة من خلال بيئة(Net Beans) .كما سنقوم بتقديم تطبيقاً لأجهزة الموبايل العاملة بنظامAndroid يقوم بتأدية مهمات الإخفاء والتشفير وفك التشفير لنص او اي ملف يريده المستخدم .اعتمدنا في هذه التطبيق على بيئة اندرويد استديو (Android Studio) وهي عبارة عن منصة تتيح للمطورين كتابة الشيفرة المصدرية للتطبيقات مع إمكانية معاينة الهيئة التطبيقية لها على مختلف قياسات الشاشات بشكل فوري. كما استخدمنا لغة java في بناء هذا التطبيق.

الفهرس

فهرس الأشكال

جدول المصطلحات:

المعنى	المصطلح
Least Significant Bit	*LSB*
Rivest Cipher	*RC6*
Rivest-Shamir-Adleman	*RSA*

الفصل الأول

خوارزمية إخفاء البيانات ضمن البكسلات

1.1 - إخفاء البيانات:

ميزة إخفاء البيانات أنها لا تثير الشك بوجود بيانات مخفية بكونها تحتوي على رسالة أو ملف مخفي، وهي تختلف عن التشفير، فناتج التشفير يكون مختلفا تماما وغير مقروء أو مفهوم، مما قد يثير الشكوك حوله . للتبسيط يمكن تشبيه التشفير بخزنة تضع بداخلها البيانات، ولا يمكن فتح الخزنة إلا بمفتاح معين، بينما إخفاء البيانات عبارة عن تمويه فقط للبيانات، ككتابة رسالة معينة، لو جمعت أول حرف من كل كلمة فيها تجد رسالة أخرى مثلا.

هناك عدد من الطرق والخوارزميات لإخفاء البيانات، حيث بإمكانك اخفاء نوع من البيانات (كالنصوص والملفات) داخل نوع آخر من البيانات، فمثلا بإمكانك اخفاء صورة بداخل صورة أخرى، أو صورة بداخل ملف فيديو، أو نص بداخل ملف صوتي، أو نص بداخل صورة وهكذا، دون أن يلاحظ من يشاهد الصورة بتغيير فيها، فالخوارزميات المختصة بإخفاء البيانات تتلاعب بمحتويات الملف، حيث تعدل الـ *bits* الخاصة بالملف وتتلاعب به بحيث لا تؤثر على محتويات الملف، وبنفس الوقت تحقن بداخله الـ *bits* الخاصة بالملف الآخر، فعندما ترى صورة تحتوي بداخلها على نص مثلا، فإنك لن ترى النص، فالنص يُمثل داخل الصورة على شكل*bits* ، ولن تلاحظ كذلك تغيرا على جودة الصورة بحسب الطريقة المستخدمة طبعا .

2.1 - مم تتكون الصور:

تتكون الصور من مجموعة كبيرة من النقاط يسمى الواحد منها *Pixel* ، وكل بكسل يحتوي لون واحد فقط، بالتالي فإننا عندما نتكلم عن طريقة "تمثيل الصور" نتكلم فعليا عن طريقة تمثيل كل بكسل داخل الصورة، لذلك بإمكانك تشبيه الصورة بـ *Matrix* مصفوفة ثنائية البعد ويحتوي كل عنصر في هذه

المصفوفة على بكسل واحد فقط، تختلف طريقة تمثيل البيكسل بحسب عمق الألوان ووجود شفافية في الصورة من عدمه، لكن بشكل عام، نستطيع أن نقول أن اللون يتكون من تداخل قيم ثلاثة ألوان، وهي الأحمر والأخضر والأزرق (*RGB*) ، وقيمة كل لون من هذه الألوان تُمثل باستخدام رقم من 0 إلى 255 وهو الرقم الذي يحدد كثافة اللون، ومما سبق معرفته عن طريقة تمثيل أرقام *Decimal* باستخدام *Binary* ، نعلم أن الرقم 255 هو *(11111111)* في نظام *Binary* ، أي أننا سنكون بحاجة لبايت واحد فقط لتمثيل اللون، وعلى افتراض ان البيكسل فيه 3 قنوات فقط دون قناة الشفافية الموجودة في بعض صيغ الصور كـ *PNG* مثلا، فإننا سنمثل البيكسل باستخدام *3 Bytes* حيث يُمثل كل لون في بايت واحد.

في تطبيقنا نستخدم خوارزميتي إخفاء وهما خوارزمية *LSB* وخوارزمية *Frame* ، ال *LSB* هي خوارزمية إخفاء بالبيكسلات ، *Frame* خوارزمية إخفاء بالإطار، وفيما يلي شرح عن كل خوارزمية على حدى.

3.1 – خوارزمية الإخفاء في البت الأقل تأثيراً *Least Significant Bit (LSB) :*

1.3.1– ما هو البت الأقل تأثيراً :

لو قمنا بقراءة الـ *Bits* من اليمين إلى اليسار، فالقيمة العددية التي يمثلها الـ Bit تزداد، فالـ *Bit* الأول من اليمين يمثل لنا الرقم 1 ، بينما يمثل الأخير الرقم 128، لذلك فإننا كلما تحركنا إلى اليمين تقل القيمة العددية التي يمثلها الـ *bit* ويقل تأثيره على البيانات ، فمثلا لو كان لدينا صورة وعكسنا قيمة الـ *bit* الأول من اليمين لأحد البكسلات (أو حتى لجميعها) ففعليا لن تلاحظ فرقا كبيرا، فلو كانت كثافة اللون الأحمر في ذلك البيكسل 127، فستصبح القيمة 126 بعد تغيير الـ *bit* الأول من 1 إلى 0 ، والعين البشرية لن تلاحظ التغيير، فهو تغيير طفيف جدا، لذلك يسمى الـ *bit* الأقل تأثيرا، فكلما تحركنا إلى اليمين قلت أهمية الـ*bit* . [1]

ليكن هذا مثال على أحد الألوان وتمثيله باستخدام *Binary :*

شكل *1-1*: تمثيل اللون باستخدام *Binary* في الصورة المجاورة. [1]

الصورة التالية ستعرض نفس اللون بعد تغيير أول Bit من كل لون، إضافة إلى عرض اللون القديم لتقارن بينهما:

شكل*1-2* : عرض مقارنة اللون القديم واللون الجديد بعد تغيير البت الأول (الأقل تأثيراً) من كل لون . [1]

نلاحظ أنه من غير الممكن أبدا التمييز بين اللونين بالعين المجردة .

2.3.1 – آلية عمل الخوارزمية:

بما أن تغيير قيمة الـ Bit الأقل تأثيرا لا تُعد مشكلة بالنسبة للملف (سواء كانت صورة أو مقطع صوت أو فيديو أو غيرها) فإن بإمكاننا استخدام هذه الـ Bits في إخفاء بيانات أخرى.

على سبيل المثال، هذا *الـ Binary* الخاص بهذا النص *"code"* :

01100011 01101111 01100100 01100101

حيث أخذنا تمثيل الحروف ب *ASCII* ثم أخذنا التمثيل الست عشري لكل حرف و حولناه إلى النظام الثنائي. [1]

نفترض أن لدينا صورة نريد إخفاء هذا النص بداخلها ستكون خطوات الإخفاء كالتالي:

- سنقوم بقراءة كل Pixel في الصورة، بعدها نجد أبعاد المصفوفة وتساعدنا أبعاد المصفوفة على إعادة الصورة إلى شكلها الأصلي بعد خزن النص بداخلها [2]
- نستخرج الـ Binary الخاص بالألوان فيه حتى نستطيع ان نغير البت الثامن الايمن الأقل تأثيراً LSB .
- ندخل النص المرد إخفائه ونحوله الى ASCII Code ومنه الى النظام الثنائي ونجد عدد الصفوف وعدد الأعمدة له حتى نستطيع ان ننقل بايتات كل حرف الى داخل البتLSB في الصورة حيث كل بايت من بايتات الحرف سيخزن في البت الأقل تأثيراً لموقع معين في الصورة [2]
- نُغير قيمة الـ Bit الأقل تأثيرا من كل لون إلى قيمة أحد الـ Bits الخاصة بالنص السابق المدخل ، فالـ Pixel الأول من اليسار مثلا سيحتوي أول 3 Bits من النص السابق حيث يحتوي كل لون على Bit واحد من الـ Bits الخاصة بالنص .[1]
- نحول الصورة من النظام الثنائي الى النظام العشري ونعيد ابعاد المصفوفة الى أبعادها الاولية.

ستوضح هذه الصورة آلية عمل الخوارزمية:

شكل *3-1*: طريقة إخفاء بتات النص *code* ضمن بيكسلات صورة في موقع البت الأقل تأثيراً. [1]

❖ كود الخوارزمية *LSB* في مشروعنا:

```
public static boolean encode(String path, String message) throws
IOException
                {
                  BufferedImage   image  = getImage(path);
                  //user space is not necessary for Encrypting
                  image = add_text(image,message);
                  String path1=path.substring(0,path.length()-
               4)+"1"+path.substring(path.length()-4, path.length());
                  return(setImage(image,new File(path1),"png"));
               }
               public static BufferedImage getImage(String f)

              {  {

  BufferedImage image= null;

  File file = new File(f);
```

```
   try
   {
     image = ImageIO.read(file);
   }
   catch(Exception ex)
   {
     JOptionPane.showMessageDialog(null,  " النص المدخل كبير جرب الاخفاء عن طريق
الاطار!"،"خطأ",JOptionPane.ERROR_MESSAGE);
   }
   return image;
  }
public static boolean setImage(BufferedImage image, File file, String ext)
{
   try
   {
     file.delete(); //delete resources used by the File
     ImageIO.write(image,ext,file);
     return true;
   }
   catch(Exception e)
   {
     JOptionPane.showMessageDialog(null,
       "خطأ"،"لانستطيع حفظ الملف!",JOptionPane.ERROR_MESSAGE);
     return false;
   }
  }
```

```
    ////////////////----------------------/////////
  private static BufferedImage add_text(BufferedImage image, String text)
  {
    byte img[]  = get_byte_data(image);
    byte msg[] = text.getBytes();
    byte len[]   = bit_conversion(msg.length);
    try
    {
      encode_text(img, len,  0); //0 first positiong
      encode_text(img, msg, 32); //4 bytes of space for length: 4bytes*8bit = 32 bits
    }
    catch(Exception e)
    {
      JOptionPane.showMessageDialog(null,
"النص كبير حاول اخفاؤو بخوارزمية الإطار!"، "خطأ",JOptionPane.ERROR_MESSAGE);
    }
    return image;
  }
```

3.3.1 – إعادة النص المخفي من الصورة:

- نحول الصورة التي اخفينا النص في داخله الى صيغة النظام الثنائي.
- نأخذ البت الثامن من كل صف ونخزنه في مصفوفة جديدة ونلاحظ انه يجب ان يرسل لنا صاحب الرسالة المشفرة عدد صفوف وعدد اعمدة النص المشفر ومن اين يبدأ التشفير داخل الصورة حتى نستطيع إخراجه . في هذه الحالة يجب ان نقول للمستلم الصورة توضيح عن

النص المرسل وهو نقول له عدد الاحرف وعدد Bit لكل حرف حتى نستطيع ارجاع النص من داخل الصورة. [2]

- نعيد تحجيم المصفوفة النص الى شكلها الأولي بصيغة النظام الثنائي ونحولها من النظام الثنائي الى النظام العشري .
- نحول البيانات من العشري الى الحروف . [2]

❖ كود استخراج النص المخفي من صورة بواسطة LSB:

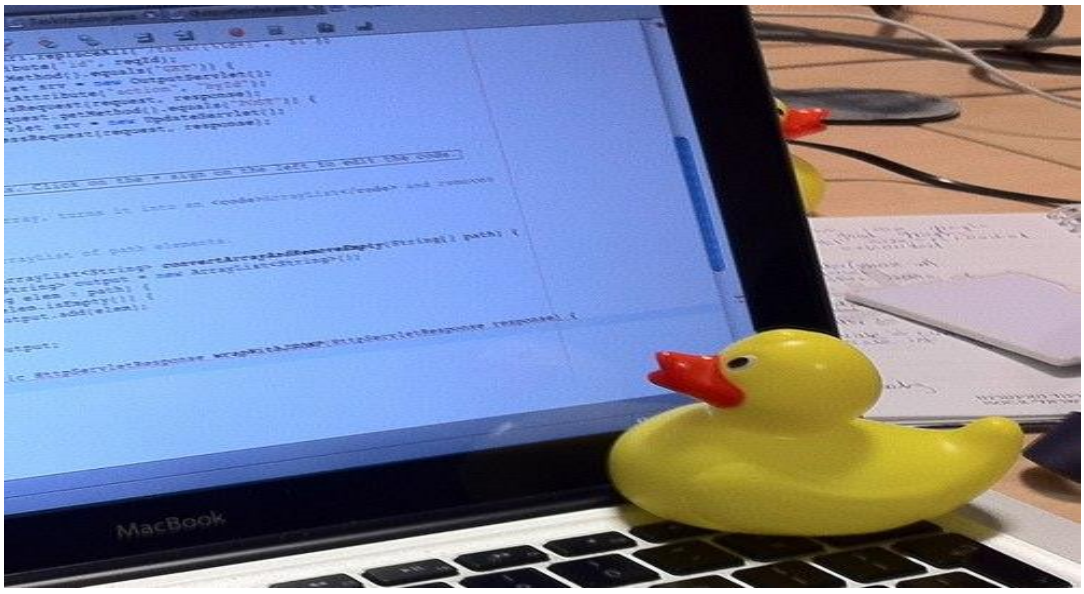
```
public static String decode(String path)
 {
  byte[] decode;
  try
  {
   //user space is necessary for decrypting
            BufferedImage  image  = Steganography.getImage(path);
   //BufferedImage image  = user_space(getImage(path));
   decode = decode_text(get_byte_data(image));
   return(new String(decode));
  }
    catch(Exception e)
    {
   JOptionPane.showMessageDialog(null,
    "لايوجد رسالة لاخفاؤها!"،"خطأ",
    JOptionPane.ERROR_MESSAGE);
   return "";
    }
public static byte[] get_byte_data(BufferedImage image)
 {
  WritableRaster raster   = image.getRaster();
  DataBufferByte buffer = (DataBufferByte)raster.getDataBuffer();
  return buffer.getData();
 }
    ////////////-/-;;;;;/////
 public static byte[] bit_conversion(int i)
 {
```

```
  //only using 4 bytes
  byte byte3 = (byte)((i & 0xFF000000) >>> 24); //0
  byte byte2 = (byte)((i & 0x00FF0000) >>> 16); //0
  byte byte1 = (byte)((i & 0x0000FF00) >>> 8 ); //0
  byte byte0 = (byte)((i & 0x000000FF)       );
  return(new byte[]{byte3,byte2,byte1,byte0});
 }

     ////////////--------///
 private static byte[] encode_text(byte[] image, byte[] addition, int offset)
 {
  //check that the data + offset will fit in the image
  if(addition.length + offset > image.length)
  {
   throw new IllegalArgumentException("الملف صغير!");
  }
  //loop through each addition byte
  for(int i=0; i<addition.length; ++i)
  {
   int add = addition[i];
   for(int bit=7; bit>=0; --bit, ++offset) //ensure the new offset value
carries on through both loops
   {
    int b = (add >>> bit) & 1;
    image[offset] = (byte)((image[offset] & 0xFE) | b );
   }
  }
  return image;
 }
private static byte[] decode_text(byte[] image)
 {
  int length = 0;
  int offset  = 32;
  //loop through 32 bytes of data to determine text length
  for(int i=0; i<32; ++i) //i=24 will also work, as only the 4th byte
contains real data
  {
   length = (length << 1) | (image[i] & 1);
  }
```

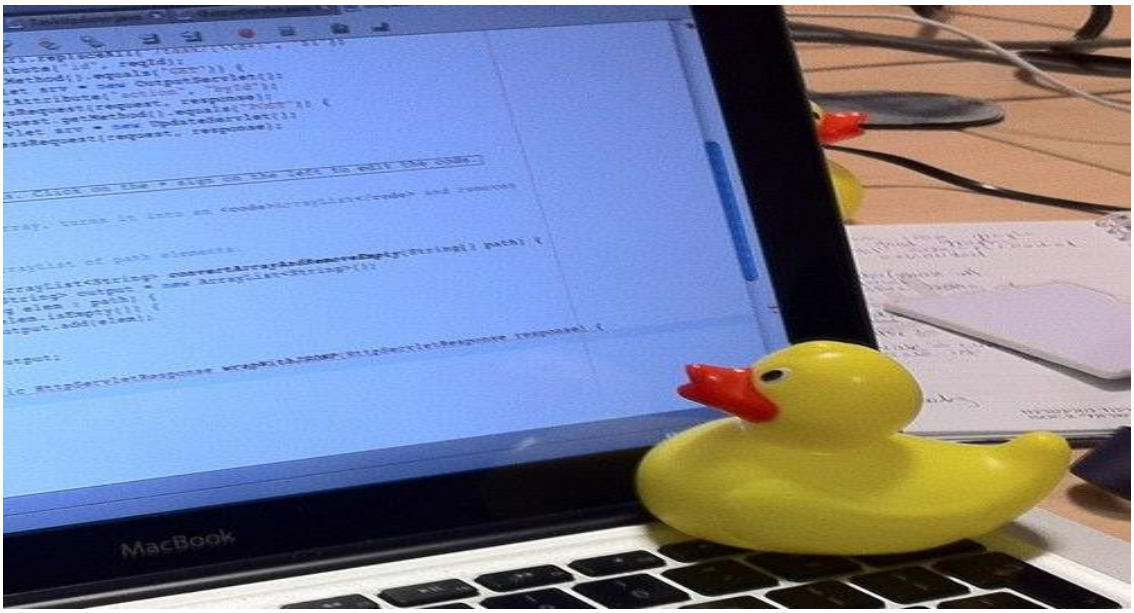
```
 byte[] result = new byte[length];

 //loop through each byte of text
 for(int b=0; b<result.length; ++b )
 {
  //loop through each bit within a byte of text
  for(int i=0; i<8; ++i, ++offset)
  {
   //assign bit: [(new byte value) << 1] OR [(text byte) AND 1]
   result[b] = (byte)((result[b] << 1) | (image[offset] & 1));
  }
 }
 return result;
}
```

4.1 – تطبيق عملي :

الصورة التالية توضح عملية الاخفاء باستخدام خوارزمية *LSB*



شكل1-4:الصورة الاصلية قبل اخفاء البيانات في البكسلات



شكل 1-5: الصورة بعد اخفاء البيانات في البكسلات

الفصل الثاني

خوارزمية الإخفاء ضمن الإطار *Frame*

تعمل خوارزمية الاطار على إخفاء البيانات على شكل إطار للصورة ، حيث لن يكتشف المستخدم وجود الإطار إلا في حال وجود الصورة الأصلية لديه.

1.2 – آلية عمل الخوارزمية:

بعد تحويل النص المطلوب إخفاؤه الى ثنائي يتم إخفاء البيانات في البايتات على عكس خوارزمية *LSB* حيث هنا يتم إخفاء البيانات في بتات الصورة.

تعمل خوارزمية الاطار على إخفاء البيانات على شكل إطار للصورة الأصلية حيث بعد إدخال النص المطلوب إخفاؤه وتحديد الصورة يتم حساب حجم النص ويحجز له أربع بايتات في البداية (يخزن حجم النص في أول أربع بايتات) ثم تبدأ عملية الاخفاء في السطر الأول ثم السطر الأخير ثم العمود الأول ماعدا اول واخر عنصر لأنه تم تعبئتهم في السطرين الاول والأخير ثم العمود الأخير أيضا بدون اول واخر عنصر لان تم تعبئتهم في السطرين الأول والأخير ثم ننتقل للسطر الثاني وهكذا حتى ننتهي من ادخال النص حيث سنواجه عقبة هنا حيث اذا تم ادخال النص ولم يتم تعبئة الاتجاهات الأربعة فيجب ان نقوم بالحشو حتى لا يتم تشويه الصورة وإثارة الريب حيث سيدل ذلك على وجود بيانات مخفية ان لم يكتمل الاطار وان كان النص كبير سيزداد عرض خط الاطار ليتم الاخفاء بشكل كامل دون نقص بالبيانات المدخلة.

مثال :

شكل 2-1: مثال عن خوارزمية الFRAME

❖ كود إخفاء نص باستخدام خوارزمية Frame :

```
public static boolean encode(String path, String message) throws
IOException
 {
   BufferedImage   image  = Steganography.getImage(path);
   //user space is not necessary for Encrypting
   image = add_text(image,message);
   String path1=path.substring(0,path.length()-
4)+"1"+path.substring(path.length()-4, path.length());
   return(Steganography.setImage(image,new File(path1),"png"));
 }
private static BufferedImage add_text(BufferedImage image, String text)
```

```
  {   // System.out.println(image.getWidth());
            //System.out.println(image.getHeight());
    //convert all items to byte arrays: image, message, message length
    byte img[]  = Steganography.get_byte_data(image);
    byte msg[] = text.getBytes();
    byte len[]   = Steganography.bit_conversion(msg.length);
//    try
//    {
      encode_text(img,image, len,  0); //0 first positiong
      encode_text(img,image, msg, 4); //4 bytes of space for length:
4bytes*8bit = 32 bits
//    }
//    catch(Exception e)
//    {System.err.println(e);
//      JOptionPane.showMessageDialog(null,
//"Target File cannot hold message!",
"Error",JOptionPane.ERROR_MESSAGE);
//    }
    return image;
  }
private static byte[] encode_text(byte[] image,BufferedImage image1, byte[]
addition, int offset)
  {//System.out.println(image.length);
    //check that the data + offset will fit in the image
    if(addition.length + offset > image.length)
    {
      throw new IllegalArgumentException("الملف صغير!");
    }
    //loop through each addition byte
            int m=image1.getHeight();
           // System.out.println("sss"+m);
            int n=image1.getWidth()*3;
          //              System.out.println("sss"+n);
        // System.out.println(image.length);
            byte[][]im=new byte[m][n];
            int k=0;
            for(int i=0;i<m;i++){
            for(int j=0;j<n;j++)
            {im[i][j]=image[k];
            k++;}
            }
            int i=0;
             int r=0;
            int c=offset;
            int len=addition.length;
```

```
            while(r<len){
               ////////السطر الاول
            for(int j=c;j<n-i;j++){

           if(r>=len)im[i][j]=(byte)122;
           else{int add=addition[r++];
           im[i][j]=(byte)add;}
            }
             //السطر الاخير
            for(int j=i;j<n-i;j++){

             if(r>=len)im[m-1-i][j]=(byte)122;
             else{  int add=addition[r++];
               im[m-1-i][j]=(byte)add;}
            }
  //العمود الاول
            for(int j=i+1;j<m-i-1;j++){

            if(r>=len)im[j][i]=(byte)122;
            else{int add=addition[r++];
               im[j][i]=(byte)add;}
            }

            //العمودالاخير
             for(int j=i+1;j<m-i-1;j++){

       if(r>=len)im[j][n-1-i]=(byte)122;
       else{ int add=addition[r++];
               im[j][n-1-i]=(byte)add;}
            }

            i++;
            c=i;
            }////while


k=0;
for(int x=0;x<m;x++){
            for(int j=0;j<n;j++)
            {image[k]=im[x][j];
            k++;}
            }

   return image;
  }
```

2.2 – استخراج البيانات المخفية:

عند استخراج البيانات المخفية عن طريق خوارزمية الاطار هنا سنقوم بالعملية العكسية حيث نستخرج طول النص من اول اربع بايتات ثم نقوم بأخذ البايتات من الصورة بنفس ترتيب الادخال حتى حجم النص الذي تم استخراجه من الأربع بايتات الأولى .

❖ كود استخراج نص من صورة بواسطة Frame :

```
public static String decode(String path)
 {
   byte[] decode;
   try
   {
    //user space is necessary for decrypting
                BufferedImage   image  = Steganography.getImage(path);
    decode = decode_text(Steganography.get_byte_data(image),image);
    return(new String(decode));
   }
     catch(Exception e)
     {
    JOptionPane.showMessageDialog(null,
      "لايوجد رسالة مخفية!"،"خطأ",
      JOptionPane.ERROR_MESSAGE);
    return "";
     }
private static byte[] decode_text(byte[] image,BufferedImage image1)
{
   int len = 0;
   int offset  = 4;
   //loop through 32 bytes of data to determine text length
   for(int i=0; i<4; ++i) //i=24 will also work, as only the 4th byte contains
real data
   {
    len = (len << 8) | (image[i] &0xff);
   }

   byte[] result = new byte[len];

          int m=image1.getHeight();
```

```
        int n=image1.getWidth()*3;
        byte[][]im=new byte[m][n];
        int k=0;
        for(int i=0;i<m;i++){
        for(int j=0;j<n;j++)
        {im[i][j]=image[k];
        k++;}
        }
        int i=0;
         int r=0;

        int c=offset;
        while(r<len){
           ////////السطر الاول
        for(int j=c;j<n-i;j++,r++){
       if(r<len)
       result[r]=im[i][j];
        }
         //السطر الاخير
        for(int j=i;j<n-i;j++,r++){
      if(r<len)
          result[r]=im[m-1-i][j];
        }
//العمود الاول
        for(int j=i+1;j<m-i-1;j++,r++){
       if(r<len)
          result[r]=im[j][i];
        }

        //العمودالاخير
         for(int j=i+1;j<m-i-1;j++,r++){
            if(r<len)
      result[r]=im[j][n-1-i];
         }

        i++;
        c=i;
        }////while


return result;  }
```

3.2 – تطبيق عملي:

الصورة التالية توضح عملية الاخفاء باستخدام خوارزمية *Frame* حيث يتم تشكيل اطار حول الصورة الاصلية .

شكل 2-2 : الصورة الأصلية.

شكل 2-3 :الصورة بعد إخفاء نص ضمن الإطار.

الفصل الثالث

خوارزميات التشفير

1.3 –خوارزميات التشفير المتناظرة وغير المتناظرة:

يستخدم في عمليات التشفير العديد من الخوارزميات حيث تصنف خوارزميات التشفير حسب نوع مفتاح التشفير وفك التشفير إلى نوعين: المتناظرة وغير المتناظرة.

1.1.3- خوارزميات التشفير بالمفتاح المتناظر *Symmetric Key* :

نستخدم المفتاح نفسه في التشفير وفك التشفير، ومن مزايا التشفير المتماثل انه سهل الاستعمال وسريع حيث أن المستخدمين لا يعانون من تأخير طويل ويمنح التشفير بالمفتاح العام درجة مقبولة من إثبات هوية طرفي الاتصال حيث أنه لا يمكن فك تشفير المعلومات باستخدام مفتاح آخر غير الذي استخدم في التشفير،ولكن يفقد التشفير المتناظر فعاليته في حال حصول طرف ثالث على المفتاح وذلك قلما يحدث إلا في الشبكات الكبيرة نتيجة لتبادل المفاتيح.

من الأمثلة على الخوارزميات التي تستخدم المفتاح المتناظر *DES, ASE, RC6, 3DES* .

2.1.3– خوارزميات التشفير بالمفتاح غير المتناظر *Asymmetric Key* :

يتم توليد مفاتيح مختلفة ثم استخدامها في تشفير وفك تشفير زوجين من مفاتيح الحماية وباستخدام هذين الزوجين من المفاتيح، أحدهما عام public والآخر خاص private ، يستطيع مفتاح واحد منهما فقط أن يقوم بفك الشفرة التي ينشئها الآخر. حيث يتم توزيع المفاتيح يعطى لطرف مفاتيح التشفير فقط ويعطى للطرف الآخر مفاتيح فك التشفير، ومن غير المرجح أن تؤدي معرفة مفتاح واحد فقط إلى تحديد المفتاح الآخر، ولهذا يتم استخدام نظام التعمية غير المتماثل في إنشاء التوقيعات الرقمية ونقل المفاتيح المتماثلة . من الامثلة خوارزمية *RSA* .

2.3 - خوارزميات التشفير الكتلي والمتسلسل:

1.2.3 - التشفير التسلسلي *(Stream Cipher):*

هو ذلك النوع من التشفير الذي يقوم بتعمية سيالة من المعطيات الرقمية بشكل تسلسلي بحيث يتم تشفير *bit* واحد أو *byte* واحد في كل لحظة زمنية مثل خوارزمية *RC4, Vigenere* .

2.2.3 - التشفير الكتلي *(Block Cipher):*

تتم معالجة كتلة من النص الأصلي كوحدة متكاملة بحيث تنتج كتلة مساوية لها في الطول، يكون طول الكتلة القياسية المستخدمة 64 بت أو 128 بت .من الأمثلة *DES, RC6* .

الفصل الرابع

خوارزمية التشفير *RC6 (Rivest Cipher)*

خوارزمية*RC6* هي خوارزمية تشفير متناظرة تعتمد مبدأ التشفير بالكتل *(Block Cipher algorithm)* بطول كتلة 128 بت (16 بايت) ومفتاح يمكن أن يأخذ قيم 16 أو 24 أو 32 بايت.

1.4 – ميزات الخوارزمية:

تتميز بسرعتها وتحتاج ذاكرة صغيرة، ومن الميزات الإضافية لهذه الخوارزمية مرونتها حيث تسمح باستخدام قيم متعددة للمفاتيح وعدد الدورات وطول بلوك التشفير وهذا ما جعلها إحدى أهم وأفضل الخوارزميات وهي عبارة عن تطوير لخوارزمية RC عبر سلسلة زمنية بدأت من RC2 حتى وصلنا الى RC6

Algorithm	RC2	RC 4	RC 5	RC 6
year	1987	1987	1994	1998
cipher	block	stream	block	block
Block size	64	2064	32, 64, 128	128, 256
Key size	8-128 default 64	1-256	0-2048	128, 192, 256
Rounds	16	256	0-255	20 (recommended)

شكل *4-1: تطور الخوارزميات من RC2 حتى RC6 ومواصفات كل خوارزمية*

نلاحظ أن جميع الخوارزميات اعتمدت على التشفير الكتلي ماعدا خوارزمية واحدة اعتمدت على *stream cipher* أو التشفير التدفقي وهي خوارزمية *RC4* .

Pseudo code for RC6 encryption: [3]

Input:

- *Plain text stored in four w-bit input registers A, B, C ,D*
- *Number r of rounds*
- *w-bit round keys S[0, … ,2r + 3]*

Output:

- *Cipher text stored in A, B, C, D*

Procedure:

$B = B + S[0]$

$D = D + S[1]$

for i = 1 to r do

{

$t = (B * (2B + 1)) <<< lg\ w$

$u = (D * (2D + 1)) <<< lg\ w$

$A = ((A \oplus t) <<< u) + S[2i]$

$C = ((C \oplus u) <<< t) + S[2i+ 1]$

$(A, B, C, D) = (B, C, D, A)$

}

$A = A + S[2r + 2]$

$C = C + S[2r + 3]$

2.4 - دخل الخوارزمية:

يكون دخل الخوارزمية كتلة عبارة عن اربع كلمات *A,B,C,D* كل كلمة بطول *4Byte* فيكون دخلها عبارة عن كتلة 16*byte* .

في حال كان طول الدخل أكبر من 128 بت يقتطع الدخل ويأخذ أول 128 بت والبتات المتبقية ندخلها كتلة ثانية ونضيف حشو للكتلة الثانية.

في حال كان طول الدخل أقل من 128 بت نضيف للكتلة حشو بحيث نصل لطول الدخل المطلوب.

ولدينا الدخل مفتاح ذو طول *16byte* يتم إنشاء 44 مفتاحا جزئيا منه، وذلك في خوارزمية توسيع المفاتيح، وكل مفتاح جزئي بطول *4byte* فيكون الحجم النهائي لكتلة المفتاح الموسع هو 176 بايت.

3.4 - خرج الخوارزمية:

يكون الخرج عن كتلة مشفرة بحجم 16byte *(128 bits)*، كل *4 byte* تكون كلمة خرج لأحد المسارات الأربعة للخوارزمية.

4.4 - عمليات الخوارزمية: [3]

تعتمد هذه الخوارزمية على العمليات التالية:

- التدوير لليسار a<<<b :

rotate the w-bit word a to the left by the amount given by the least significant lgw bits of b.[3]

private static int rotl (int val, int pas) {

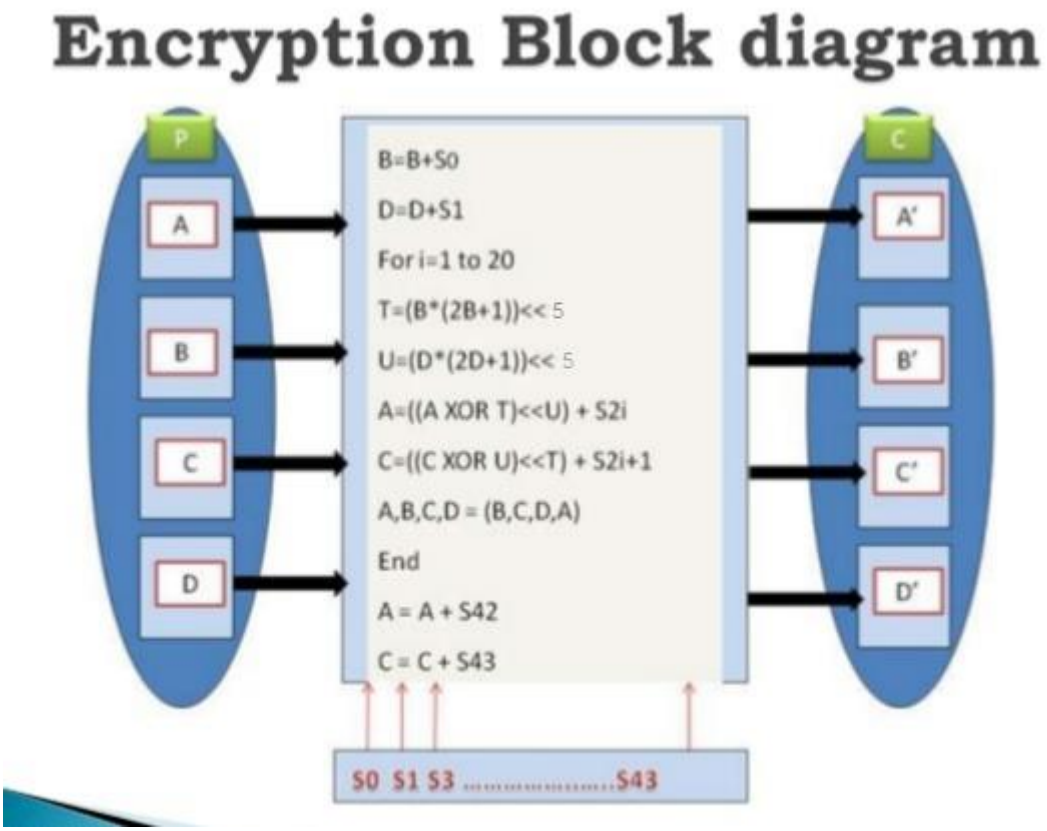
return (val << pas) | (val >>> (32 - pas));}

- التدوير لليمين a>>>b :

rotate the w-bit word a to the right by the amount given by the least significant lgw bits of b. [3]

```
Private static int rotr(int val , int pas){
return (val>>>pas)|(val<<(32-pas)) ;}
```

- عملية XOR .
- عملية الجمع *Integer addition modulo 2w*
- عملية الطرح *Integer subtraction modulo 2w*
- عملية الضرب *Integer multiplication modulo 2wn*



شكل 4-2 :مخطط التشفير لبلوك واحد .

5.4- توسيع المفتاح *Expanding Key* :

قبل أن نتحدث عن توسيع المفتاح يجب ان نعلم ان خوارزمية *RC6* محددة ب *W/R/B* حيث:

W: هو حجم الكلمة، يمكن أن يكون *(16، 32، 64) bytes*

وحجم الكلمة المستخدم في هذا المشروع هو *16 bytes* .

R: هو عدد دورات الخوارزمية.

عدد الدورات القياسي لخوارزمية RC6 هو 20 دورة وهو المستخدم في تحقيق المشروع.

B: طول مفتاح التشفير بالبايت .يمكن أن يكون *(16، 32، 64) byte* .

يكون الدخل عبارة عن المفتاح ذو الطول 16 بايت، يتم تحويله إلى شكل سداسي عشر، والخرج هو مصفوفة تحوي 44 مفتاحا جزئيا طول كل منها 4 بايت.

يستخدم ثابتان في توسيع المفتاح:

Pw=0xB7E15163 وهي تساوي *e* ^-2

Qw=0x9e3779b9 تساوي النسبة الذهبية

حيث يتم اسناد *Pw* الى اول دليل من المفتاح الموسع *S [0]* ثم يسند ناتج جمع *Qw* مع العنصر السابق الى كل عنصر *S[1..43]* .

Pseudo code for expanding key :

S [0] = Pw,

For i= 1 to 2r+3 do

S[i] = S [i – 1] + Qw

A = B = k = j = 0

*v =3 * max(c, 2r+4)*

For s = 1 to v do {

A = S[k] = (S[k] + A + B) <<< 3

B =L[j] = (L[j] +A + B) <<< (A + B)

k = (k + 1) mod (2r + 4)

j = (j + 1) mod c}

حيث *c =key.length /u* علما ان c هو عدد الكتل المشكلة بعد تقسيم المفتاح إلى مداخل الخوارزمية الأربعة وهي 4 كتل.

u هو عدد البايتات في الكلمة.

*key.length*هو طول المفتاح المدخل.

استعنا بأربع متحولات مساعدة لبناء المفاتيح وهي *(A,B,K,j)* حيث تكون *k* دليلا لمصفوفة المفاتيح الجزئية *S* . *j* دليل للمفتاح المدخل من قبل المستخدم .

نجد جمع المفتاح الموسع مع *A* و *B* مع تدوير يساري بمقدار 3 ، ثم اسناد الناتج لكل من المفتاح الموسع و *A* .

❖ تحقيق خوارزمية التوسيع في المشروع:

```
public static int[] generateSubkeys(byte[] key) {
int u = w / 8;
int c = key.length / u;
int t = 2 * r + 4;
int[] L = convBytesWords(key, u, c);
int[] S = new int[t];
for (int i = 0 ; i<S.length;i++){
```

```
S[i]=0;
}
S[0] = Pw;
for (int i = 1; i < t; i++)
S[i] = S[i - 1] + Qw;
int A = 0;
int B = 0;
int k = 0, j = 0;
int v = 3 * Math.max(c, t);
for (int i = 0; i < v; i++) {
A = S[k] = rotl((S[k] + A + B), 3);
B = L[j] = rotl(L[j] + A + B, A + B);
k = (k + 1) % t;
j = (j + 1) % c;
}
return S;
}
```

المتحولات في الكود:

u=w/8 تشير الى عدد البايتات المخصص لكل كلمة من المفتاح.

C=key.length /u تشير الى عدد الكلمات في المفتاح وهنا استخدمنا 4 كلمات (16/4)

t=2r+4 عدد المفاتيح الجزئية التي ستنتج بعد توسيع المفتاح.

عرفنا مصفوفة *L* اسندنا اليها التابع *convertByteWords* سنخزن فيها المفتاح بعد تحويله إلى سداسي عشر وتخزين كل 4 بايت من المفتاح في خانة من المصفوفة *L* .

المصفوفة *S* بحجم *t* سوف نخزن فيها المفاتيح الجزئية.

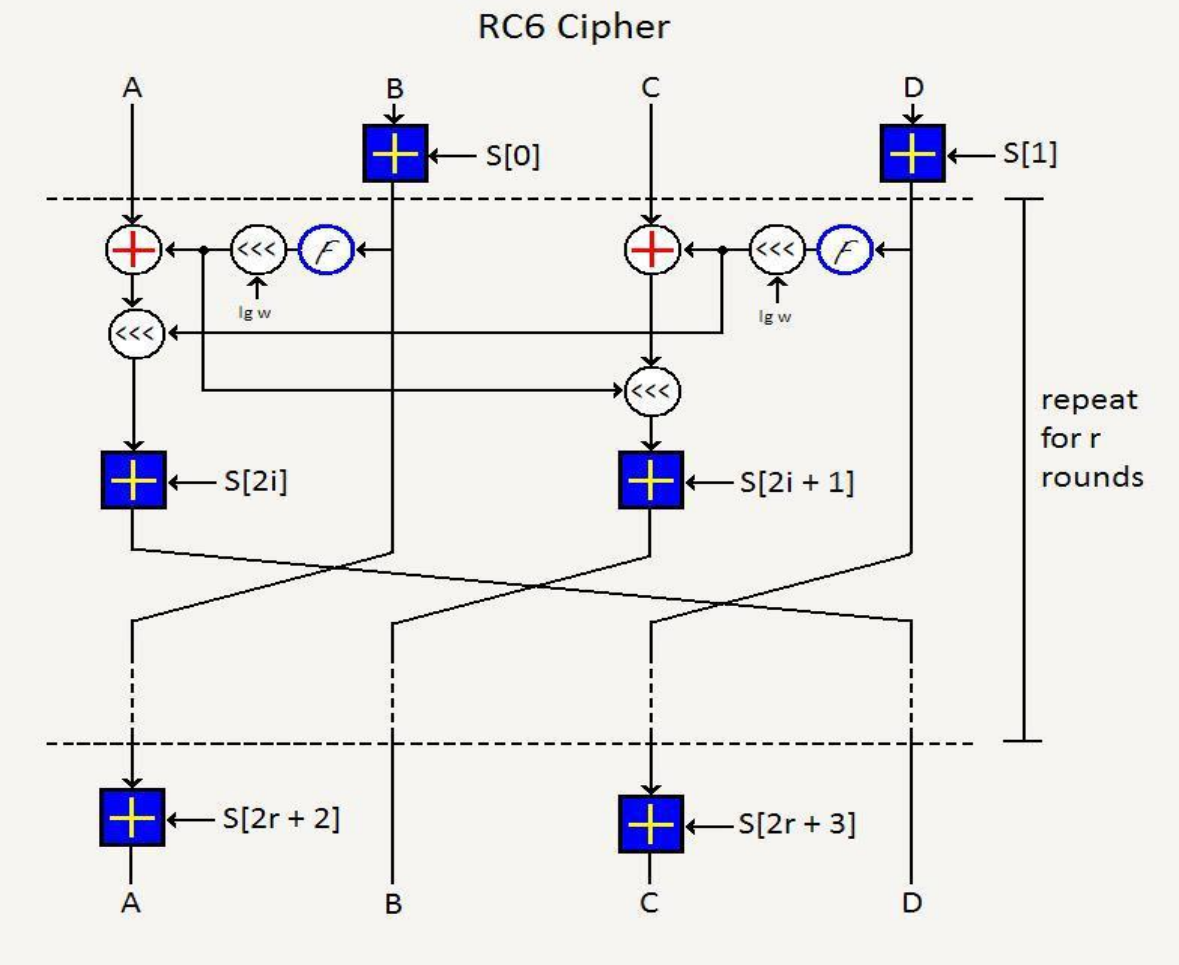
- التابع *convertByteWords*

```
private static int[] convBytesWords(byte[] key, int u, int c) {
```

```
int[] tmp = new int[c];
for (int i = 0; i < tmp.length; i++)
    tmp[i] = 0;
for (int i = 0, off = 0; i < c; i++)
   tmp[i] = ((key[off++] & 0xFF)) | ((key[off++] & 0xFF) << 8)
     ((key[off++] & 0xFF) << 16) | ((key[off++] & 0xFF)<< 24);|
   return tmp;
}
```

6.4 - خطوات خوارزمية التشفير :

بعد توسيع المفتاح ندخل الكتلة المراد تشفيرها على شكل اربع كلمات *A,B,C,D* مع استخدام المفتاح الموسع في التشفير.



شكل *3-4* : مخطط التشفير للخوارزمية من أجل *20* دورة . *Here f(x) = x * (2x + 1).*

حسب مخطط *RC6* السابق فإن عملية التشفير في هذه الخوارزمية تمر بعدة مراحل يمكن أن تلخص بالخطوات التالية: [4]

1.6.4 - ادخال بلوك البيانات حجم *128* بت.

- تقسيم البلوك الى اربعة اجزاء وهي *A,B,C,D* .كل جزء يكون حجمه *32* بت .

2.6.4 - ادخال مفتاح التشفير ويكون حجمه *128* بت أيضاً ، ويبدأ دورته من *S[0,..,2r+3]* من خلال الخطوات التالية :

1.2.6.4 - نأخذ الجزء الاول B ونجري عليها اول خطوات التشفير، حيث نأخذ الجزء B وندمجه مع المفتاح *S[0]* وحسب المعادلة الآتية:

$$B=B+S[0]$$

2.2.6.4 - ان النتيجة التي تم الحصول عليها من الخطوة السابقة تدخل الى دالة f ، وهذه الدالة تجري عمليات معقدة على تلك النتيجة حيث تقوم بإضافة تصريح معين الى المفتاح او الكتلة او الاثنين معاً.

3.2.6.4 - نأخذ النتيجة من الخطوة السابقة ثم نجري عليها عملية الازاحة 5 مراتب،

$W=32$ بالتالي $log2\ (32) = 5$ ونضع الناتج بمتحول t

حسب المعادلة الأتية:

$$t = (B(2B+1)) <<< lgw$$

ملاحظة : نحول نتيجة هذه الخطوة *(3.2.6.4)* الى الجزء C وكما موضح في الشكل *(3-4)*

4.2.6.4 - بعد ذلك يتم اخذ الكتلة A ودمجها مع نتيجة الخطوة السابقة t بواسطة العملية *XOR* ثم يزاح الناتج بمقدار الناتج الذي تم الحصول عليه من الجزء D من خلال نفس العملية التي حصلت للجزء B في الخطوات السابقة (*4-6-2-1, 4-6-2-2*) .

5.2.6.4 - نأخذ نتيجة الخطوة (*4.2.6.4*) ونجمعها مع المفتاح ذو الدليل الزوجي *S[2i]*

حسب المعادلة الآتية:

$$A= ((A^{\wedge} t) <<< u) + S[2i]$$

6.2.6.4 - ان نتيجة الخطوات السابقة ادت الى تحول الجزء A الى D حسب الشكل *(4-3)* .

ملاحظة : كما لاحظنا في العملية التي جرت في الخطوات السابقة قد تمت بين الجزئين A وB فقط.

3.6.4 - نجري نفس العمليات السابقة ولكن هذه المرة بين الجزئين C و D ،حيث نأخذ D ونجمعه مع المفتاح *S[1]* ونجري العمليات السابقة نفسها التي تمت على الكلمة B، من حيث الإزاحة و الدالة f ولكن مع اختلاف المعادلات ، وكما نلاحظ المعادلات التالية:

$$u = ((D\ (2D+1)) <<< lgw$$

$$C = ((C \wedge u) <<< t) + S[2i+1]$$

ملاحظة : ان هذه الخطوات تتكرر في كل دورة الى ان ينتهي عدد الدورات التي تمر بها الخوارزمية وهي 20 دورة ، يتم تنفيذ العديد من العمليات في كل دورة يتم استخدام كلمتين من المفتاح الموسع هما *S[2i]* و *S[2i+1]* التالي خلال 20 دورة يتم استخدام 40 كلمة من المفتاح.

4.6.4 - ان النتائج النهائية التي نحصل عليها من هذه الخوارزمية هي كالتالي:

D ⇦ A , A ⇦ B, B ⇦ C , C ⇦ D

في نهاية دورات الخوارزمية ، الكلمة B نضيف إليها المفتاح $S[2r+2]$ اي المفتاح $S[42]$ قبل الأخير من المفاتيح الموسعة ، كذلك الأمر بالنسبة للكلمة D حيث يضاف لها المفتاح الأخير $S[43]$.

❖ كود الخوارزمية في تطبيقنا:

```
public static byte[] encryptBloc(byte[] input) {
        byte[] tmp = new byte[input.length];
        int t, u;
       int aux;
      int[] data = new int[input.length / 4];
      for (int i = 0; i < data.length; i++)
            data[i] = 0;
     int off = 0;
     for (int i = 0; i < data.length; i++) {
          data[i] = ((input[off++] & 0xff)) |
                    ((input[off++] & 0xff) << 8) |
                   ((input[off++] & 0xff) << 16) |
                   ((input[off++] & 0xff) << 24);
      }
     int A = data[0], B = data[1], C = data[2], D = data[3];
     B = B + S[0];
     D = D + S[1];
     for (int i = 1; i <= r; i++) {
          t = rotl(B * (2 * B + 1), 5);
    u = rotl(D * (2 * D + 1), 5);
    A = rotl(A ^ t, u) + S[2 * i];
   C = rotl(C ^ u, t) + S[2 * i + 1];
      aux = A;
      A = B;
      B = C;
      C = D;
     D = aux;
  }
  A = A + S[2 * r + 2];
 C = C + S[2 * r + 3];
 data[0] = A;
 data[1] = B;
 data[2] = C;
 data[3] = D;
for (int i = 0; i < tmp.length; i++) {
tmp[i] = (byte) ((data[i / 4] >>> (i % 4) * 8) & 0xff);
```

```
}
return tmp;
}
```

هذا الكود كان من اجل تشفير بلوك واحد أي دخل 128 بت 16 بايت وهو ما عملنا عليه في تطبيقنا سيتم الاستفادة من هذا الكود واستدعاؤه لمعالجة الدخل مهما كان طوله وذلك عبر تشفير كل بلوك على حدى بشكل متتابع

- يتم تنجيز ذلك في الكود التالي:

```
public static byte[] encrypt(byte[] data, byte[] key) {
        byte[] bloc = new byte[16];
       S = generateSubkeys(key);
       int lenght = 16 - data.length % 16;
       byte[] padding = new byte[lenght];
       padding[0] = (byte) 0x80;
      for (int i = 1; i < lenght; i++)
         padding[i] = 0;
     int count = 0;
    byte[] tmp = new byte[data.length + lenght];
    int i;
     for (i = 0; i < data.length + lenght; i++) {
          if (i > 0 && i % 16 == 0) {
               bloc = encryptBloc(bloc);
               System.arraycopy(bloc, 0, tmp, i - 16, bloc.length);
            }
          if (i < data.length)
               bloc[i % 16] = data[i];
          else {
               bloc[i % 16] = padding[count];
               count++;
               if (count > lenght - 1) count = 1;
            }
           }
           bloc = encryptBloc(bloc);
           System.arraycopy(bloc, 0, tmp, i - 16, bloc.length);
           return tmp; }
```

هنا قمنا بعمليات تقسيم الكتلة المدخلة الى بلوكات كل بلوك هو 16 بايت وقمنا بتشفير بلوك تلو الاخر وتخزينه في مصفوفة كما قمنا ب اجراء عمليات حشو اذا كان طول البلوك اقل من طول 128 بت(16 بايت)يتم الحشو حتى يكتمل البلوك، القيم التي قمنا بالحشو بها هي اصفار.

7.4 – فك التشفير :

حتى نقوم بعملية فك الشفرة فاننا نتبع نفس الخطوات السابقة ولكن بصورة معاكسة اي عكس عملية التشفير، حيث تم طرح آخر كلمتين من المفتاح أو آخر مفتاحين جزئيين من كل من *A* و*c*

حيث تم طرح المفتاح الأخير من *C* والمفتاح ما قبل الأخير من*A*

ثم دخلنا بحلقة *for* تنازلية كل تكرار فيها يتم تنفيذ عمليات الإبدال بين الكتل والعمليات هي:

1.7.4 - ضرب *D* ب 2 ثم جمعها مع 1 وضرب الناتج ب*D* نفسها ثم إزاحة الناتج النهائي بمقدار *5*

2.7.4 - تكرار نفس العملية مع *B ونضع الناتج في t* .

3.7.4 - طرح المفتاح الجزئي ذو الدليل الفردي من *C* وإزاحة الناتج بمقدار *t* نحو اليمين ثم إجراء *XOR* مع *u*

4.7.4 - نفس العملية تتكرر مع A

بعد الانتهاء من حلقة *for* يتم طرح المفتاح الثاني من *D* ثم طرح المفتاح الأول من *B* .

حسب *pseudo code* الآتي:

Decryption with RC6-w/r/b [3]

Input:

- *Cipher text stored in four w-bit input registers A;B;C;D*
- *Number r of rounds*
- *w-bit round keys S[0 ,… , 2r + 3]*

Output:

- *Plaintext stored in A,B,C,D*

Procedure:

C = C - S[2r + 3]

A = A - S[2r + 2]

for i = r downto 1 do

{

(A, B,C,D) = (D, A,B,C)

u = (D ×(2D + 1))<<<lg w

t = (B ×(2B + 1))<<<lg w

C = ((C - S[2i + 1])>>>t) ⊕ _ u

A = ((A - S[2i])>>>u) ⊕ t

}

D = D - S[1]

B = B - S[0]

❖ كود الخوارزمية في تطبيقنا:

```
public static byte[] decryptBloc(byte[] input){
        byte[] tmp = new byte[input.length];
        int t,u;
       int aux;
      int[] data = new int[input.length/4];
      for(int i =0;i<data.length;i++)
            data[i] = 0;
     int off = 0;
    for(int i=0;i<data.length;i++){
          data[i] = ((input[off++]&0xff))|
                ((input[off++]&0xff) << 8) |
               ((input[off++]&0xff) << 16) |
             ((input[off++]&0xff) << 24);
```

```
                }
            int A = data[0],B = data[1],C = data[2],D = data[3];
            C = C-S[2*r+3];
           A = A-S[2*r+2];
          for(int i = r;i>=1;i--){
                  aux = D;
                  D = C;
                 C = B;
                B = A;
               A = aux;
               u = rotl(D*(2*D+1),5);
               t = rotl(B*(2*B + 1),5);
              C = rotr(C-S[2*i + 1],t) ^ u;
               A = rotr(A-S[2*i],u) ^ t;
            }
            D = D-S[1];
           B = B-S[0];
          data[0] = A;data[1] = B;data[2] = C;data[3] = D;
            for(int i = 0;i<tmp.length;i++){
                        tmp[i] = (byte)((data[i/4] >>> (i%4)*8) & 0xff);
                }
           return tmp;
}
```

هذا التابع من أجل فك تشفير كتلة واحدة كما تحدثنا سابقا

❖ سنستفيد من هذا التابع من أجل استدعائه في فك التشفير من أجل كل البلوكات:

```
public static byte[] decrypt(byte[] data, byte[] key) {
    byte[] tmp = new byte[data.length];
   byte[] bloc = new byte[16];
    key = paddingKey(key);
    S=generateSubkeys(key);
   int i;
  for(i=0;i<data.length;i++){
           if(i>0 && i%16 == 0){
             bloc = decryptBloc(bloc);
              System.arraycopy(bloc, 0, tmp, i-16, bloc.length);
     }
    if (i < data.length)
          bloc[i % 16] = data[i];
   }
  bloc =decryptBloc(bloc);
 System.arraycopy(bloc, 0, tmp, i - 16, bloc.length);
 tmp = deletePadding(tmp);
```

```
return tmp;
}
```

هذا التابع من أجل فك التشفير للبلوكات واحداً تلو الآخر حتى يتم فك تشفير الدخل الناتج عن التشفير كاملا نلاحظ هنا أننا استخدمنا تابع لحذف الحشو الذي تمت إضافته في عملية التشفير وهو *delete padding*

❖ تابع ال: Delete padding

```
public static byte[] deletePadding(byte[] input){
    int count = 0;
    int i = input.length - 1;
    while (input[i] == 0) {
        count++;
        i--;
    }
    byte[] tmp = new byte[input.length - count - 1];
    System.arraycopy(input, 0, tmp, 0, tmp.length);
    return tmp;
}
```

هذا التابع يبدأ من آخر بايت في الدخل إذا لاحظ وجود القيمة 0 يزيد قيمة عداد معرف مسبقا يحسب عدد الأصفار وهكذا تنازليا إلى أن يحسب عدد الأصفار وهو يشير إلى كمية الحشو ثم تعريف مصفوفة طولها هو الطول الحقيقي للبلوك 16 بايت مطروح منه قيمة العداد1 – ثم استخدمنا التابع *arraycopy*المعرف في java وله الشكل:

public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

حيث البارامترات هي:

- *Input* وهو المصفوفة المصدر أي التي سننسخ منها
- 0: وهو يشير إلى أنه سيتم بدأ النسخ من العنصر ذو الدليل 0 أي العنصر الأول في المصفوفة input
- Tmp : وهي مصفوفة الوجهة التي سيتم النسخ إليها
- 0 : وهو الرقم الذي سيبدأ اللصق عنده في مصفوفة الوجهة
- *tmp.length* : وهو طول المصفوفة التي نسخنا إليها

ثم قمنا ب إعادة المصفوفة *tmp* وهو البلوك الأصلي خاليا من الحشو

8.4 - مثال عددي لتمثيل دخل الخوارزمية والعمليات عليه :

ان خوارزمية التشفير بصورة عامة للصور تشبه في مضمونها خوارزمية تشفير النص ولكنها تختلف في التعامل حيث ان ملف الصورة يعامل بصورة مختلفة عن ملف النص، حيث تركز عملية التشفير في الصورة على تشفير البتات لكل بكسلات الصورة.

هنا تطبيق عملي لخطوات الخوارزمية خطوة خطوة على نص ومفتاح مدخلين من قبل المستخدم ، ولدي تابع يحول الدخل و المفتاح إلى ست عشري. ليكن الدخل التالي:

- *Input: baraa Mohamad nassif (18 bytes)*
- *Key: aaaaaaaaaaaaaaaa (16 bytes)*

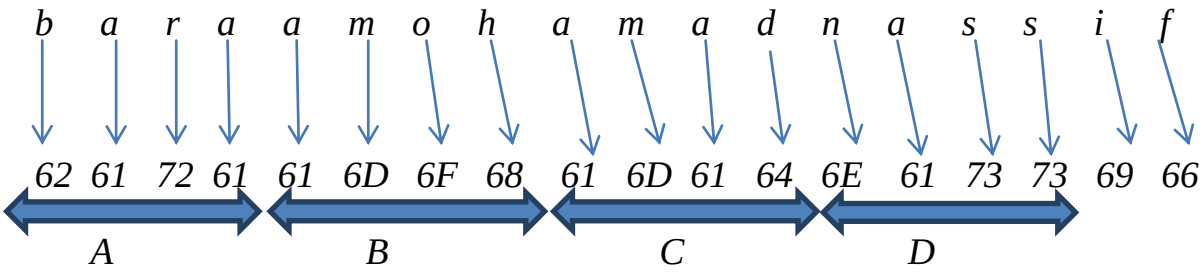
كل حرف يمثل ببايت واحد (*8 bits*) ، نحول حروف الدخل إلى نظام *ASCII* ونأخذ تمثيله بالنظام الست عشري.

فمثلاً الحرف b رمزه ب *ASCII* هو98 وتمثيله الست عشري هو 62

الحرف a رمزه ب *ASCII* هو 97 وتمثيله الست عشري هو 61

الحرف r رمزه ب *ASCII* هو 114 وتمثيله بالست عشري 72 وهكذا بالنسبة لبقية الدخل.....

فيكون التمثيل للدخل بالشكل:



نلاحظ أن كتلة الدخل أكبر من 16 بايت لذلك نقسم الدخل إلى كتلتين كل منهما طولها *16 bytes* ونعمل على كل كتلة على حدى ،حيث نضيف إلى الكتلة الثانية حشو لتناسب طول دخل الخوارزمية.

- نقسم كتلة الدخل الأولى إلى 4 كلمات *A , B, C, D* كما لاحظنا على الشكل السابق وأوجدنا التمثيل الست عشري لكل حرف.

تمثيل الكلمة A بالثنائي :

01100010 01100001 01110010 01100001 01100001

- نقسم المفتاح كذلك الأمر إلى 4 كلمات s[0], s[1], s[2], s[3] ، ثم نوجد التمثيل الست عشري للمفتاح بما أن المفتاح *aaaaaaaaaaaaaaaa* فتمثيله الست عشي كالتالي:

61616161616161616161616161616161

- نأخذ الجزء B ونجري عليه أولى الخطوات وهي دمجه مع المفتاح *s[0]* بالشكل

B =B + s[0]

61 6D 6F 68

61 61 61 61 +

C2 CE D0 C9 = B

- الآن ندخل حلقة for (التكرار الأول) حيث تعاد الخطوات التالية على B حتى التكرار 20 .
- تدخل النتيجة السابقة إلى دالة f لتجري عليها عمليات معقدة حيث *f(x) = x*(2x+1)* كالتالي:

B C2 CE D0 C9

02 X

---

85 9D A1 92

01 +

---

85 9D A1 93

C2 CE D0 C9 X

---

80 8C 80 81

10000000100011001000000010000001

- الناتج السابق نجري عليه عملية إزاحة دورانية نحو اليسار بمقدار *5 bits* ، ونخزن النتيجة في t .

t= (B(2B+1))<<< lgw*

حيث *w* هو طول الكلمة بالبتات *(32 bits)* ، فيكون الناتج بعد الإزاحة الدورانية نحو اليسار بمقدار 5 بتات:

t = 00010001100100000001000000110000

11 90 10 30

- الأن نعمل على الجزء *A* نقوم بعملية *XOR* بين A و t:

b a r a

62 61 72 61

A 01100010011000010111001001100001

t 00010001100100000001000000110000 ⊕

01110011111100010110001001010001

- الناتج السابق نزيحه بمقدار u سنحسبها لاحقاً.

- الآن العمليات على D هي العمليات نفسها التي تمت على B وفق التالي:
- نضيف للجزء D المفتاح s[1] كالتالي:

D=D+ S[1]

D 6E 61 73 73

S[1] 61 61 61 61 +

CF C2 D4 D4

الآن ندخل الحلقة والعمليات التالية تكرر على *D* خلال 20 دورة:

- تدخل *D* إلى التابع *f* حيث *f(x) = x*(2x+1)*

D CF C2 D4 D4

02 X

9E 85 A9 A8

01 +

9E 85 A9 A9

CF C2 D4 D4 X

8E 80 80 80

- نزيح الناتج السابق بمقدار 5 بتات إزاحة دورانية نحو اليسار :

8E 80 80 80

10001110100000001000000010000000

$u =$ *11010000000100000001000000010001*

- الان نعود الى الجزء A بعد عملية XOR مع t يزاح الناتج بمقدار u التي حسبناها الان

لكن كيف تتم عملية الإزاحة؟ هنا نعتمد على البتات الخمسة الأقل أهمية من u وحسب قيمتهم تكون الازاحة.

$A \oplus t =$ *73 F1 62 51*

01110011111100010110001001010001

البتات الخمسة الأقل أهمية من u هي *10001*

- ناتج الإزاحة هو :

11000100101000101110011111100010

C4 A2 E7 E2

- هذا الناتج نضيف له المفتاح $s[2]$ (في كل دورة نضيف للجزء A المفاتيح ذات الدليل الزوجي).

C4 A2 E7 E2

61 61 61 61 +

---

26 04 49 43

- نخزن الناتج السابق فيD $A \rightarrow D$

- الآن الجزء C :

تبدأ الحلقة عنده بعمليات مشابهة لتلك التي أجريناها على A تماماً.

- نبدأ بعملية XOR مع u التي حسبناها مسبقاً:

$$C \qquad 61\ 6D\ 61\ 64$$

$$u \qquad D0\ 10\ 10\ 11 \qquad \oplus$$

$$01100001011011010110000101100100$$

$$11010000000100000001000000010001 \quad \oplus$$

$$10110001011111010111000101110101$$

- هذا الناتج نزيحه بمقدار t التي حسبناها سابقاً أي حسب البتات الخمسة الاقل تأثيراً في t :

$$t \quad = \quad 11\ 90\ 10\ 30$$

البتات الخمسة الأقل أهمية في t هي 10000 .

- فيكون ناتج الإزاحة :

$$01110001011101011011000101111101$$

$$71\ 75\ B1\ 7D$$

- نضيف للناتج السابق المفتاح $s[3]$ (حيث في كل دورة يضاف للجزء C المفاتيح ذات الدليل الفردي)

$$71\ 75\ B1\ 7D$$

61 61 61 61 +

D2 D7 12 DE

- نخزن الناتج السابق في B $C \rightarrow B$

- تعاد العمليات السابقة ما عدا إضافة المفتاحين *s[0], s[1]* على الجزأين B, D على الترتيب تعاد على كل جزء من أجل كل دورة ، حتى نصل إلى 20 دورة ، لا ننسى أنه في كل مرة يحصل تبديل بين أجزاء الكلمة بالشكل:

$A \rightarrow D$, $B \rightarrow A$, $C \rightarrow B$, $D \rightarrow C$

عندما نصل للدورة 20 وبعد الانتهاء من عمليات الإضافة و *XOR* و الإزاحة على كل جزء يكون لدينا :

- الكلمة B قبل تخزينها في A نضيف لها المفتاح قبل الأخير من المفاتيح الموسعة المفتاح *s[42]* ثم نخزنها في A لتكوين النتيجة النهائية.
- الكلمة D قبل تخزينها في C نضيف لها المفتاح الأخير من المفاتيح الموسعة *s[43]* ثم نخزنها في C لتكوين النتيجة النهائية للتشفير.

حيث نكون قد استخدمنا خلال دورات الخوارزمية المفاتيح :

s[2], s[3], s[4], s[5] , …… , s[41].

وقبل الدخول بالحلقة المفاتيح:

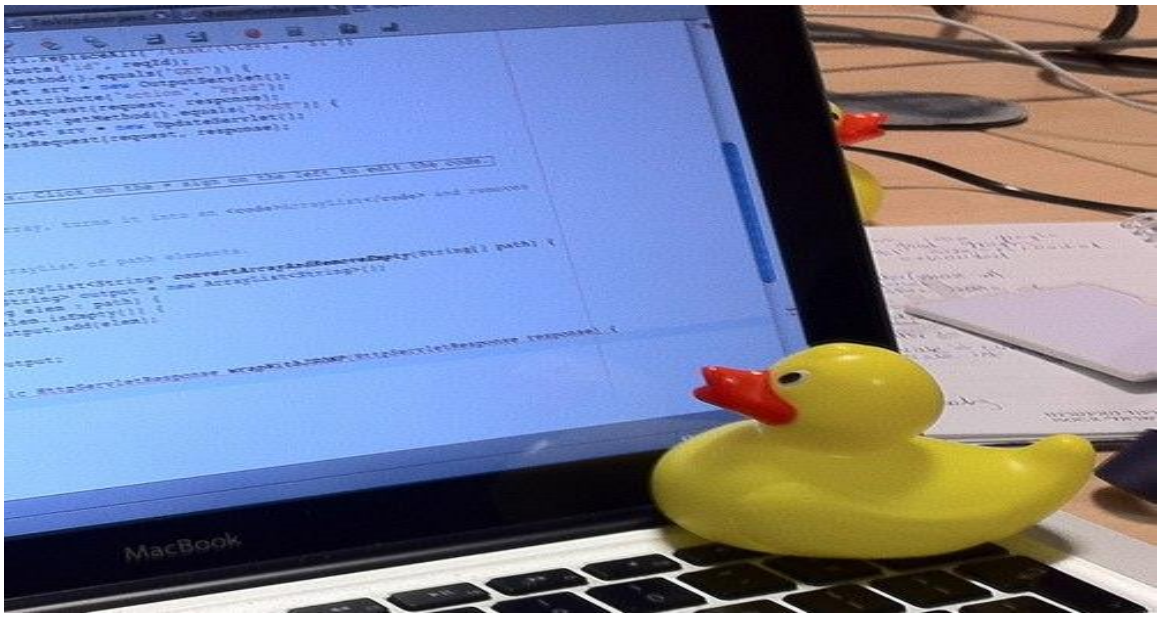
s[0] , s[1].

وبعد الانتهاء من تنفيذ دورات الخوارزمية كلها المفاتيح:

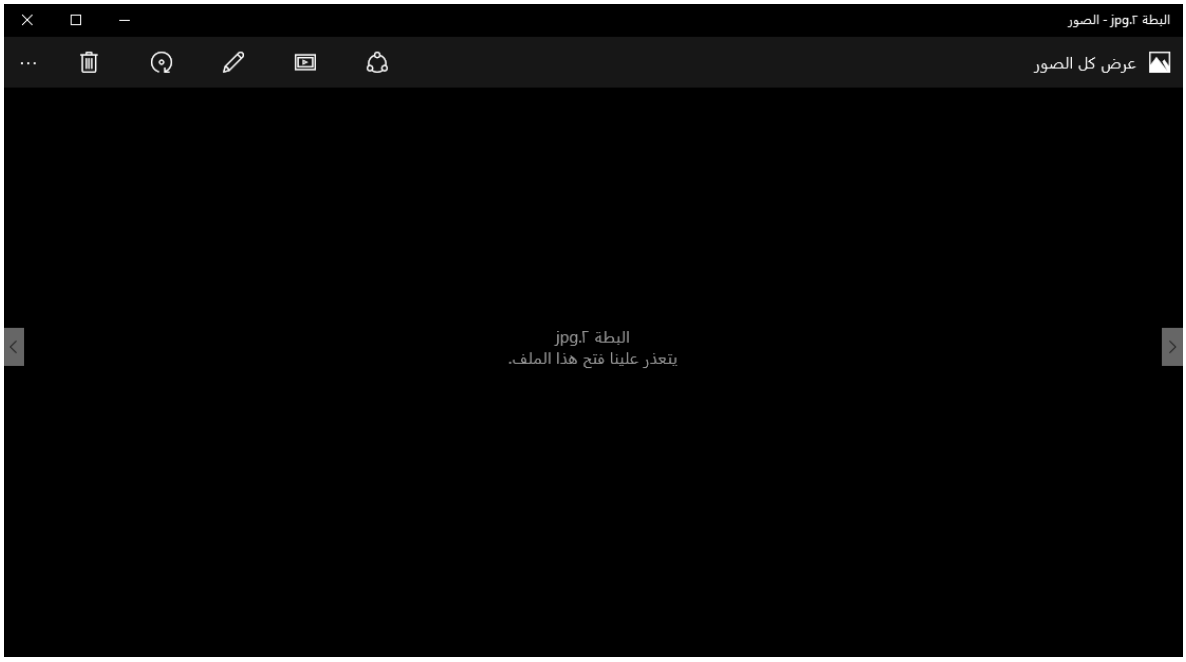
$s[42], s[43]$.

9.4 – تطبيق عملي :

الصورة التالية توضح عملية التشفير باستخدام خوارزمية *RC6*



شكل 4–4: الصورة الاصلية قبل التشفير



شكل4-5:الصورة بعد التشفير

الفصل الخامس

خوارزمية *RSA (Rivest-Shamir-Adleman)*

هي خوارزمية للتشفير بواسطة مفتاح عام. كانت القاعدة الأولى المعروفة بكونها مناسبة للتوقيع بالإضافة إلى التشفير، وكانت أحد التقدّمات العظيمة الأولى في التشفير بواسطة مفتاح عامّ. وهي مستخدمة في بروتوكولات التّجارة الإلكترونية على نطاق واسع، وهي مضمونة على اعتبار انه يوجد مفاتيح طويلة بشكل كافي . [7]

وهي أحد خوارزميات التشفير غير المتناظر تستخدم في التشفير و التوقيع الرقمي ،فالهدف من التشفير الحماية والهدف من التوقيع إثبات هوية المرسل و عدم الإنكار، حيث تبقى بنية الخوارزمية نفسها وما يختلف هو المفاتيح المستخدمة (مفاتيح المرسل) .[8].

سميت نسبة إلى العلماء الثلاثة الذين ابتكروا هذه الخوارزمية ، وُصِفَت علناً في عام *1977* من قبل ليونارد أدليمان وأدي شامير ورون ريفيست في معهد مساشوستس للتكنولوجيا [6]، وهي عبارة عن خوارزمية تشفير مبنية على الأعداد الأولية تقوم بإنتاج مفتاحين أحدهما هو المفتاح العام *Public Key*الذي يشفر به الرسالة والآخر المفتاح الخاص *Private Key* وهذا الأخير يتم الحصول عليه عن طريق خوارزمية *Extending Key algorithm* أو ما تعرف بخوارزمية إقليدس.

تأخذ حمايتها من تصنيع الأعداد الصحيحة الكبيرة الناتجة من عددين كبيرين أوليين ، جداؤهما سهل لكن إيجاد قيمة العددين من خلال هذا الناتج صعب يحتاج لساعات طويلة من الجهد وحواسيب ذات قدرات فائقة. [6]

طبعاً خوارزمية *RSA* أبطأ من ال *DES* والعديد من الخوارزميات المتناسقة، لكنها صعبة الكسر نوعاً ما .

يتم تحقيق الخوارزمية في مشروعنا من خلال استخدامها في تشفير مفتاح خوارزمية RC6 المكون من 16 بايت ، وذلك من أجل إرساله إلى الجهة المطلوبة بأمان واستخدامه بعد فك تشفيره.

1.5 – طريقة عمل الخوارزمية:

خوارزمية RSA تستخدم المفتاح العام والمفتاح الخاص للمستقبل . المفتاح العام يمكن أَنْ يعرف إلى كل شخص حيث يستعمل لتَشْفير الرسائل. الرسائل المشفرة بالمفتاح العام يمكن أن تفك فقط باستخدام المفتاح الخاص.

1.1.5 – توليد المفاتيح:

المفاتيح لقاعدة *RSA* تم توليدها بالطريقة التالية: [8]

- نختار عددين أوليين عشوائيين كبيرين مختلفين q و p (يفضل أن يكونا أعداد كبيرة)
- حساب $n = p * q$

  n يستخدم كالمعامل لكلا المفاتيح الخاصّة والعامّة .
- نحسب $\varphi(n) = (p-1)(q-1)$.
- نختار عدد أولي e ضمن المجال $1<e<\varphi(n)$ بحيث يحقق الشرط $gcd(e, \varphi(n)) = 1$
- نحسب $d = e^{-1} \bmod \varphi(n)$.

بالتالي يصبح المفتاح العام (e, n) يتكون من المعامل n و الأس e

والمفتاح الخاص (d, n) يتكون من المعامل n والأُس d .

ملاحظة 1 : المفتاح العام والخاص كل منهما معكوس للآخر لهذا هما يحققان $d * e = 1$

ملاحظة 2 : في التشفير و فك التشفير نستخدم مفاتيح المستقبل حيث نستخدم المفتاح العام للمستقبل للتشفير وهو متوفر للجميع ويستطيع أي أحد إرسال الرسائل للمستقبل ، وفك التشفير سيتم في طرف المستقبل باستخدام المفتاح الخاص أي هو متاح للمستقبل فقط . *[8]*

2.1.5- تشفير الرسائل : [7]

في جهة المرسل A يقوم بالخطوات التالية:

- يحصل على المفتاح العام للمستقبل B والذي هو (e , n) .
- يحول الرسالة من لغة إلى رقم صحيح عن طريق بروتوكول قابل للعكس
- يوجد ناتج التشفير لهذا الرقم عن طريق المعادلة $C = M \wedge e \bmod n$.

  حيث C : هي النص المشفر

  M : هي النص الأصلي ،

  e : هي المفتاح العام للمستقبل.

  n : هي المقياس الذي نعمل به نحسبه مسبقاً.
- إرسال الناتج عن التشفير C إلى المستقبلB .

3.1.5 - فك تشفير الرسائل: [7]

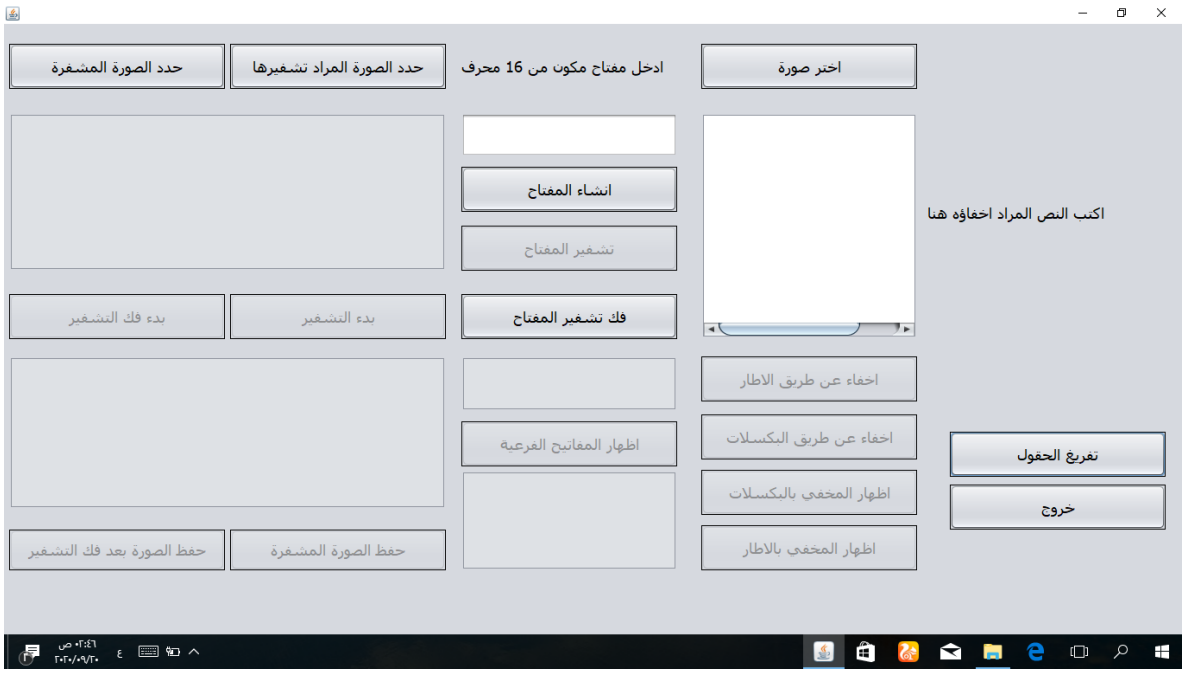
في جهة المستقبل B يقوم بالخطوات التالية:

- يستخدم مفتاحه الخاص (n, d) لحساب $M = C \wedge d \bmod n$ أي إيجاد النص الأصلي .
- يستخلص اللغة أو محتوى الرسالة الأصلي من الناتج M.

الفصل السادس

تطبيق على واجهة البرنامج:

عند تشغيل البرنامج تظهر الواجهة بالشكل التالي:



شكل 6-1 :الواجهة الرئيسية للبرنامج.

كما نلاحظ أنه يمكننا البدء بإخفاء النص من الصورة ومن ثم تشفيرها، أو ببساطة يمكننا التشفير مباشرة.

- إدخال النص المراد إخفاؤه:

شكل 6-2: ادخال النص المراد إخفاؤه وتحديد خوارزمية الإخفاء.

تركنا للمستخدم حرية اختيار أي خوارزمية يريدها للإخفاء.

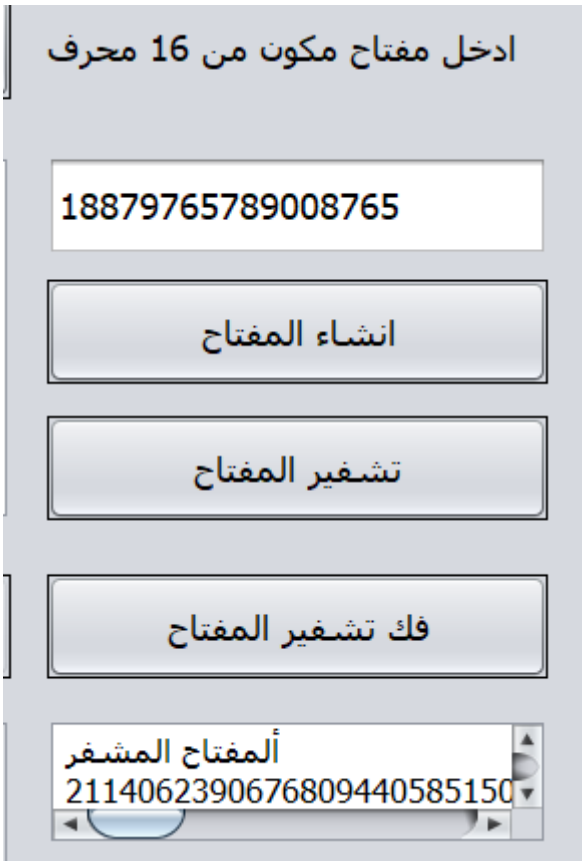
- الآن إدخال المفتاح:

شكل 3-6 :إدخال المفتاح وإنشاء المفاتيح الفرعية.

- تحديد الصورة المراد تشفيرها وتشفيرها بالمفتاح المدخل سابقاً وإظهار الوقت المستغرق:

شكل *6-4*: تحديد الصورة المراد تشفيرها وبدء عملية التشفير.

- تشفير المفتاح المستخدم في خوارزمية الـ*RC6* باستخدام خوارزمية الـ *RSA*



شكل6-5: تشفير مفتاح الـ *RC6*

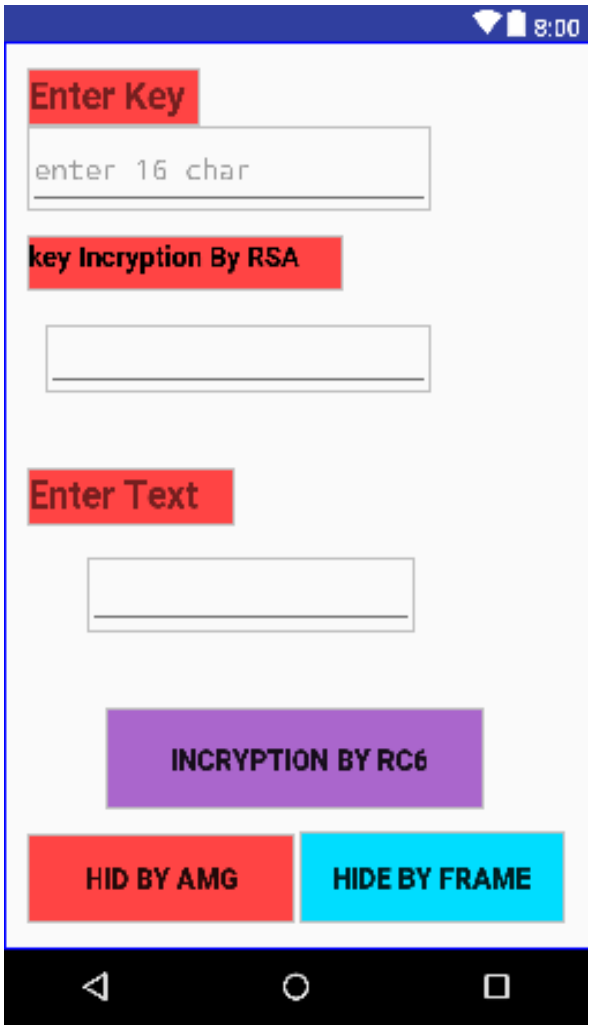
الفصل السابع :

تحقيق المشروع باستخدام نظام الاندرويد

- ❖ عند تشغيل البرنامج تظهر الواجهة التالية

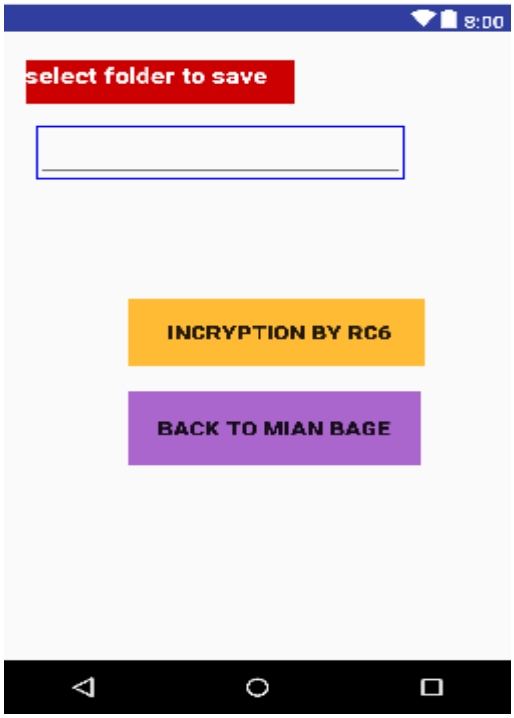
شكل 7-1: الواجهة الرئيسية للتطبيق

❖ تحديد مفتاح التشفير النص المراد تشفيره:



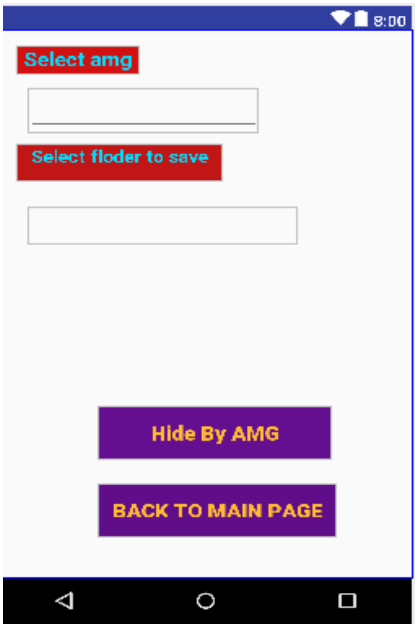
شكل7-2: تحديد مفتاح التشفير و النص المراد تشفيره

❖ التشفير:



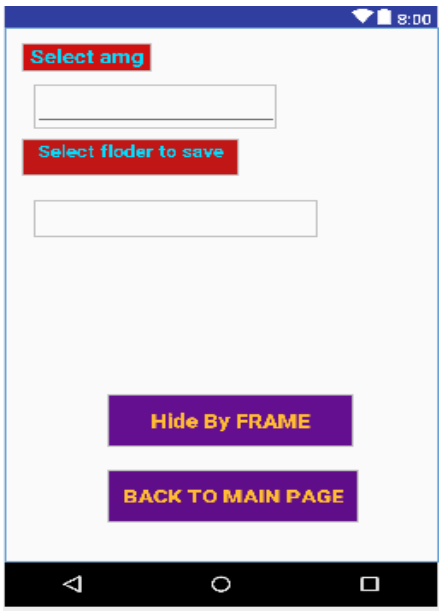
شكل 7-3: التشفير باستخدام خوارزمية RC6

❖ الإخفاء ضمن البيكسلات:



شكل 7-4: اخفاء النص ضمن بكسلات الصورة

❖ الإخفاء ضمن الإطار:

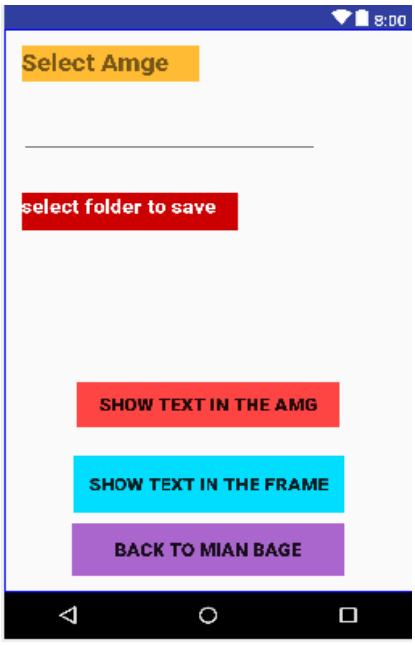


شكل 7-5: اخفاء النص ضمن الاطار

❖ فك تشفير المفتاح بواسطة خوارزمية RSA

شكل 7-6: فك تشفير المفتاح باستخدام خوارزمية RSA

❖ إظهار النص المخفي ضمن الصورة أو الإطار:



شكل 7-7 : اظهار النص المخفي ضمن الصورة او الاطار

المراجع

1. *https://3alam.pro/3mmarg97/articles/steganography-using-lsb-implementation-in-php*

2. *http://hussienahmmed.blogspot.com/2013/03/hide-text-in-image-by-matlab.html* حسين الربيعي الأحد، ١٧ مارس ٢٠١٣

3. *The RC6 TM Block Cipher Ronald L. Rivest1, M.J.B. Robshaw2, R. Sidney2, and Y.L. Yin2 Version 1.1 - August 20, 1998*

4. *تشفير صورة Image Encoding بواسطة خوارزمية RC6 ميمونة الحداد*

5. *RC6 Implementation including key scheduling using FPGA (ECE 646 Project, December 2006) Fouad Ramia, Hunar Qadir, GMU*

6. *https://searchsecurity.techtarget.com/definition/RSA*

7. https://www.startimes.com/?t=24133011

8. مقرر أمن المعلومات العام الدراسي ٢٠١٩/٢٠٢٠ د. بسيم برهوم