INDEX

`ABSTRACT

Image, which covers the highest percentage of the multimedia data, its protection is very important. This can be achieved by image encryption. As the use of digital techniques for transmitting and storing image are increasing, it becomes an important issue that how to protect the confidentiality, integrity, and authenticity of image. There are various techniques which are discovered from time to time to encrypt the images to make images more secure. Image encryption techniques scrambled the pixels of the image and decrease the correlation among the pixels, so that we will get lower correlation among the pixel and get the encrypted image. There are so many different technique used to protect confidential image data from unauthorized access. In this paper, we have done the literature review on existing work which was done form different techniques for image encryption and also given the general introduction about images and image encryption with advantages and disadvantages.

KEYWORDS: Cryptography, Image, Encryption, Security and AES

INTRODUCTION

Images are generally the collection of pixels. Basically Image Encryption means that convert the image into unreadable format. Many digital services require reliable security in storage and transmission of digital images. Due to the rapid growth of the internet in the digital world today, the security of digital images has become more important and attracted much attention. The prevalence of multimedia technology in our society has promoted digital images to play a more significant role than the traditional texts, which demand serious protection of user's privacy for all applications.

Encryption techniques of digital images are very important and should be used to frustrate opponent attacks from unauthorized access. Digital images are exchanged over various types of networks. It is often true that a large part of this information is either confidential or private.

Encryption is the preferred technique for protecting the transmitting data. There are various encryption systems to encrypt and decrypt image data. However, it can be argued that there is no single encryption algorithm which satisfies the different image types. In general, most of the available encryption algorithms are used for text data. However, due to large data size and real time constants, algorithms that are good for textual data may not be suitable for multimedia data. Even though tripledata encryption standard (T-DES) and international data encryption algorithm (IDEA) can achieve high security, they may not be suitable for multimedia applications Therefore, encryption algorithms such as Data Encryption Standard (DES), Advanced Encryption Standard (AES), and International Data Encryption Standard (IDEA) were built for textual data. Although we can use the traditional encryption algorithm to encrypt images directly, this may not be a good idea for two reasons. First, the image size is often large than text. Consequently, the traditional encryption algorithms need a longer time to directly encrypt the image data.

Second, the decrypted text must be equal to the original text but this requirement is not necessary for image data.

Due to characteristic of human perception, a decrypted image containing small distortion is usually acceptable. The intelligible information in an image is due to the correlation among the image elements in a given arrangement. This perceptive information can be reduced by decreasing the correlation among image elements using certain transformation techniques. In addition to cryptography, chaotic image transformation techniques are getting significantly more sophisticated and have widely used. The chaotic image transformation techniques are perfect supplement for encryption that allows a user to make some transformation in the image, and then the image is totally diffracted, so nobody could see that what information could be shown through that image. Thus, it is often used in conjunction with cryptography so that the information is doubly protected, that is, first it is transformed by chaotic map transformation techniques, and then it is encrypted so that an adversary has to find the hidden information before the decryption takes place.

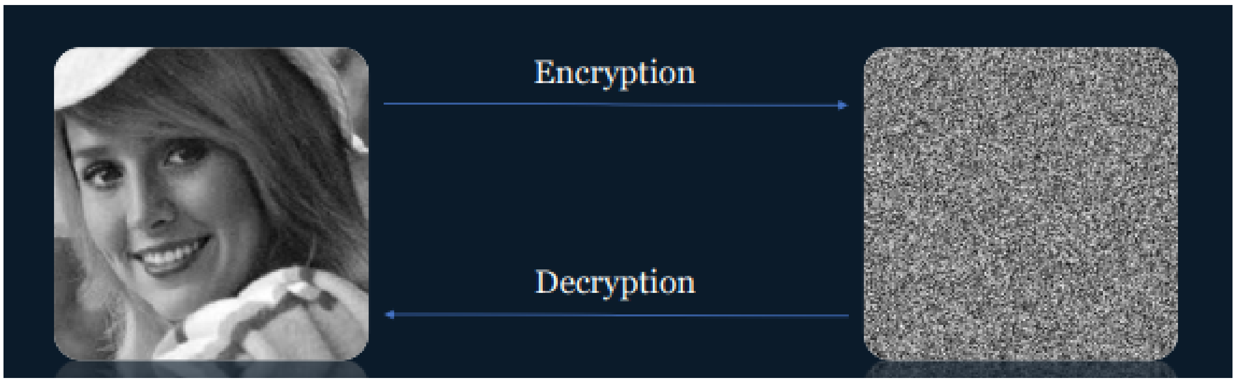
LITERATURE SURVEY

As we said the significance of network security is increased day by day as the size of data being transferred across the Internet. This issue pushes the researchers to do many studies to increase the ability to solve security issues. A solution for this issue is using the advantage of cryptography and steganography combined in one system. Many studies propose methods to combine cryptography with steganography systems in one system. This Project has been implemented on the basis of the requirements of security i.e. authentication, confidentiality, and robustness. There has been a continuous rise in the number of data security threats in the recent past and it has become a matter of concern for the security experts. Cryptography and steganography are the best techniques to nullify this threat. The researchers today are proposing a blended approach of both techniques because a higher level of security is achieved when both techniques are used together. In proposed an encrypting technique by combining cryptography and steganography techniques to hide the data. In cryptography process, we proposed an effective technique for data encryption using one's complement method. It used an Asymmetric key method where both sender and receiver share the Secret key for encryption and decryption. In steganography part, we used the LSB method that is used and mostly preferred.

OBJECTIVE

Encrypting and Decrypting an image using A5/1 stream cipher in Verilog, for over-the-air communication privacy in GSM cellular telephone standard

Given a 256 x 256 grayscale image which we want to encrypt and decrypt using **A5/1 stream cipher.**



PROBLEM

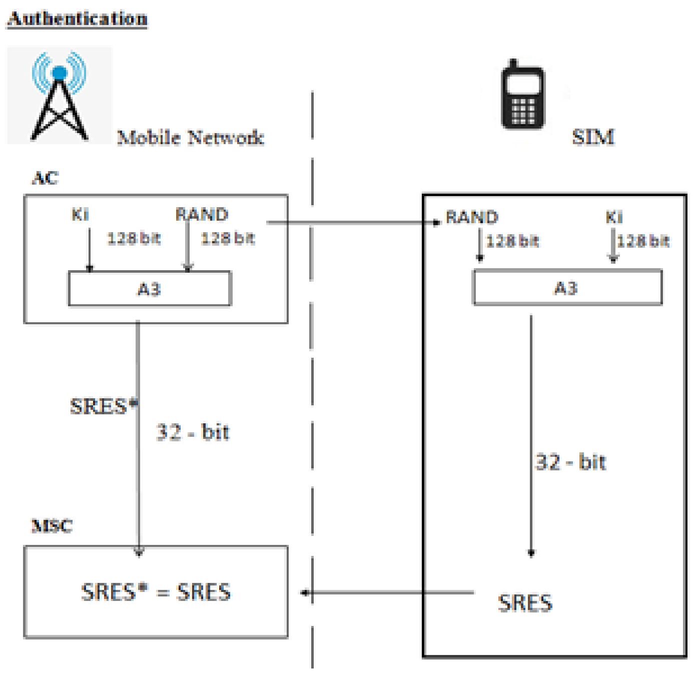
PROBLEMS IN GSM SECURITY:

1. Unilateral authentication & vulnerability to the man-in-the-middle: This is the network that verifies users. The user does not verify network so the attacker can use a false BTS (Base Transciever Station) with the same mobile network and perform a man-in-the-middle attack. The attacker can perform various structure to modify the exchanged data.

2. SIM card cloning: This attack can be indicated as a more active attack because of the attacker clone the SIM and use it for his unlawful purposes. The attacker gets the data from the Auc servers of the user's network operator and with the help of that data, the attacker listens to the user's communications.

3. Over-the-air cracking: This can be practised by sending various challenges over the air to the SIM and examine the responses . after finding Ki and IMSI of the target subscriber, the attacker can clone the SIM and make other services such as SMS in the name of the victim subscriber. The attacker will experience a minor problem. The GSM network allows only one SIM to access to the network so if the attacker and the victim subscriber try to access the SIM from a different location, the network will detect the duplication and damage the affected account.

4. User Authentication in GSM:



As you can see in figure that how authentication has been done in Mobile network and SIM (End user). The mobile network generates a random number(RAND) of 128bits then send it to SIM. SIM receives those RAND number from a mobile network. SIM is having Ki (ki is Individual subscriber authentication number) which is different for a person to person. Every different SIM having different Ki number. Ki is assigned by Mobile network. SIM took RAND and Ki (authentication number) of 128 bits then we apply the A3 algorithm on it which is used for Authentication. Then SIM generates SRES(Signal response) of 32bits. Parallel, the mobile network takes RAND and Ki of 128bits to use the data to produce an ideal signal response by using A3 also. Then both the SRES are stored in MSC(Mobile switching centre) in which we check that whether the SRES(Signal response) of mobile network and SIM are equal or not. Once access control (AC) found both SRES are same then it connect the call.

SOLUTION PROPOSED

The world we live in is nothing but a huge web of networks. Today, billions of bits are generated and communicated daily.

Communication is a strong and useful tool that connects the entire human race in a single string. Any communication medium depends on a number of publicly active aspects for the transmission of data from one end to the other. The data may be extremely sensitive ranging from a simple account password to information related to nuclear weapons.

As the data needs to travel through different public domains to reach the destination, it becomes important to provide for some mechanism for secure transmission of this data.

The answer to these concerns is simple. **What if we encrypt the data at the source end and then decrypt it (i.e. convert it back into the original form) at the receiver's end?** This way the data, while in midst of transmission, is in random and useless form and so **communication is secured.**

There are many schemes to encrypt data but not all can be efficiently modelled as hardware circuits. In this project we will be using a famous Stream Cipher that is used for over-the-air communication privacy in the GSM cellular telephone standard.

This Cipher is called A5/1 algorithm.

A5/1 SYMMETRIC CRYPTOGRAPHY ALGORITHM

A5/1 Cipher

A5/1 is a symmetric stream encryption algorithm (cipher) which is hardware-based and is used for confidentiality in GSM cell phones. It is called hardware based because it can be implemented using the instruction set on computer architecture.

Working of A5/1

It employs three Linear Feedback Shift Registers (LFSRs) X,Y, and Z.

The three LFSRs X, Y and Z have lengths equal to 19, 22, and 23 bits respectively.

Register X can be represented as (x_0, x_1, x_2, …., x_{18}),
register Y can be represented as (y_0, y_1, y_2, …., y_{21}), and
register Z can be represented as (z_0, z_1, z_2, …., z_{22}).

Note: 19 + 22 + 23 = 64 bits

A 64-bit key K is used as the initial values in X, Y, and Z registers. After the registers are filled with the key, the keystream is generated through these three registers. The keystream is then XORed with the input bitstream to get the encrypted stream.

Functioning and diagrammatic representation of three LFSRs X, Y and Z in A5/1

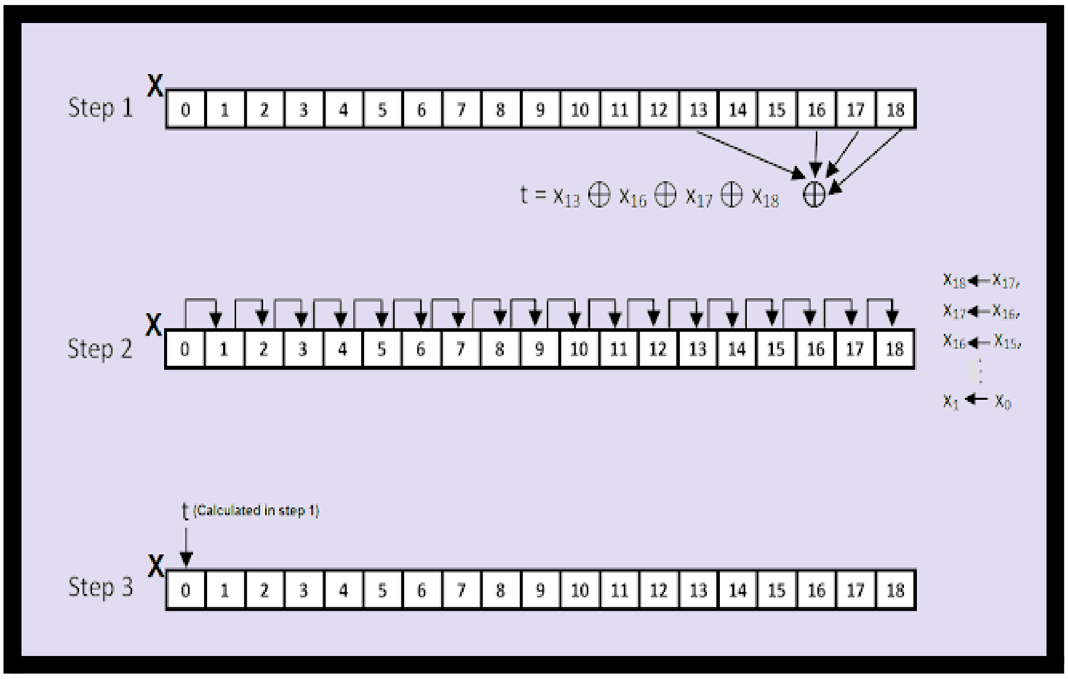
Functioning of Register X:

The 14th, 17th, 18th and 19th (x_{13}, x_{16}, x_{17}, x_{18}) bits of register X are XORed and result is saved in a temporary variable (say t). Thereafter, the bits are shifted 1-position to right direction, i.e. bit x_{17} is shifted to x_{18}, bit x_{16} is shifted to x_{17} and so on till x_0 is shifted to x_1. After shifting, the bit saved in temporary variable t is saved in x_0. It can be equated and diagrammatically represented as:

$t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$

$x_i = x_{i-1}$ for i = 18, 17, 16, …, 1

$x_0 = t$


Step 1
X
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
$t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
Step 2
$x_{18} \leftarrow x_{17}$,
$x_{17} \leftarrow x_{16}$,
$x_{16} \leftarrow x_{15}$,
$x_1 \leftarrow x_0$
Step 3
t (Calculated in step 1)

Shifting of Register X

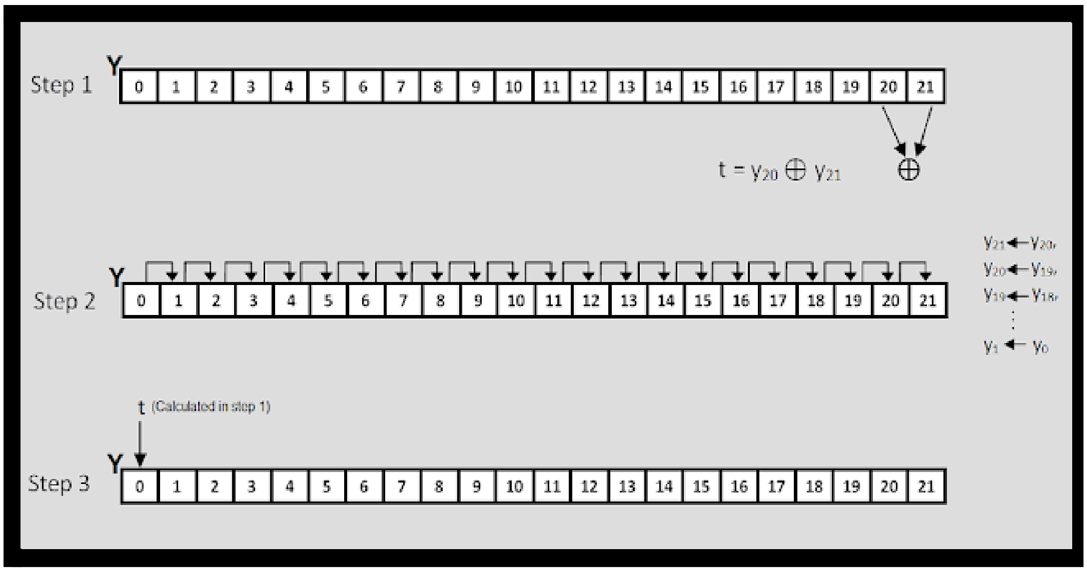
Functioning of Register Y:

The 21st and 22nd bits (y_{20}, y_{21}) of register Y are XORed and result is saved in a temporary variable (say t). Thereafter, the bits are shifted 1-position to right direction, i.e. bit y_{20} is shifted to y_{21}, bit y_{19} is shifted to y_{20} and so on till y_0 is shifted to y_1. After shifting, the bit saved in temporary variable t is saved in y_0. It can be equated and represented diagrammatically as:

$t = y_{20} \oplus y_{21}$

$y_i = y_{i-1}$ for i = 21, 20, 19, …., 1

$y_0 = t$



Step 1
Y
$t = y_{20} \oplus y_{21}$
Step 2
$y_{21} \leftarrow y_{20}$,
$y_{20} \leftarrow y_{19}$,
$y_{19} \leftarrow y_{18}$,
$y_1 \leftarrow y_0$
t (Calculated in step 1)
Step 3

Shifting of Register Y

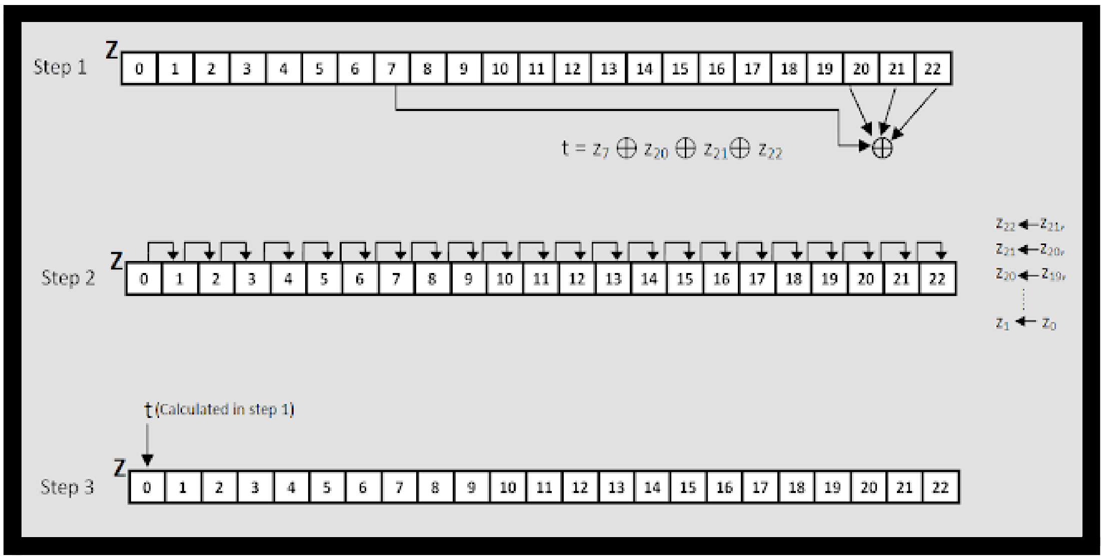
Functioning of Register Z:

The 8th, 21st, 22nd and 23rd bits (z_7, z_{20}, z_{21}, z_{22}) of register Z are XORed and result is saved in a temporary variable (say t). Thereafter, the bits are shifted 1-position to right direction, i.e. bit z_{21} is shifted to z_{22}, bit z_{20} is shifted to z_{21} and so on till z_0 is shifted to z_1. After shifting, the bit saved in temporary variable t is saved in z_0. It can be equated and represented diagrammatically as:

$$t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$$

$$z_i = z_{i-1} \text{ for } i = 22, 21, 20, \ldots, 1$$

$$z_0 = t$$



Shifting of Register Z

Keystream Generation

In the generation of keystream, not all three X, Y and Z registers shift. The shifting of registers will be based on the function maj(x_8, y_{10}, z_{10}) which is computed as:

If the majority of bits x_8, y_{10}, and z_{10} are 0, the function returns 0; If the majority of bits x_8, y_{10}, and z_{10} are 1, the function returns 1.

In A5/1, for each keystream bit that we generate, the following takes place. First, we compute

$$\boxed{m = \text{maj}(x_8, y_{10}, z_{10})}$$

Then the registers *X, Y,* and *Z* shift (or not) as follows:
• If $x_8 = m$ then *X* shifts, i.e. if the 9th bit of register X is equal to m, the shift operation is applied to register X.
• If $y_{10} = m$ then *Y* shifts, i.e. if the 11th bit of register Y is equal to m, the shift operation is applied to register Y.
• If z_{10} = m then *Z* shifts, i.e. if the 11th bit of register Z is equal to m, the shift operation is applied to register Z.

A few examples of registers' shift using majority function m:

x_8	y_{10}	z_{10}	**m**	**Registers to shift**
0	0	0	0	X, Y, Z
1	1	1	1	X, Y, Z
0	0	1	0	X, Y

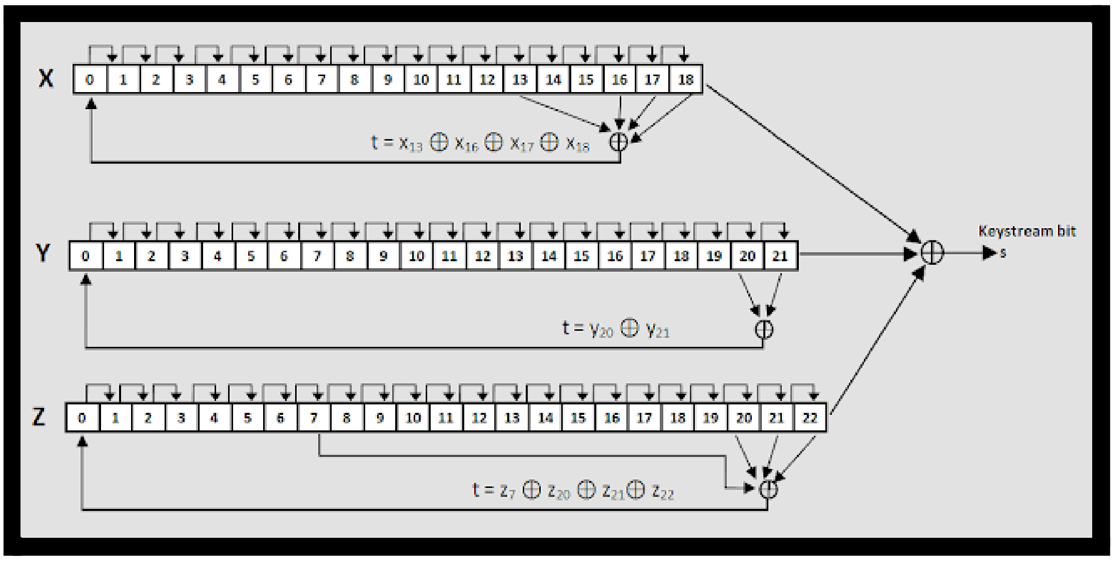
1	0	1	1	X, Z
0	1	0	0	X, Z

Finally, a single keystream bit *s* is generated using the least significant bits from all three registers respectively as:

$$s = x_{18} \oplus y_{21} \oplus z_{22}$$

This generates only one keystream bit s. The whole keystream S of length n, equal to plaintext, is generated by applying the same steps n times.

The wiring diagram of A5/1 algorithm can be shown as:



A5/1 Algorithm

Generating Ciphertext

The keystream generated in the A5/1 algorithm is then XORed with the plaintext to get the ciphertext. That is,

$$\text{plaintext} \oplus \text{keystream} = \text{ciphertext}$$

On the other hand, at the destination, keystream is generated using the same 64 bit key in A5/1 algorithm which is then XORed with the ciphertext to get the plaintext. That is,

$$\text{ciphertext} \oplus \text{keystream} = \text{plaintext}$$

METHODOLOGY

We break the image into 8 bit-planes.

Each plane contains 256 x 256 bits. For example, if the first pixel has value 8’b0111_0011 then the first plane will contain only 1’b0, second plane will contain 1’b1, third again 1’b1 and so on.
Similarly for other pixels.

Thus, our image will be actually broken into 8 planes of 256 x 256 pixels with each pixel only 1 bit in length.
So the first bit of every pixel goes into the first plane with the plane number as 1 which becomes the last four bits of the public key and similarly for the other bits of every pixel.
Now we simply encrypt the image plane by plane.

But to make the cipher strong, before the start of a new plane, **we re-initialize the LFSRs following the same steps mentioned earlier**.

LOADING IMAGE IN VERILOG

- The 256*256 image cannot be loaded into Verilog directly.

- The only way to load the image in Verilog is to first serially store all the pixel values by converting into hexadecimal(base 16) in a hexfile.txt

- Then Verilog can read the contents of the file using the function

  $readmemh("file_name",memory register)

- We converted the image to this required hexfile.txt using MATLAB the code for which is present below.

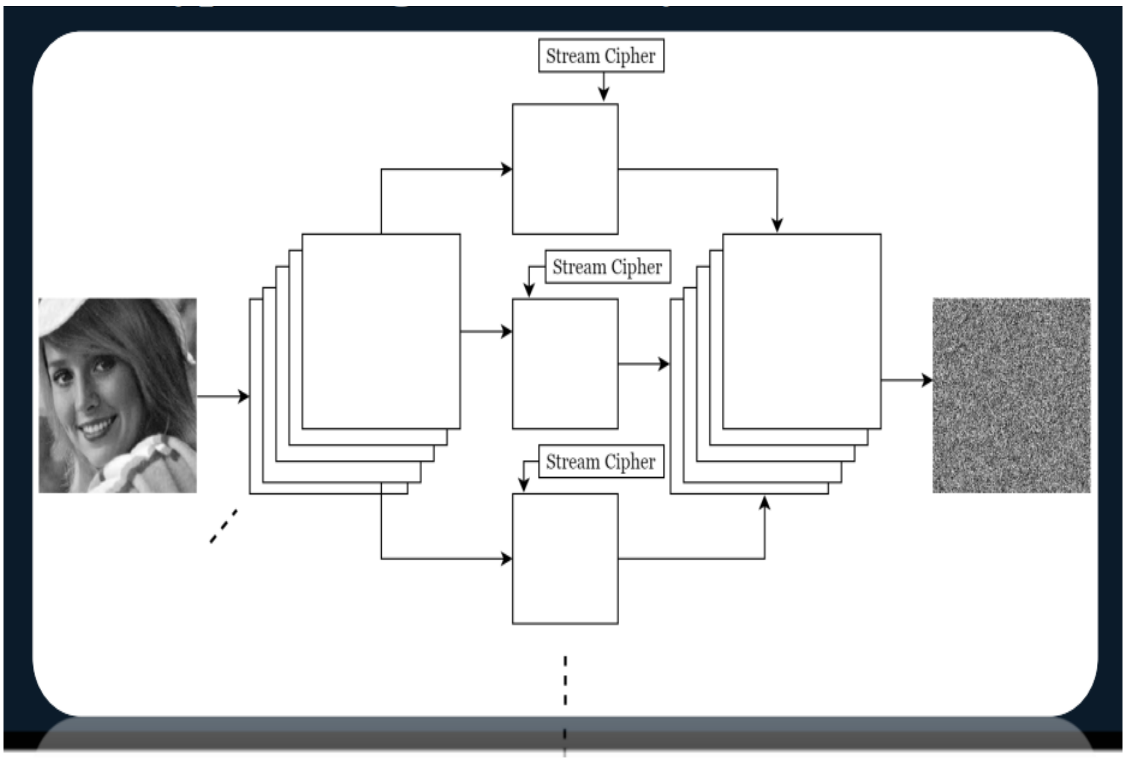
- The encrypted image was similarly written to a file in hexadecimal which was again processed in MATLAB to obtain the image in its proper form.

  $fwrite(file_id,"%x",value)

PROJECT DESIGN

Suppose we take the following picture to encrypt and decrypt.

22 bit public key used = 22’b11_0100_1110_0001_1001_0001



Each image pixel contains 8 bit of data. We have 256 x 256 = 65536 pixels.

1. The contents of the hexfile is read in the testbench and stored in an array of 8 bit registers ‘mem’.

2. Now for the first 65536 positive clock edges the first bit from the right of every member of ‘mem’ is stored in a register of 65535 bit size p1.

3. Next the public key is generated, that is by taking the first 18 bits of the chosen public key and appending the plane number of the current plane. Plane numbers go from 1 to 8

4. Now the design module is activated and the inputs are the public key, private key, p1 containing the first bit of every pixel, clock and trigger tr. tr=1 activates the design module.

5. Now in the design module, the LSFRs are initialized with zeros, the first 64 clock cycles are used feeding the private key and the next 22 are used feeding the generated public key.
6. Then 100 clock cycles with majority logic ignoring outputs and the next 65535 cycles generating the final cipher key for the plane.

7. Now p1 is fed to the design module in batches of 256 bits since passing all the 65535 bits together caused memory issues.

8. For every such input batch an answer of 256 bits is produced by xoring the input with the cipher key. This answer is then stored in the testbench in a register of 65535 bits.

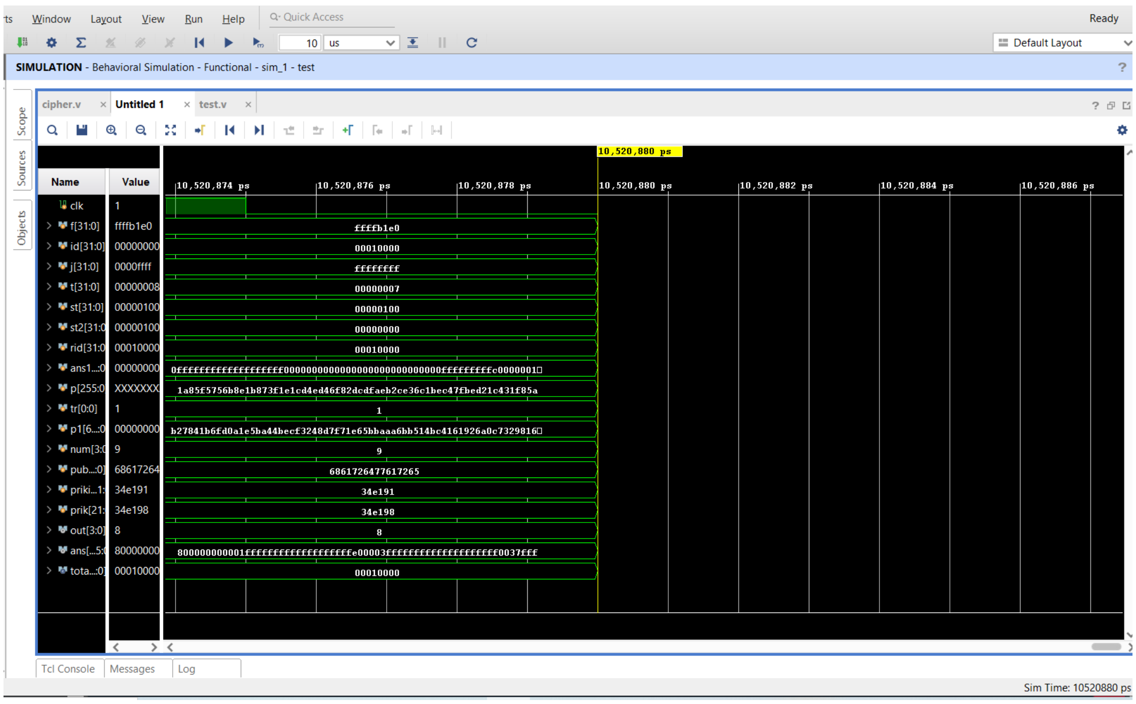
9. Once the entire plane has been processed, we have an encrypted plane and the bits are then reassembled and stored in the final output memory.

10. Then all the LSFRs are reinitialized, everything is reset and the next plane is processed with the public key ending with the new plane number.

11. Once all the planes are processed the output register contents are written to the output file.

The same testbench and design module can be reused to get back the original image by replacing the input file with output.txt and output file as decrypt.txt.

SIMULATION DIAGRAM:



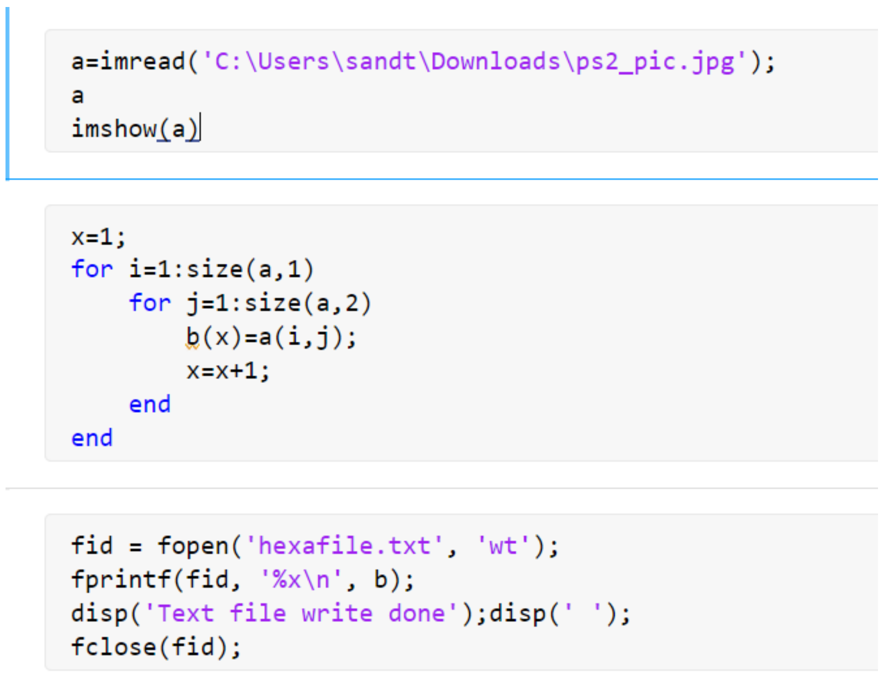
SIMULATION - Behavioral Simulation - Functional - sim_1 - test

Name	Value	
clk	1	
f[31:0]	ffffb1e0	ffffb1e0
id[31:0]	00000000	00010000
j[31:0]	0000ffff	ffffffff
t[31:0]	00000008	00000007
st[31:0]	00000100	00000100
st2[31:0	00000100	00000000
rid[31:0	00010000	00010000
ans1...:0	00000000	0ffffffffffffffffffff000000000000000000000000000000ffffffffc0000001
p[255:0]	XXXXXXX	1a85f5756b8e1b873f1e1cd4ed46f82dcdfaeb2ce36c1bec47fbed21c431f85a
tr[0:0]	1	1
p1[6...:0	00000000	b27841b6fd0a1e5ba44becf3248d7f71e65bbaaa6bb514bc4161926a0c7329816
num[3:0	9	9
pub...:0]	68617264	6861726477617265
priki...1:	34e191	34e191
prik[21:	34e198	34e198
out[3:0]	8	8
ans[...5:0	80000000	800000000001fffffffffffffffffffe00003ffffffffffffffffffff0037fff
tota...:0]	00010000	00010000

Sim Time: 10520880 ps

CODE SNIPPETS

MATLAB Code to Generate Hexadecimal File:

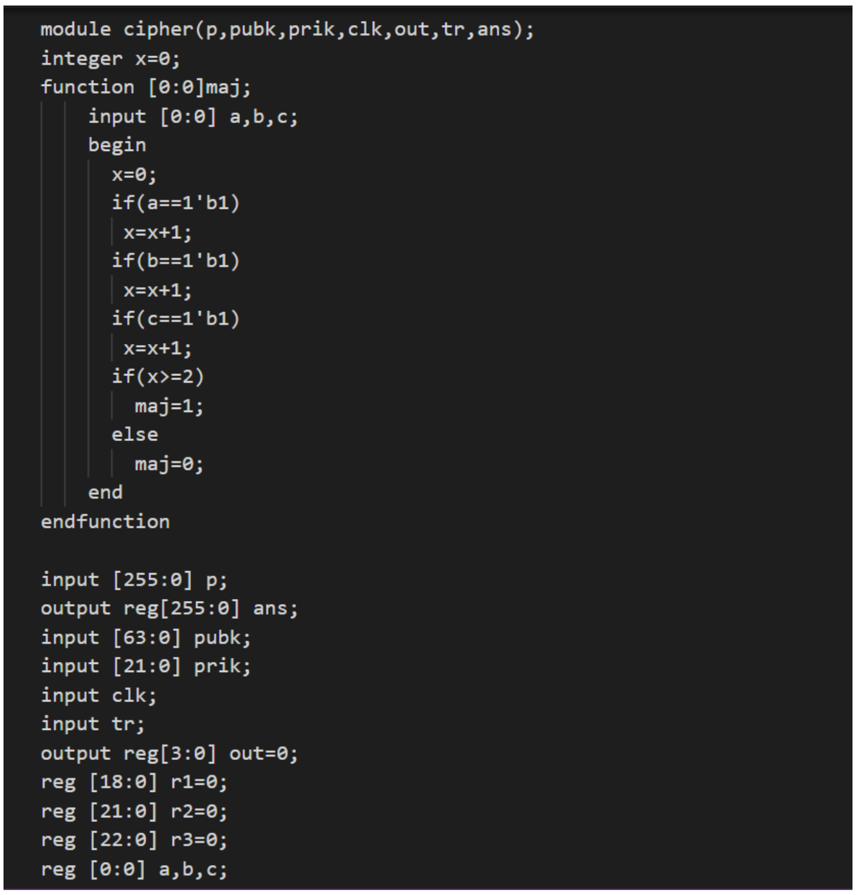


```
a=imread('C:\Users\sandt\Downloads\ps2_pic.jpg');
a
imshow(a)
```

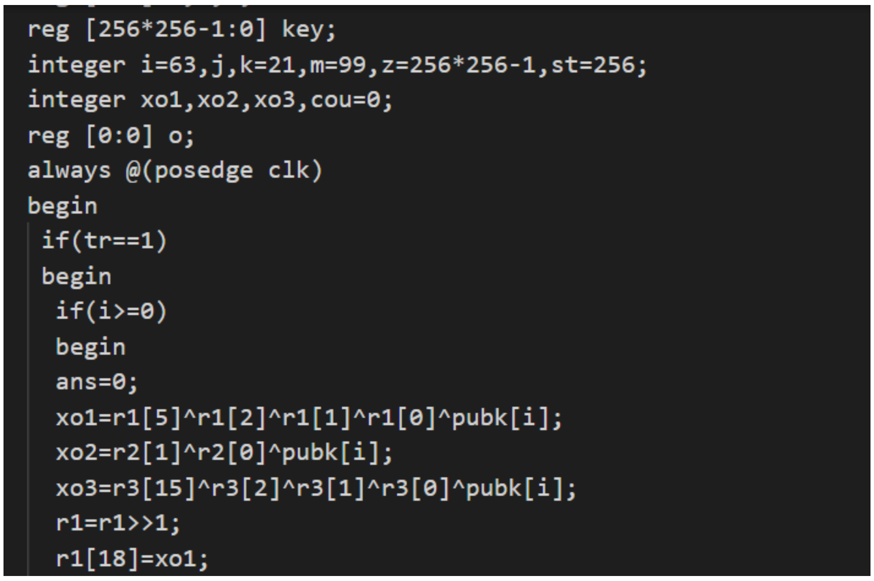
```
x=1;
for i=1:size(a,1)
    for j=1:size(a,2)
        b(x)=a(i,j);
        x=x+1;
    end
end
```

```
fid = fopen('hexafile.txt', 'wt');
fprintf(fid, '%x\n', b);
disp('Text file write done');disp(' ');
fclose(fid);
```

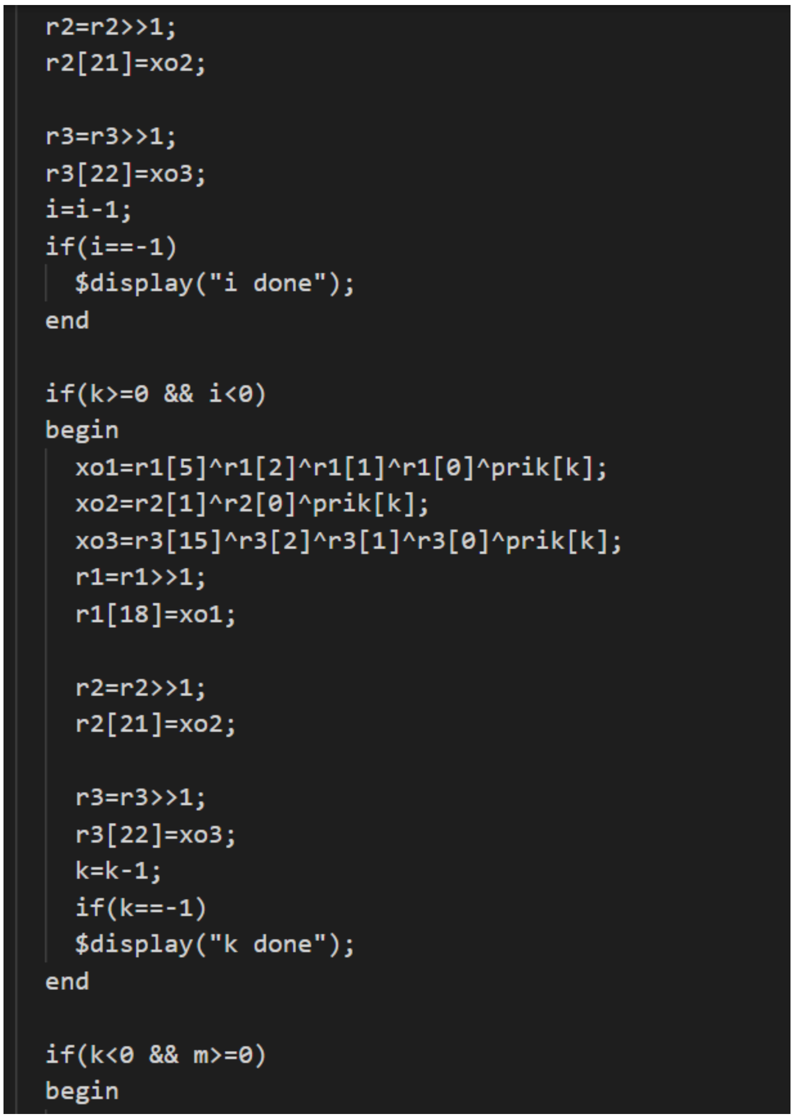
Following is the code of design module cipher :



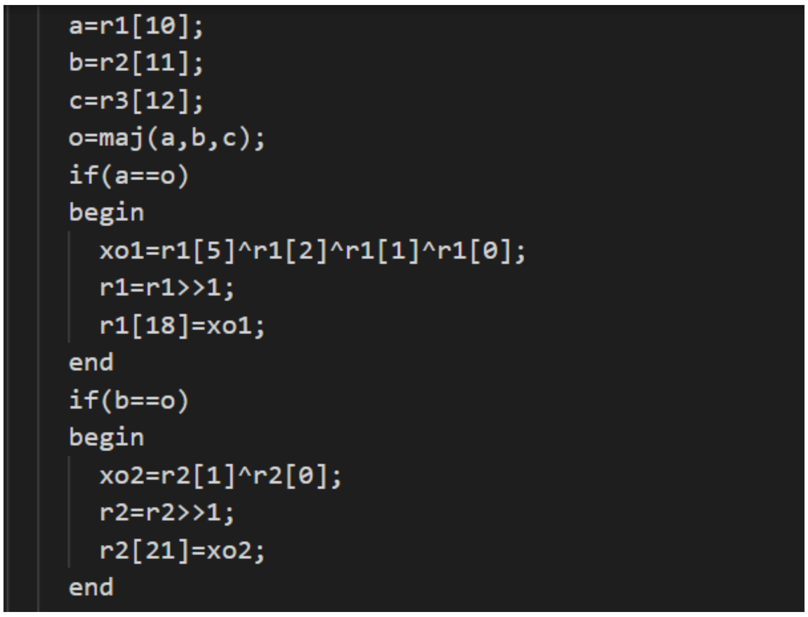
```
module cipher(p,pubk,prik,clk,out,tr,ans);
integer x=0;
function [0:0]maj;
    input [0:0] a,b,c;
    begin
      x=0;
      if(a==1'b1)
       x=x+1;
      if(b==1'b1)
       x=x+1;
      if(c==1'b1)
       x=x+1;
      if(x>=2)
        maj=1;
      else
        maj=0;
    end
endfunction

input [255:0] p;
output reg[255:0] ans;
input [63:0] pubk;
input [21:0] prik;
input clk;
input tr;
output reg[3:0] out=0;
reg [18:0] r1=0;
reg [21:0] r2=0;
reg [22:0] r3=0;
reg [0:0] a,b,c;
```



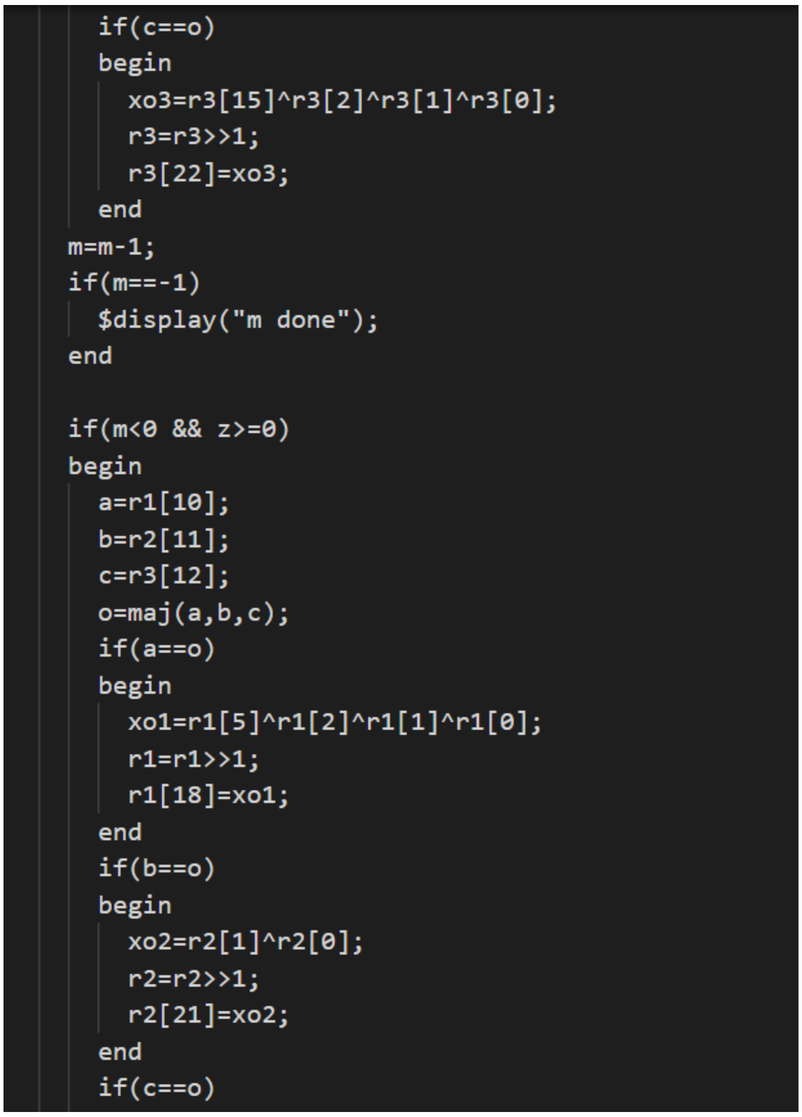
```
reg [256*256-1:0] key;
integer i=63,j,k=21,m=99,z=256*256-1,st=256;
integer xo1,xo2,xo3,cou=0;
reg [0:0] o;
always @(posedge clk)
begin
 if(tr==1)
 begin
  if(i>=0)
  begin
  ans=0;
  xo1=r1[5]^r1[2]^r1[1]^r1[0]^pubk[i];
  xo2=r2[1]^r2[0]^pubk[i];
  xo3=r3[15]^r3[2]^r3[1]^r3[0]^pubk[i];
  r1=r1>>1;
  r1[18]=xo1;
```



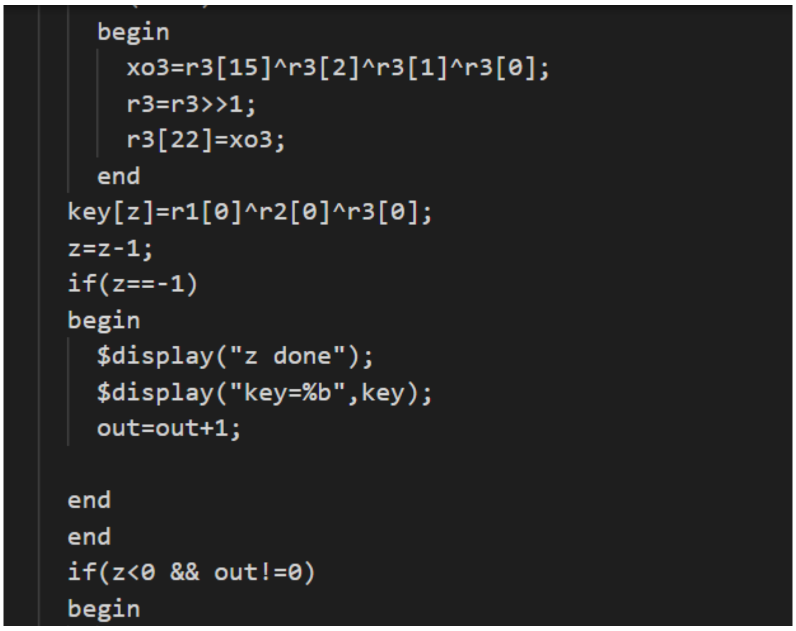
```
r2=r2>>1;
r2[21]=xo2;

r3=r3>>1;
r3[22]=xo3;
i=i-1;
if(i==-1)
  $display("i done");
end

if(k>=0 && i<0)
begin
  xo1=r1[5]^r1[2]^r1[1]^r1[0]^prik[k];
  xo2=r2[1]^r2[0]^prik[k];
  xo3=r3[15]^r3[2]^r3[1]^r3[0]^prik[k];
  r1=r1>>1;
  r1[18]=xo1;

  r2=r2>>1;
  r2[21]=xo2;

  r3=r3>>1;
  r3[22]=xo3;
  k=k-1;
  if(k==-1)
  $display("k done");
end

if(k<0 && m>=0)
begin
```



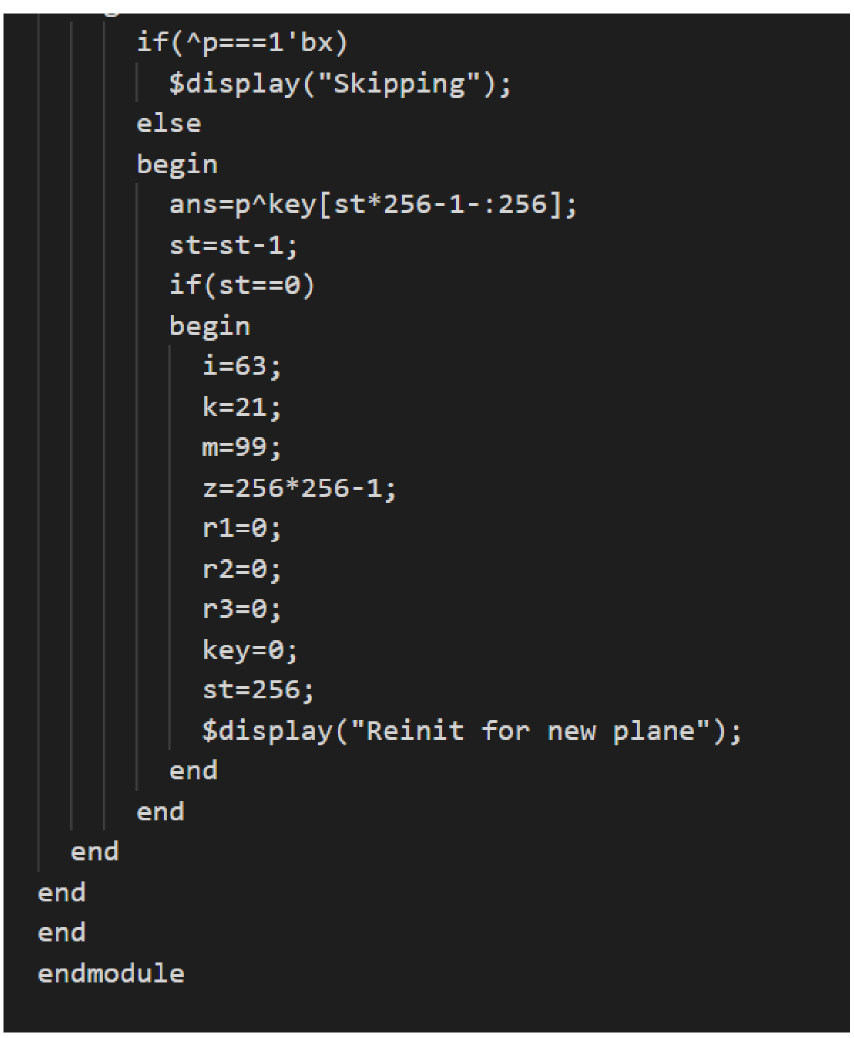
```
a=r1[10];
b=r2[11];
c=r3[12];
o=maj(a,b,c);
if(a==o)
begin
  xo1=r1[5]^r1[2]^r1[1]^r1[0];
  r1=r1>>1;
  r1[18]=xo1;
end
if(b==o)
begin
  xo2=r2[1]^r2[0];
  r2=r2>>1;
  r2[21]=xo2;
end
```



```
  if(c==o)
  begin
    xo3=r3[15]^r3[2]^r3[1]^r3[0];
    r3=r3>>1;
    r3[22]=xo3;
  end
m=m-1;
if(m==-1)
  $display("m done");
end

if(m<0 && z>=0)
begin
  a=r1[10];
  b=r2[11];
  c=r3[12];
  o=maj(a,b,c);
  if(a==o)
  begin
    xo1=r1[5]^r1[2]^r1[1]^r1[0];
    r1=r1>>1;
    r1[18]=xo1;
  end
  if(b==o)
  begin
    xo2=r2[1]^r2[0];
    r2=r2>>1;
    r2[21]=xo2;
  end
  if(c==o)
```

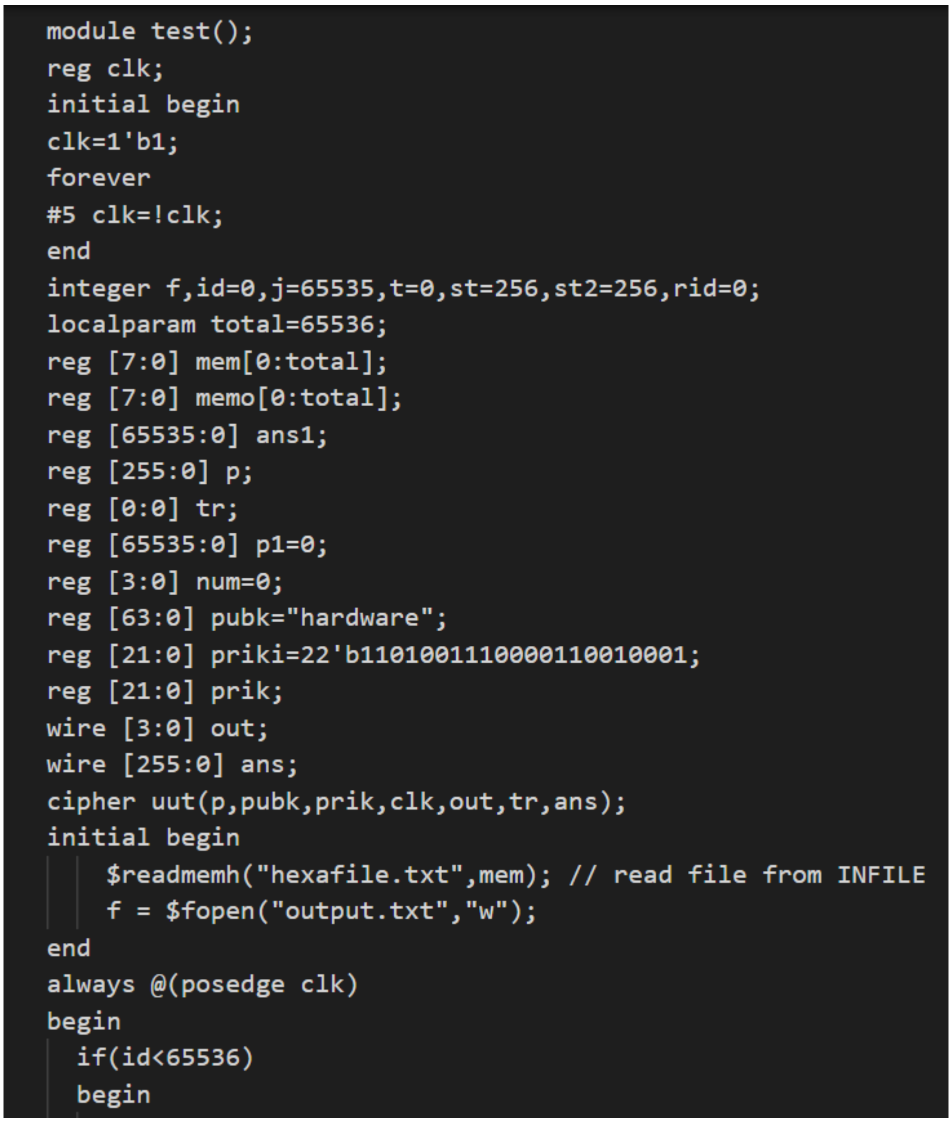


```
  begin
    xo3=r3[15]^r3[2]^r3[1]^r3[0];
    r3=r3>>1;
    r3[22]=xo3;
  end
key[z]=r1[0]^r2[0]^r3[0];
z=z-1;
if(z==-1)
begin
  $display("z done");
  $display("key=%b",key);
  out=out+1;

end
end
if(z<0 && out!=0)
begin
```

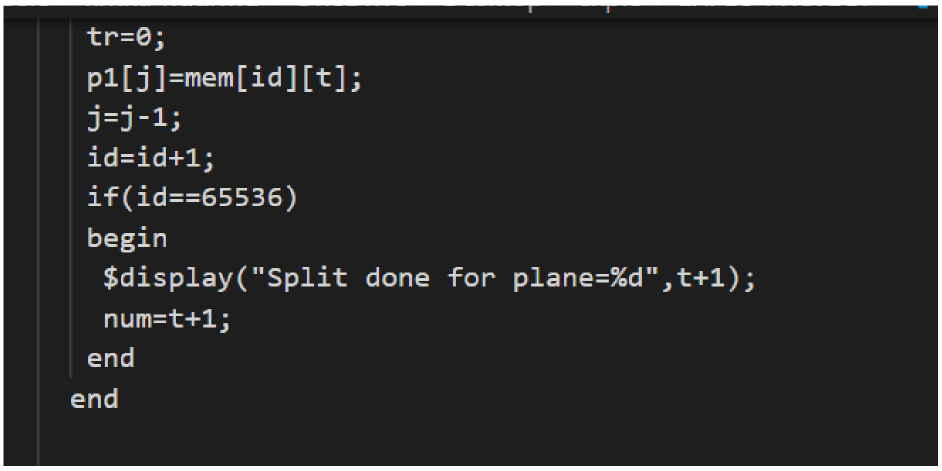


```
        if(^p===1'bx)
          $display("Skipping");
        else
        begin
          ans=p^key[st*256-1-:256];
          st=st-1;
          if(st==0)
          begin
            i=63;
            k=21;
            m=99;
            z=256*256-1;
            r1=0;
            r2=0;
            r3=0;
            key=0;
            st=256;
            $display("Reinit for new plane");
          end
        end
    end
  end
  end
  endmodule
```

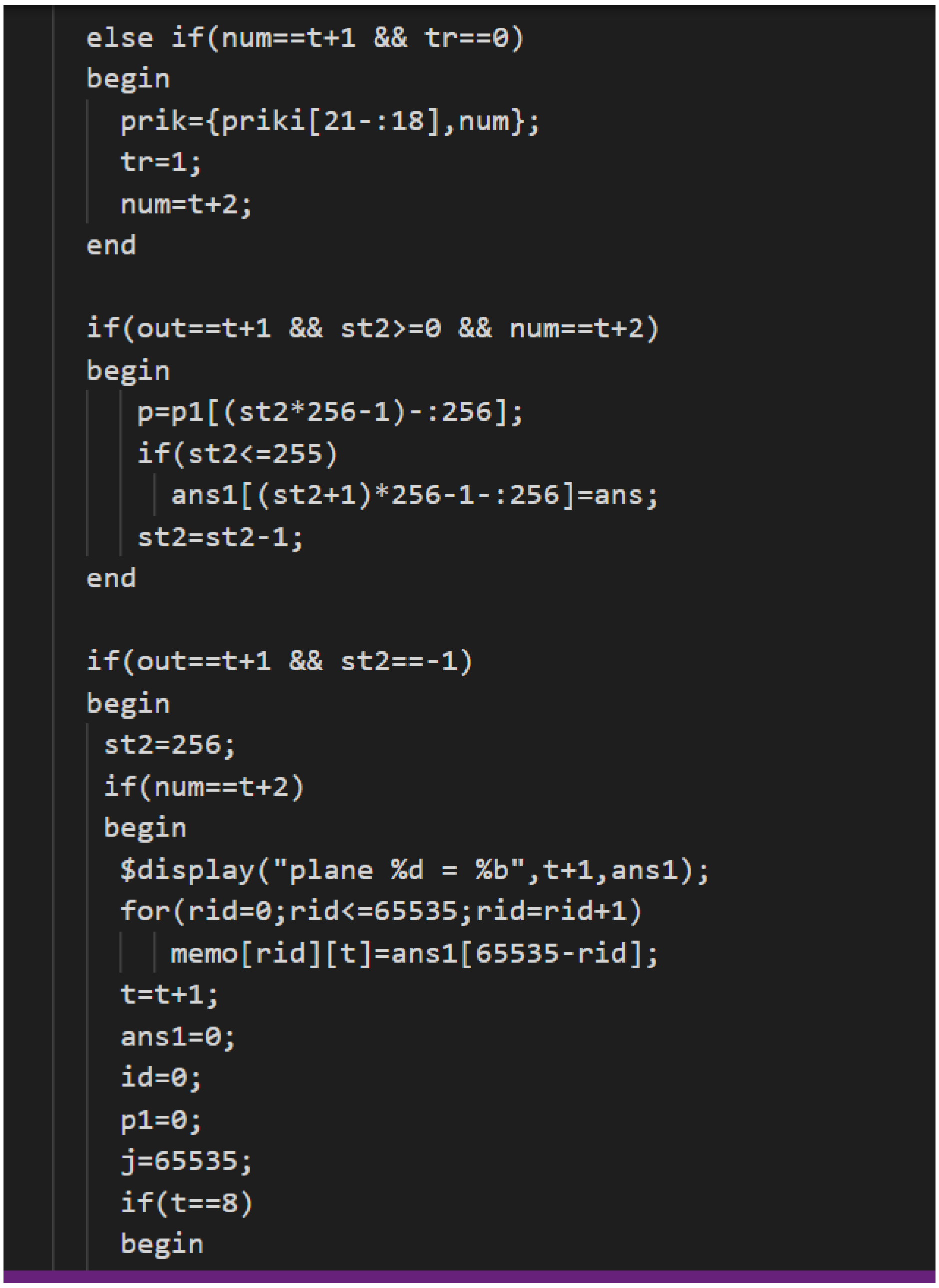
TESTBENCH Encrypt.v:



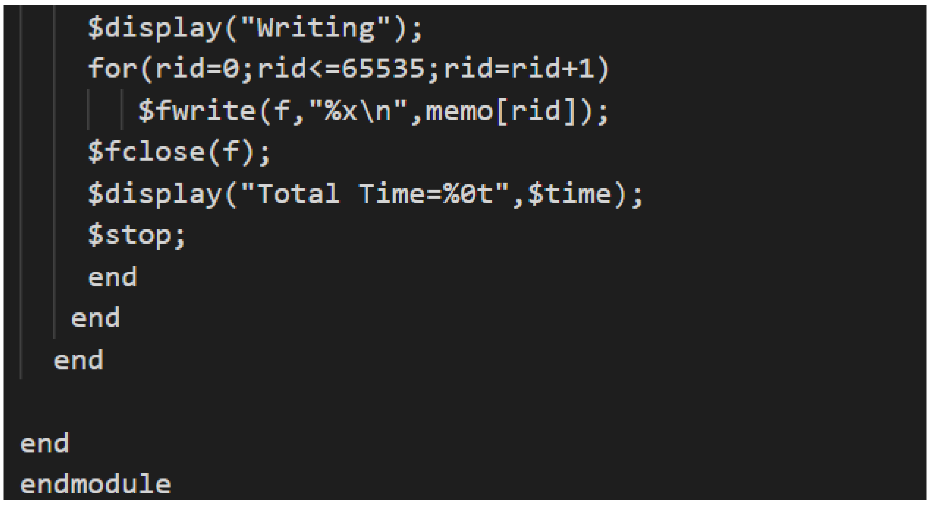
```
module test();
reg clk;
initial begin
clk=1'b1;
forever
#5 clk=!clk;
end
integer f,id=0,j=65535,t=0,st=256,st2=256,rid=0;
localparam total=65536;
reg [7:0] mem[0:total];
reg [7:0] memo[0:total];
reg [65535:0] ans1;
reg [255:0] p;
reg [0:0] tr;
reg [65535:0] p1=0;
reg [3:0] num=0;
reg [63:0] pubk="hardware";
reg [21:0] priki=22'b1101001110000110010001;
reg [21:0] prik;
wire [3:0] out;
wire [255:0] ans;
cipher uut(p,pubk,prik,clk,out,tr,ans);
initial begin
    $readmemh("hexafile.txt",mem); // read file from INFILE
    f = $fopen("output.txt","w");
end
always @(posedge clk)
begin
  if(id<65536)
  begin
```



```
    tr=0;
    p1[j]=mem[id][t];
    j=j-1;
    id=id+1;
    if(id==65536)
    begin
     $display("Split done for plane=%d",t+1);
     num=t+1;
    end
  end
```

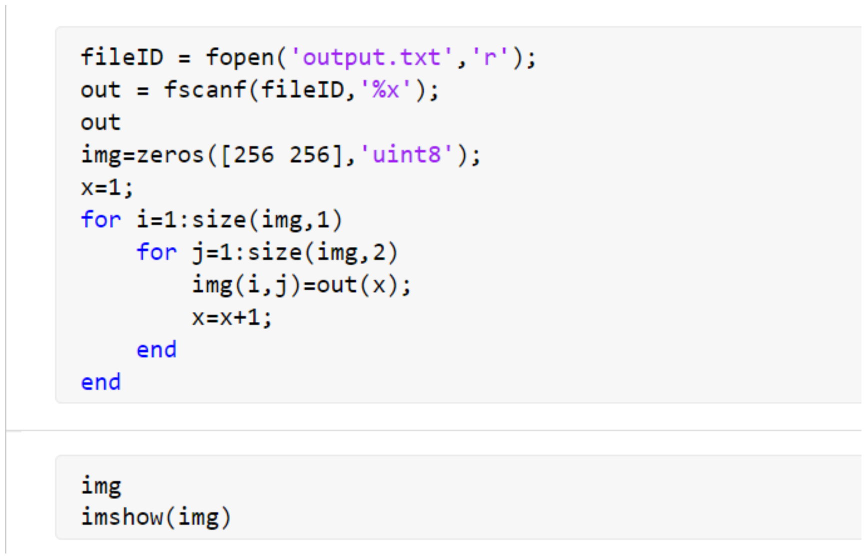


```
    else if(num==t+1 && tr==0)
    begin
      prik={priki[21-:18],num};
      tr=1;
      num=t+2;
    end

    if(out==t+1 && st2>=0 && num==t+2)
    begin
       p=p1[(st2*256-1)-:256];
       if(st2<=255)
         ans1[(st2+1)*256-1-:256]=ans;
       st2=st2-1;
    end

    if(out==t+1 && st2==-1)
    begin
     st2=256;
     if(num==t+2)
     begin
      $display("plane %d = %b",t+1,ans1);
      for(rid=0;rid<=65535;rid=rid+1)
         memo[rid][t]=ans1[65535-rid];
      t=t+1;
      ans1=0;
      id=0;
      p1=0;
      j=65535;
      if(t==8)
      begin
```



```
    $display("Writing");
    for(rid=0;rid<=65535;rid=rid+1)
       $fwrite(f,"%x\n",memo[rid]);
    $fclose(f);
    $display("Total Time=%0t",$time);
    $stop;
    end
   end
  end

end
endmodule
```

MATLAB Code to get back Image from Hexadecimal File:



```matlab
fileID = fopen('output.txt','r');
out = fscanf(fileID,'%x');
out
img=zeros([256 256],'uint8');
x=1;
for i=1:size(img,1)
    for j=1:size(img,2)
        img(i,j)=out(x);
        x=x+1;
    end
end
```

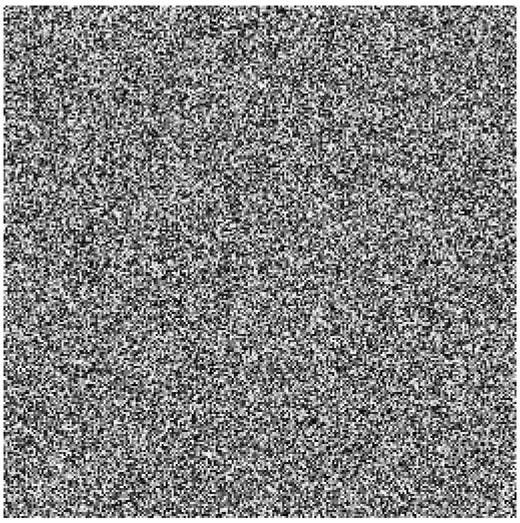
```matlab
img
imshow(img)
```

RESULTS

We were successfully able to implement our design for encryption and decryption of image.

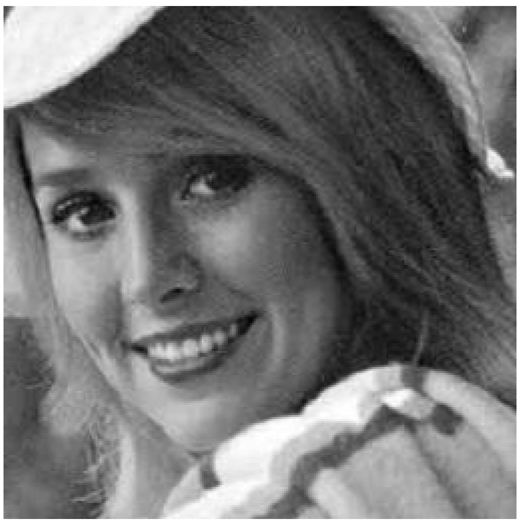
Following was the input image:

Following was the encrypted image:



(not readable)

Following was the Decrypted image:



CONCLUSION

In this project, we dealT with the concepts of security of digital data communication across the network. This project is designed by using the cryptography features factors for better performance. We Designed a design using A5/1 Algorithm for over-the-air communication privacy in GSM cellular telephone standard.

REFERENCES

https://www.cryptographynotes.com/2019/02/symmetric-stream-a5by1-algorithm-linear-feedback-shift-register.html

https://en.wikipedia.org/wiki/A5/1

https://www.thalesgroup.com/en/markets/digital-identity-and-security/technology/ota