DESIGN AND IMPLEMENTATION OF ENCRYPTION AND DECRYPTION USING RIVEST CIPHER (RC4)

NAME	RONANKI BHANU SAMPATH	PENMETSA VENKATA PAVAN KUMAR VARMA	I VARSHITA	KATARI YUVA SAI KIRAN	NATHANIEL VINNY ALEX
REG. NO	200932046	200906228	200907232	200932016	200932120

Introduction:

RC4 is a stream cipher and variable-length key algorithm. This algorithm encrypts one byte at a time. User Input refers to Plain text and Common Key (k) shared with the known parties where in between the data communication took place. Firstly, S-Array is taken as S[i]=i where i varies from 0 to 31, and "key length" is defined as the number of bytes in the key. After that, KSA would take place, which we discussed later in this report, and then PRGA will be discussed later. After getting a KEY STREAM, we do Encryption by performing an Exclusive-OR operation between Plain Text and Key stream which produces the Cipher. Finally, we can perform Modulo addition/Exclusive-OR operation between Ciphertext and Key stream to get Plain text.

Theory:

● INITIALIZATION:

The initialization of S-Array.

● KEY SCHEDULING ALGORITHM(KSA) - (Used to perform different permutations on the S-array using T[i] values to at least move each of the elements of the S-array.)

Initializing T-Array and performing Key Scheduling Algorithm.

(The number of iterations we perform is equal to the length of the state vector we choose)

● PSEUDO-RANDOM GENERATION ALGORITHM (KEY STREAM GENERATION)

(Used to generate the Key Stream for Encryption and Decryption)

1. The S-Array obtained after KSA will be used,
2. Encryption and Cipher Generation,
3. Decryption.

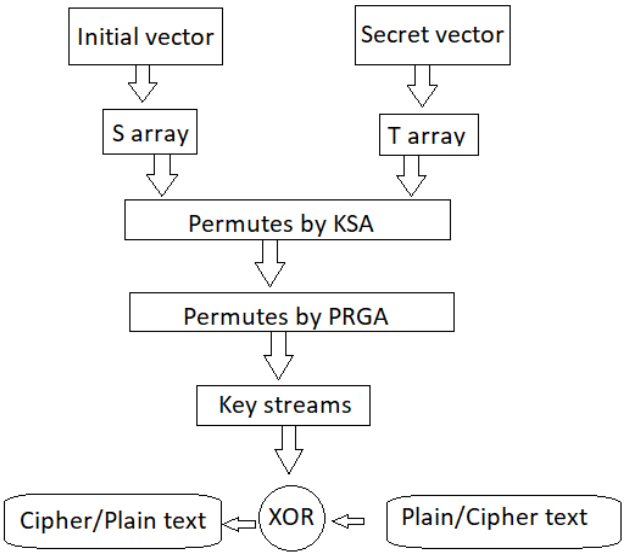

Fig.1 flowchart representing the logic used for encryption and decryption.

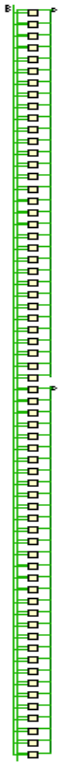

Fig.2 The RTL of the circuit used.

Result:

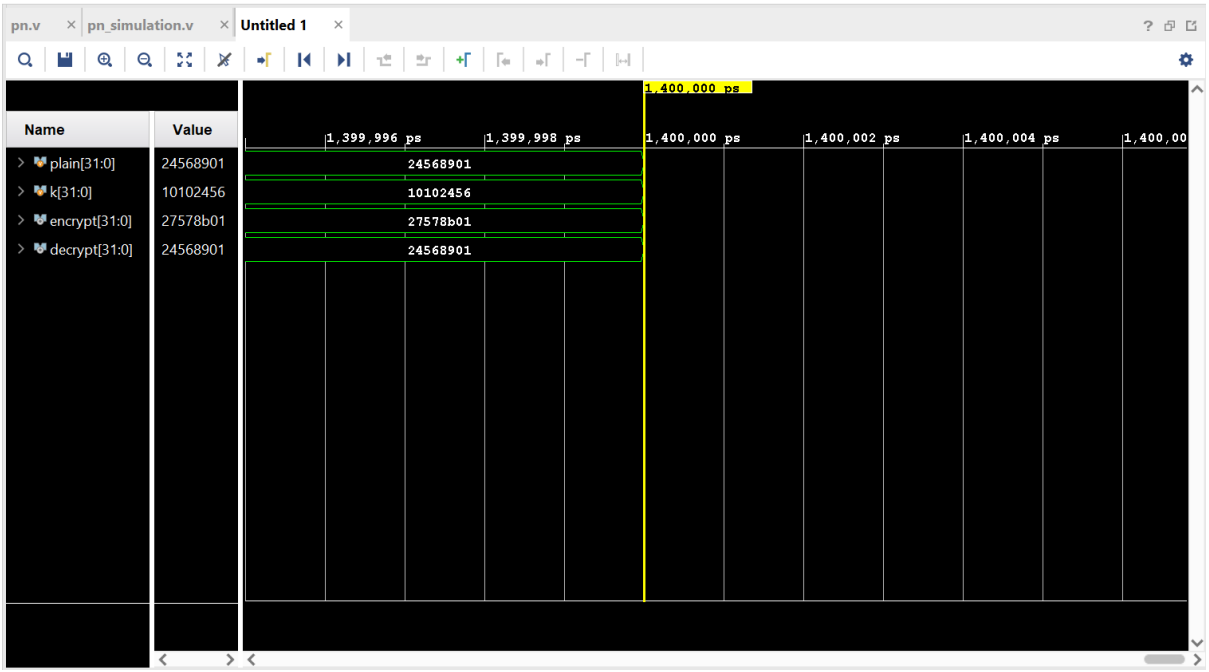

Fig.3 shows us the simulation result.

Contribution:

1. Pavan worked with the RTL design. He has a good grasp and idea of RTL circuitry and has moreover worked with the testability of the circuit.
2. Varshita and Sampath had dealt with the Verilog coding part as they have a good command of it moreover, they had explored the world of encryption and decryption and came up with this idea.
3. Sai Kiran had worked with the Verilog part along with a good amount of time spared for the research in the theoretical, practical and feasibility of the design over others.
4. Nathaniel has taken part in all aspects of the project and has specially given great attention to the design feasibility, availability, and commerciality of the product, etc.

Conclusion:

The project was performed in Vivado and the results obtained were up to the requirement Specification and the proposed design follows a simple and structured approach to the problem statement.

Reference:

1. https://www.geeksforgeeks.org/what-is-rc4-encryption/
2. https://www.academia.edu/5370660/Hardware_Implementation_of_Modified_RC4_Stream_Cipher_Using_FPGA

Appendix:

Assumptions while writing the code,

s - S-ARRAY (STATE VECTOR)

plain - PLAIN TEXT

k – KEY VALUES (secret code that the sender and receiver have.)

klen - KEY LENGTH

t - TEMPORARY VECTOR

key - KEY STREAM GENERATED

(The user has to provide “plain text” and “k”)

Source Code,

```
module pn(
    input [31:0] plain,
    input [31:0] k,
    output reg [31:0] encrypt,
    output reg [31:0] decrypt
);
    integer i = 0, j = 0, klen = 32;
    integer n = 0, l = 0, m = 0, x = 0;
    reg [31:0] s = 0;
    reg [31:0] t = 0;
    reg [7:0] trail = 0;
    reg [31:0] key = 0;
```

/*consider s[31:0] array and initialize it with a trail[7:0] in such a way that s[31:0] gets 4 copies of t[7:0] sequentially into it*/

```
    initial
    begin
        for (i = 0; i < 5; i = i + 1)
        begin
            trail = i;
            for (j = 0; j < 8; j = j + 1)
            begin
```

```
            s[j + 8 * n] = trail[j];                    //Initializing s-array
          end
          n = n + 1;
       end

for (i = 0; i < 32; i = i + 1)
begin
   t[i] = k[i];                                         //Initializing t-array
end

i = 0;
n = 0;
while (i < 4)                                           // KSA starts
begin
  for (l = 0; l < 8; l = l + 1)
  begin
    trail[l] = trail[l] ^ s[l + 8 * n] ^ t[l + 8 * n];      // Initializing the values of trail array
  end
j = trail;                                              // Initializing "j" with "trail" array
j = j % 4;
   for (l = 0; l < 8; l = l + 1)
   begin
     trail[l] = s[l + 8 * n];
     s[l + 8 * n] = s[l + 8 * j];                       // S[n] and S[j] were swapped
     s[l + 8 * j] = trail[l];
   end
n = n + 1;
i = i + 1;
end                                                     // KSA ends

i = 0;
j = 0;
```

```
m = 0;
n = 0;

while (x < 4)                                   // PRGA starts
begin
i = (i + 1) % 4;                                // "i" must be initialised
    for (l = 0; l < 8; l = l + 1)
    begin  // "i" is added to itself to provide running value to the "j" and swap operation.
       trail[l] = (trail[l] ^ s[8 * i + l]);
    end
j = trail;                                      // "j" must be initialised
j = j % 4;
trail = 0;
  for (l = 0; l < 8; l = l + 1)
  begin
    trail[l] = s[l + 8 * i];
    s[l + 8 * i] = s[l + 8 * j];                // s[i] and s[j] are swapped
    s[l + 8 * j] = trail[l];
  end
   for (l = 0; l < 8; l = l + 1)
   begin
      trail[l] = s[l + 8 * i] ^ s[l + 8 * j];   // trail is obtained using s[i] and s[j]
   end
m = trail;
m = m % 4;
    for (l = 0; l < 8; l =l + 1)
    begin
      key[l + 8 * n] = s[l + 8 * m];            //Generation of keystream
    end
n = n + 1;
x = x + 1;
end                                             // PRGA ends
```

```
 for (i = 0; i < 32; i = i + 1)
    begin
       encrypt[i] = plain[i] ^ key[i];                  // For Encryption
    end

for (i = 0; i < 32; i = i + 1)
    begin
       decrypt[i] = encrypt[i] ^ key[i];               //For Decryption
    end
end
endmodule
```

Test bench,

```
module pn_simulation(
    );
    reg [31:0] plain=32'h24568901;
    reg [31:0] k=32'h10102456;
    wire [31:0] encrypt;
    wire [31:0] decrypt;
    pn tf(plain,k,encrypt,decrypt);
    initial
    begin
    #10
    plain=32'h24568901;
    k=32'h10102456;
    end
endmodule
```

THANK YOU