Benha University

Faculty Of Engineering

Department of computers and communications

Data Encryption Standard Algorithm

By

Nagham Nasser Kamal

Noran Nabil Ahmed

Supervisors

Dr./ Ayman Mostafa

Eng./ Eman

Department of computers and communications
Faculty Of Engineering Benha University

Table of Contents

Table of Figures

1. Software Tools

1.1 Keil IDE

Keil IDE is suitable environment for coding 8051 MCU. We have a free version that has limitation on the code size to be compiled.

1.2 SDCC compiler

SDCC compiler is a suitable with no limitations on the code size for 8051. We use the CMD to use the SDCC compiler with command shown in figure 1 to compile the source file and another command to generate the hex file as sown in figure 2.

```
E:\Communication4\S1\Encryption\DES_final\DES\Encryption>sdcc main.c
```

Figure 2 shows compiling command

```
E:\Communication4\S1\Encryption\DES_final\DES\Encryption>packihx main.ihx > main.hex
packihx: read 220 lines, wrote 425: OK.
```

Figure 1 shows generating hex file command

1.3 STC89C5x Programmer

Programming is done using Arduino uno as a POV with connections shown in figure 3 and STC89C5x programmer shown in figure 4 to burn the hex file on the 8051 microcontroller. We use the crystal = 12 MHz with 2 ceramic capacitors of value 33 pf.

```
pinMode(A2, OUTPUT); // GND output to POV
pinMode(A3, OUTPUT); // connected to RxD/P3.0 on POV
pinMode(A4, INPUT_PULLUP);  // connected to TxD/P3.1 on POV
pinMode(A5, OUTPUT); // outputs +5V to power POV
```

Figure 3 shows 4 pins connected from Arduino to STC MIC

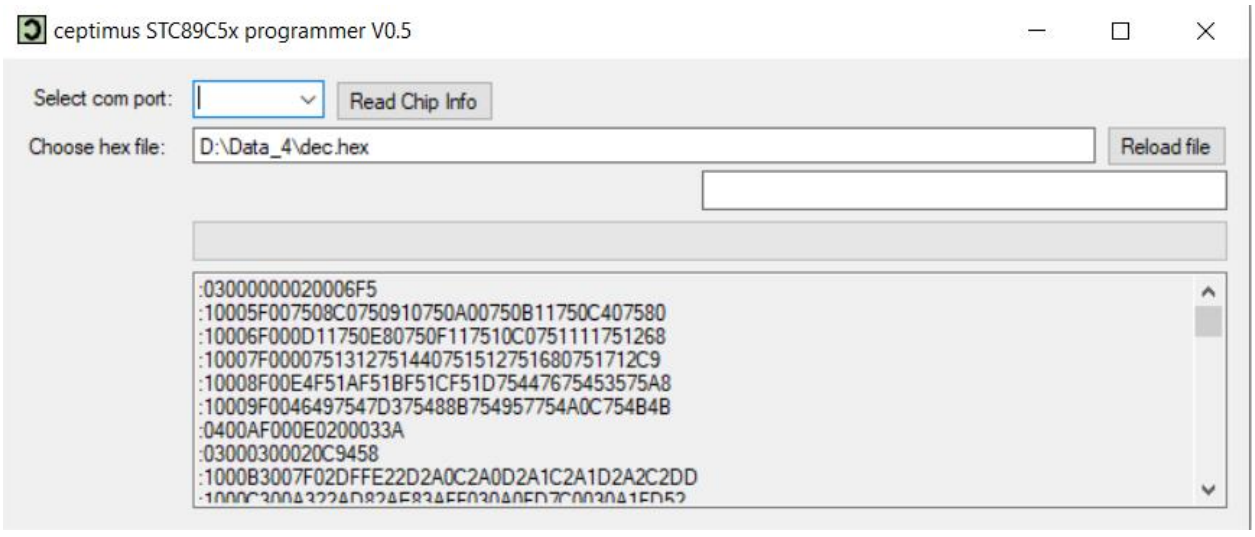

Figure 4 shows STC89C5x programmer.

2. Hardware Tools

2.1 Arduino UNO

Arduino UNO as shown in figure 6 is used as a bridge between the pc and the microcontroller to help the programmer to burn the hex file on the 8051.

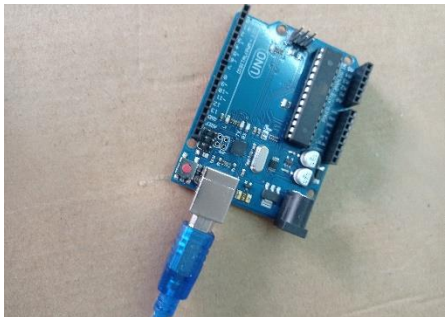

Figure 6 shows Arduino uno

2.2 STC MCU

We use two 8051 microcontrollers to act as Alice and Bob, or any two users need to communicate with each other. We use crystal = 11.0592 MHz It is suitable to use the UART connection with baud rate 9600 b/sec, Tx and Rx, to send and receive.

Figure 7 shows STC MCU

To complete the oscillator circuit for the 8051, it is required to use two capacitors which are 33PF. The connections of this circuit will be illustrated below.

2.3 Two breadboards

We use two different breadboards shown in figure 8 to isolate Alice and Bob and act as two different environments for each.

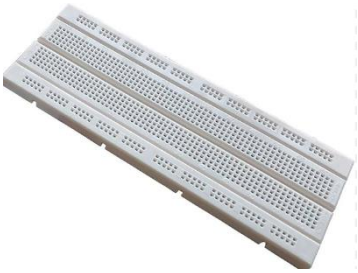

Figure 8 shows breadboard

2.4 Two TTL connectors

The TTL shown in figure 9 is used to send the Key and the plain text from one pc to one microcenter to start the process of encryption. And another TTL is used to send the Key from one pc to one microcenter to start the process of decryption. Sending the Key two times in the encryption and decryption is made because the key cannot be transferred in an unsecure channel. It must be transmitted between Alice and Bob through secure channel and each one use it in the beginning of the encryption process. The TTL is connected to the microcontroller using the UART protocol. It uses the Tx and Rx pins from the MCU.

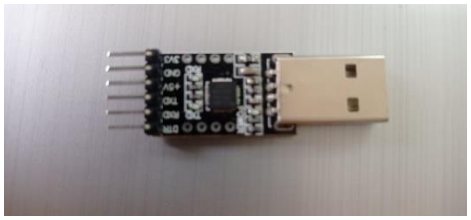

Figure 9 shows TTL

3. Burning process

Firstly, the Arduino is programmed as POV programmer, then it is connected to the STC microcontroller and programmed as shown in figure 10. For completing the programming requirements, we use two 33PF ceramic capacitors and 12 MHz crystal. One leg of each capacitor is connected to the ground. Another leg of each capacitor is connected to different leg of the crystal. The programmer reads the chip information and identify the type of the microcontroller and the frequency of the crystal by clicking the chip info button. The upload the hex file and press Reload file button.

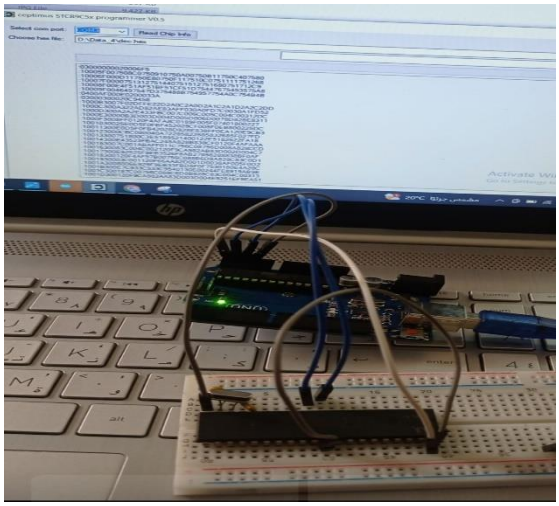

Figure 10 shows the burning process.

4. Circuit Diagram

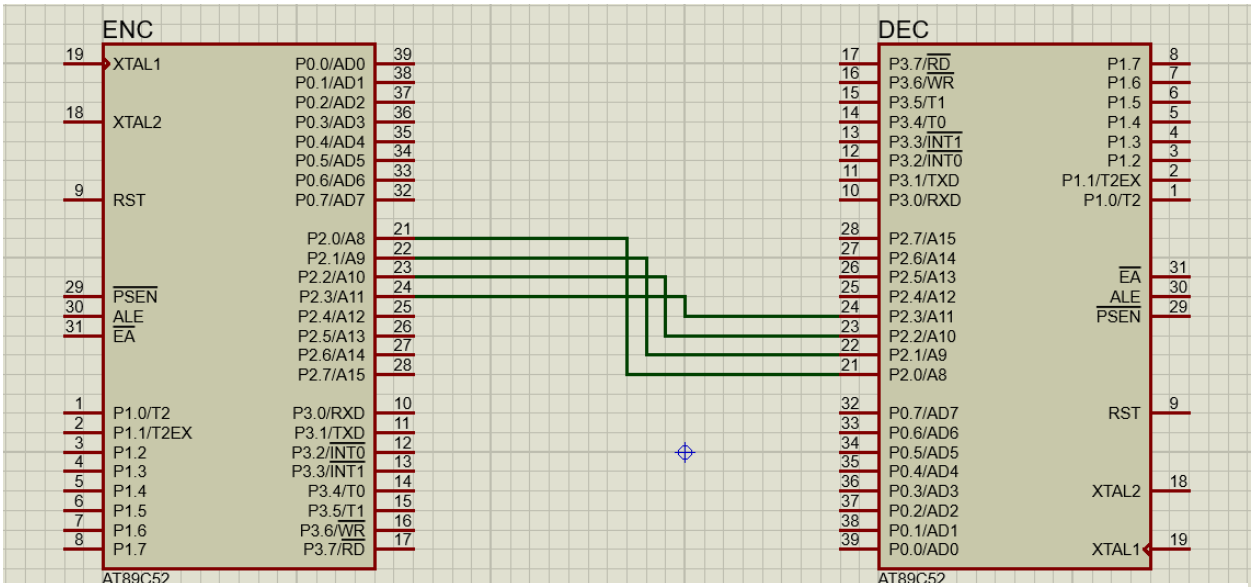

Figure 11: Simulation.

5. Serial connections

- First Microcontroller which is responsible for encryption process is connected to Alice PC using serial connection with baud rate 9600 bits/sec.
- Second Microcontroller which is responsible for decryption process is connected to Bob PC using serial connection with baud rate 9600 bits/sec.
- Figure 12 shows 89c52 MIC serial initialization.

```
SCON=0x50;
TMOD=0x20;
TH1=-3;
TL1=-3;
TR1=1;
```

Figure 12: Shows serial initialization code.

6. Encryption process

- Figure 13 shows the DES whole block diagram.

3.1 Initial permutation

- 64 bits of plain text are firstly permutated depending on IP table.
- Figure 14 shows permutation function applied on the plain text, and its call is shown in figure 15.

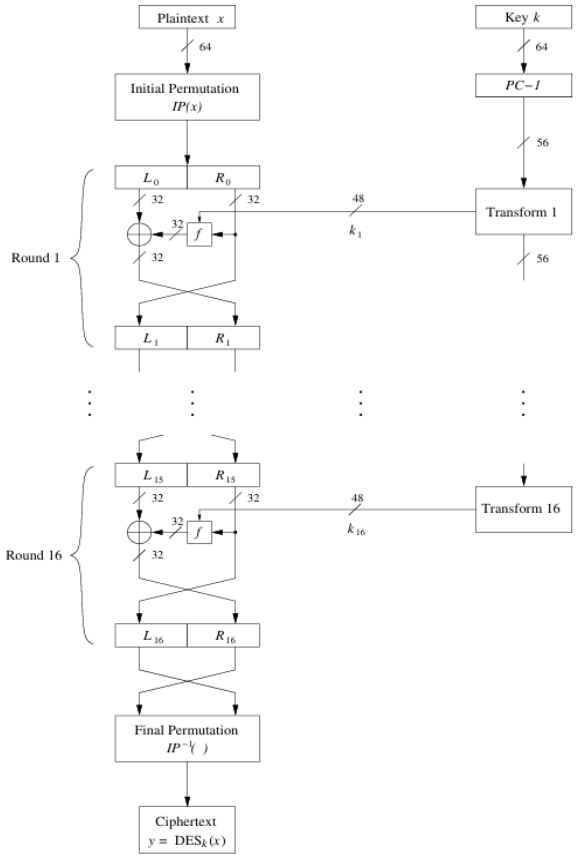

Figure 13 shows DES block diagram.

```
/* IP    FP    PC-1    PC-2*/
static void Permutation ( DATA *plain_text,DATA *permuted_text,uint8 type,uint8 *table){
    for (i = 0 ;i < type ; i++){
        //element_no = (table[i] -1) / 8;
        //bit_no = (table[i] -1) % 8;
        temp = GET_BIT_ARRAY(plain_text->Array[((table[i] -1) / 8)],((table[i] -1) % 8));
        //element_no = i / 8;
        //bit_no = i % 8;
        WRITE_BIT_ARRAY(permuted_text->Array[(i/8)],(i%8),temp);
    }
}
```

Figure 14 shows Permutation function.

```
// 1 - Initial Permutation
Permutation ( &final_plain_text, &round_in, Basic_Permutation, ip);
```

Figure 15 shows permutation function call for initial permutation.

3.2 G-Function

In each round 64 bits are splitted as left and right parts of 32 bits to be processed.

- **Expansion**

  32 bits of right part are expanded to 48 bits using E-table as shown in figure 16.

```
// 1- Expansion
for (i = 0 ;i <= 47 ; i++){ //arr = {f0 ,aa, f0, aa} >>> aa f0 aa f0
    temp =(uint8)(((Temp<<(e[i]-1)) & 0x80000000)>>31) ;
    WRITE_BIT_ARRAY(out_g->Array[(i/8)],(i%8),temp);
}
```

Figure 16 shows expansion function implementation.

- **XOR**

  Expansion function output is xored with round key as shown in figure 17.

```
//2-XOR with subkey
for (i = 0; i<7 ;i++){
    out_g->Array[i] ^=sub_key->Array[i];
}
```

Figure 17 shows XOR process.

- **S-BOX**
  XOR 48 bits result are inputs for eight s-boxes, each box outputs 4 bits so the total output is 32 bits as shown in figure 18.

- **Permutation**
  32 bits are permutated to generate G-function output of 32 bits as shown in figure 19.

```
//3- S-Boxes
uint8 in_s_box, out_s_box;
uint8 raw_s = 0, column_s = 0;
uint32 S_boxes_out = 0;
for (j = 0; j < 8; j++){
    in_s_box  = 0;
    out_s_box = 0;
    raw_s     = 0;
    column_s  = 0;
    for (i = j*6 ;i < (j*6+6) ; i++){
        temp = GET_BIT_ARRAY(out_g->Array[(i/8)],(i%8));
        in_s_box <<= 1;
        in_s_box |= temp ;
    }
    column_s = in_s_box & 0b00011110;
    column_s >>= 1;
    raw_s = GET_BIT(in_s_box,5);
    raw_s <<= 1;
    raw_s |= GET_BIT(in_s_box,0);
    out_s_box = arr[j][raw_s * 16 + column_s];
    S_boxes_out <<= 4;
    S_boxes_out |= out_s_box;
}
```

Figure 18 shows S-boxes code

```
//4-Permutation
out_g->Half[0]=0;
out_g->Half[1]=0;
 for (i = 0 ;i <= 31 ; i++){
    temp = (uint8)(((S_boxes_out<<(p[i]-1)) & 0x80000000)>>31) ;
    WRITE_BIT_ARRAY(out_g->Array[((i+32) / 8)],((i+32) % 8),temp);
 }
```

Figure 19 shows G-function permutation.

3.3 Round output

Output of G-function is xored with 32 bits of the left part as shown in figure 20 to generate right part.

Left part is the same as right of the previous round as shown in figure 20.

```
final_plain_text.Half[1] = final_plain_text.Half[1] ^ round_in.Half[0];
final_plain_text.Half[0]  = round_in.Half[1];
```

Figure 20 shows round output.

3.4 Key generation

- **PC-1**
  64 bits of the original key are permutated to output 56 bits as shown in figure 21 using the same permutation function shown in figure 14.

```
/* sub key */
//1 - pc-1
Permutation ( &final_original_key, &key_56,Pc1,pc1);
```

Figure 21 PC-1.

- **Rotate process in encryption**

  56 bits are splitted into two parts to be processed with left rotation as shown in figure 23, repeated due to the round number:

  For rounds 1,2,9,16 just one left rotation is applied, and two rotations in the remaining rounds.

  This is done using rotate function whose call is shown in figure 22.

```
//2- rotate
rotate(&key_56,round,&pc2_in);
```

Figure 22 shows rotate function call

```
#if (Operation == Encryption)
// left C
Temp &=0xfffffff0;
temp = GET_BIT(Temp,31);
Temp <<=1;
WRITE_BIT(Temp,4,temp);
for (i = 0 ; i<3 ; i++){
    Key->Array[i]= (uint8)((Temp >> ((3-i)*8)) & 0x000000ff);
}
Key->Array[3] &=0x0f;
Key->Array[3] |= (uint8)(Temp  & 0x000000f0);
//right
Temp = 0;
Temp |= Key->Array[3];
Temp = Temp <<8;
Temp |= Key->Array[4];
Temp = Temp <<8;
Temp |= Key->Array[5];
Temp = Temp <<8;
Temp |= Key->Array[6];
Temp= Temp <<4;
temp = GET_BIT(Temp,31);
Temp <<=1;
WRITE_BIT(Temp,4,temp);
Key->Array[3] &=0xf0;
Key->Array[3] |= (uint8)((Temp)>>28);
Temp <<=4;
for (i = 4 ; i<8 ; i++){
    Key->Array[i]= (uint8)((Temp >> ((7-i)*8)) & 0x000000ff);
}
```

Figure 23 shows left rotation

- **PC-2**

  After rotation PC-2 is applied using the permutation function shown in figure 14 on the 56 bits to generate the round key of 48 bits as shown in figure 24.

```
//3- pc-2
Permutation (&pc2_in,&subbkey,Pc2,pc2);
```

Figure 24 shows Permutation function call for pc-2

7. Decryption process

All functions used in encryption process are used in the decryption process except rotation process.

- **Rotation process in decryption**

  Here the rotation direction is right as shown in figure 25, with number of rotations differs with round number:

  Round 1 with no rotation, rounds 2,9,16 just one right rotation is applied, and two rotations in the remaining rounds.

```
#elif (Operation == Decryption)
if (round == 1){}
else{
    // left C
    Temp &=0xfffffff0;
    temp = GET_BIT(Temp,4);
    Temp >>=1;
    WRITE_BIT(Temp,31,temp);

    for (i = 0 ; i<3 ; i++){
        Key->Array[i]= (uint8)((Temp >> ((3-i)*8)) & 0x000000ff);
    }
    Key->Array[3] &=0x0f;
    Key->Array[3] |= (uint8)(Temp  & 0x000000f0);

    //right
    Temp = 0;
    for (i = 28 ;i <= 55 ; i++){
    temp = GET_BIT_ARRAY(Key->Array[(i / 8)],(i % 8));
    WRITE_BIT(Temp,(59-i),temp);
    }

    temp = GET_BIT(Temp,4);
    Temp >>=1;
    WRITE_BIT(Temp,31,temp);
    Key->Array[3] &=0xf0;
    Key->Array[3] |= (uint8)((Temp)>>28);
    Temp <<=4;
    for (i = 4 ; i<8 ; i++){
        Key->Array[i]= (uint8)((Temp >> ((7-i)*8)) & 0x000000ff);
    }
```

Figure 25 shows right rotation.

8. DES sixteen rounds

As shown in figure 14, DES consists of 16 rounds in both encryption and decryption as implemented as code in figure 26.

```
/************************************ Start Encrypt ***************************************/
    uint8 round = 0;
    DATA round_in = {.Array = {0,0,0,0,0,0,0,0}};
    DATA subbkey = {.Array = {0,0,0,0,0,0,0,0}};
    DATA key_56 = {.Array = {0,0,0,0,0,0,0,0}};
    DATA pc2_in = {.Array = {0,0,0,0,0,0,0,0}};
    /*************** 1 - initial Permutation ********************/
    Permutation ( &cipher_text, &round_in, Basic_Permutation, ip);
    for (i=0; i<8; i++){
        cipher_text.Array[i] = '0';
    }

    /********************************** Sub Key  ********************************/
    /*************** 1- PC-1 ********************/
    Permutation ( &final_original_key, &key_56,Pc1,pc1);
    /************************************ 16 Rounds ****************************/
    for (round = 1 ; round < 17 ; round++){
    /*************** 2- Rotate  ********************/
        rotate(&key_56,round,&pc2_in);
    /*************** 3. PC-2 ********************/
        Permutation (&pc2_in,&subbkey,Pc2,pc2);
    /********************************** G Function and XOR  *************************/
        G_Function(&round_in,&subbkey,&cipher_text);
        cipher_text.Half[1] = cipher_text.Half[1] ^ round_in.Half[0];
        cipher_text.Half[0]  = round_in.Half[1];
    /************************************ Update Variables ***************************/
        key_56.Half[0] = pc2_in.Half[0];
        key_56.Half[1] = pc2_in.Half[1];
        pc2_in.Half[0] = 0;
        pc2_in.Half[1] = 0;
        round_in.Half[0] = cipher_text.Half[0];
        round_in.Half[1] = cipher_text.Half[1];
        cipher_text.Half[0] = 0;
        cipher_text.Half[1] = 0;
    }
    pc2_in.Half[0] = round_in.Half[1];
    pc2_in.Half[1] = round_in.Half[0];

    /*************** 3- Final Permutation  ********************/
    Permutation (&pc2_in,&cipher_text,Basic_Permutation,fp);

/************************************ END Encrypt ***************************************/
```

Figure 26 shows DES sixteen rounds

9. SPI connections

For connections between ENC and DEC microcontrollers, a code simulates SPI communication protocol is implemented with four pins of MISO, SCLK, CS, MOSI with transmit function implementation shown in the figure 27 and receive function implementation shown in figure 28.

This represents the unsecure channel which Alice and Bob rely on to communicate.

```
void spi_transmit (char byte){
    char tt = 0;
    cs = 1;
    sclk = 0;
    cs = 0;
    for (tt = 0 ; tt <8 ; tt++){
        mosi = ((byte >> tt) & 0x01); // send b0 first
        spi_delay ();
        sclk = 1; // rx read at rising edge
        spi_delay2();
        sclk = 0;
        spi_delay ();

    }}
```

Figure 27 shows SPI transmit function

```
void spi_receive (char *byte){
    char ir = 0,x;
    while (cs == 0);
    for (ir = 0 ; ir <8 ; ir++){
        while (sclk == 0);
        x = mosi;
        spi_delay ();
        *byte = ((*byte & ~(1<<(ir))) | (x<<(ir)));

    }
}
```

Figure 28 shows SPI receive function

10. Alice and Bob communication process

- Alice and Bob enter DES key in hex format to be used in each session, then Alice enters her plaintext eight letters by eight letters in ascii MCU as shown in figure 29.
- The entered key and plaintext are sent to the MCU by TLL connector with baud rate 9600b/sec for encryption.
- DES encryption process is done over each eight letters at the first
- Each eight letters of the cipher text are sent on an unsecure channel using the SPI protocol and connections between first and second MCU.
- DES Decryption process is done over each received eight letters of cipher text at the second MCU.
- Each eight letters of deciphered text is sent to Bob Pc using TLL connector with baud rate 9600b/sec MCU as shown in figure 30.
- Alice can send plaintext continuously to Bob and decryption and encryption process is still occurred using the same key entered at the session start.
- To terminate a session, reset MCUs and enter a new key. It is recommended to change the key in each session to avoid the different types of attacks by Oscar. It is the main weakness of the DES algorithm or we can say the symmetric cipher algorithms.
-
- The hardware example is shown in figure 31.

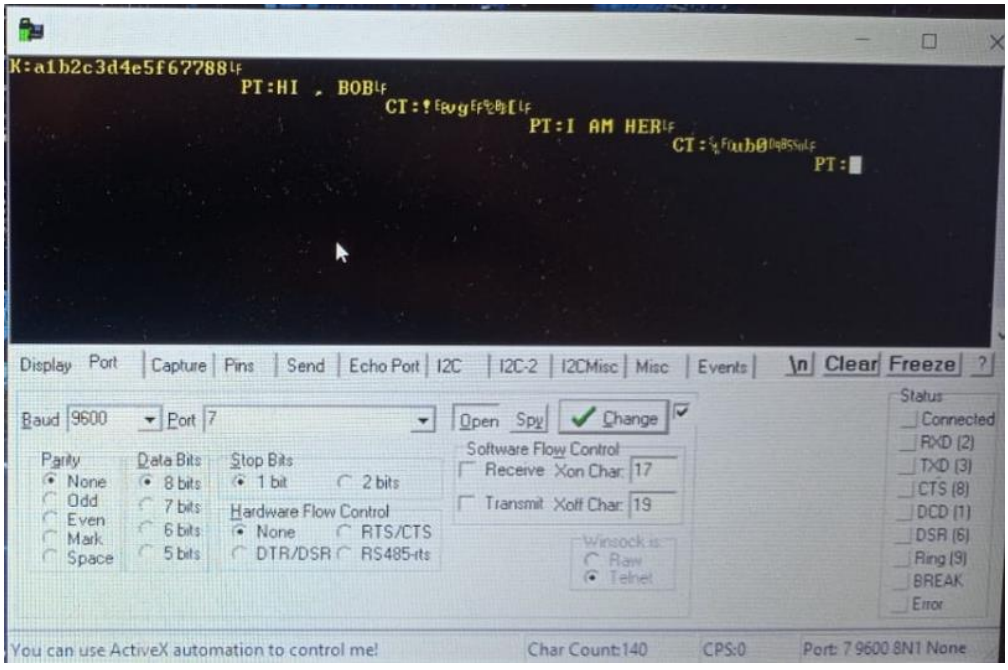

Figure 29 shows Alice's key and plaintext

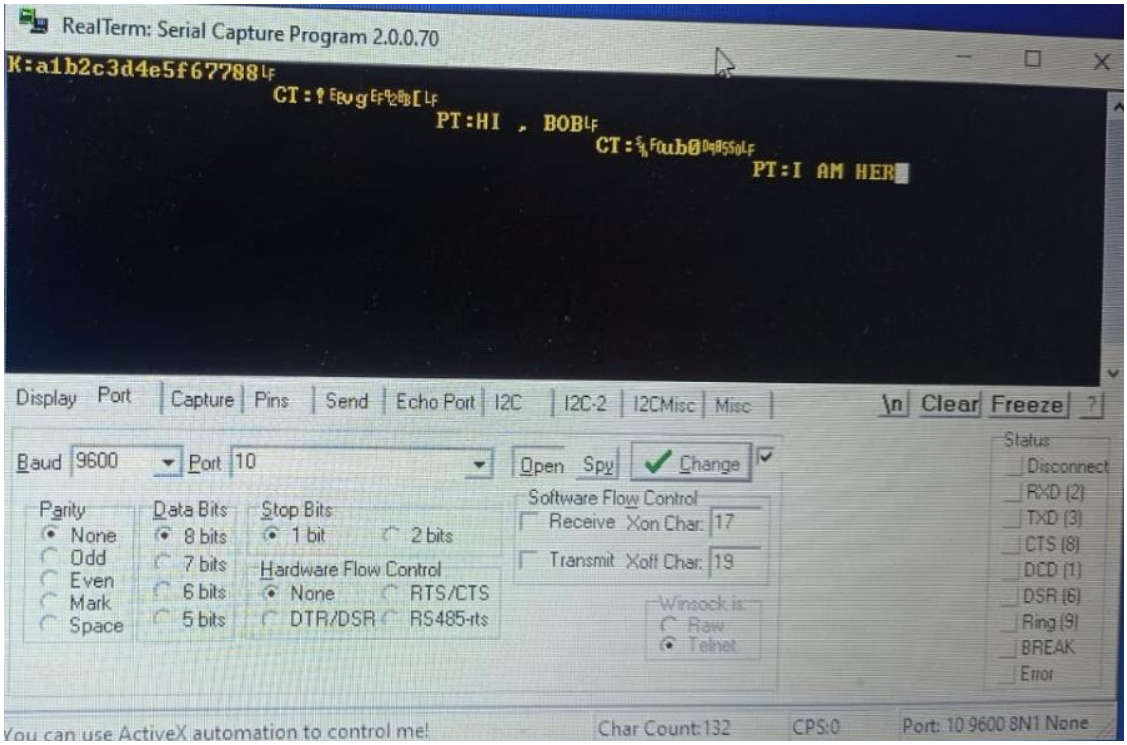

Figure 30 shows what Bob receives from Alice.

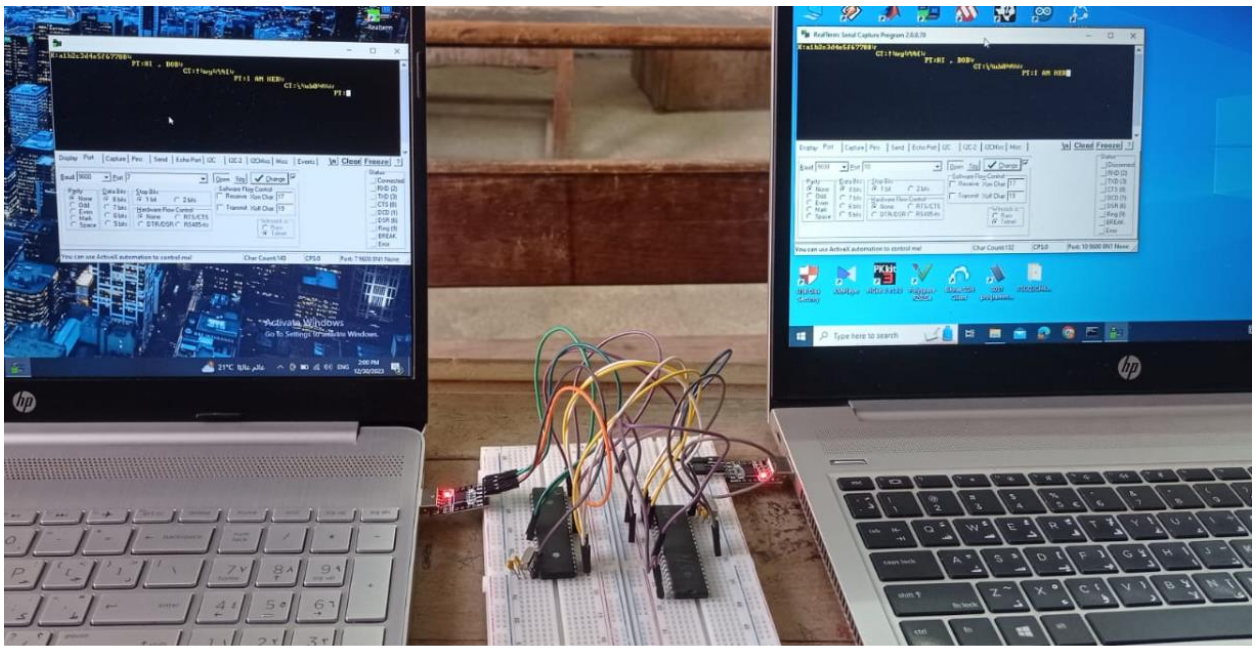

Figure 31 shows Hardware example of connection session between Alice and Bob