Implementation of RSA Cryptosystem on a Microcontroller for IOT Application

Project report submitted in fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Electronics and Communication Engineering
&
Computer Science and Engineering

By:
Arati Sharma
Saurabh Kumar Kar
Sourav Swain

Under Guidance of
Dr. Santosh Shah

Department of Electronics and Communication Engineering
The LNM Institute of Information Technology, Jaipur

July 2018

Certificate

This is to certify that the project entitled " Implementation of RSA Cryptosystem on a Microcontroller for IOT Application " submitted by Arati Sharma , Saurabh Kumar Kar and Sourav Swain in partial fulfillment of the requirement of degree in Bachelor of Technology (B. Tech), is a bonafide record of work carried out by them at the Department of Electronics and Communication Engineering,The LNM Institute of Information Technology, Jaipur,(Rajasthan) India, during the academic session 2018-2019 under my supervision and guidance and the same has not been submitted elsewhere for award of any other degree. In my opinion, this thesis is of standard required for the award of the degree of Bachelor of Technology (B. Tech).

Date : 19th July , 2018

_____

Supervisor : **Dr. Santosh Shah**

Abstract

Aim of the project is to design and implement a small cryptography system using arduino microcontroller.

RSA (Rivest–Shamir–Adleman) Cryptosystem is implemented on a microcontroller for end-to-end secure data transmission . The project is divided mainly in two categories :

1. **Hardware**
    - **a.** Connecting RF module for data transmission.
    - **b.** Connecting SD card module to store the data.
2. **Software**
    - **a.** Generation of Public and Private RSA keys.
    - **b.** Exchange of Encrypted RSA messages.
    - **c.** Generation of Decrypted message received through an insecure channel.

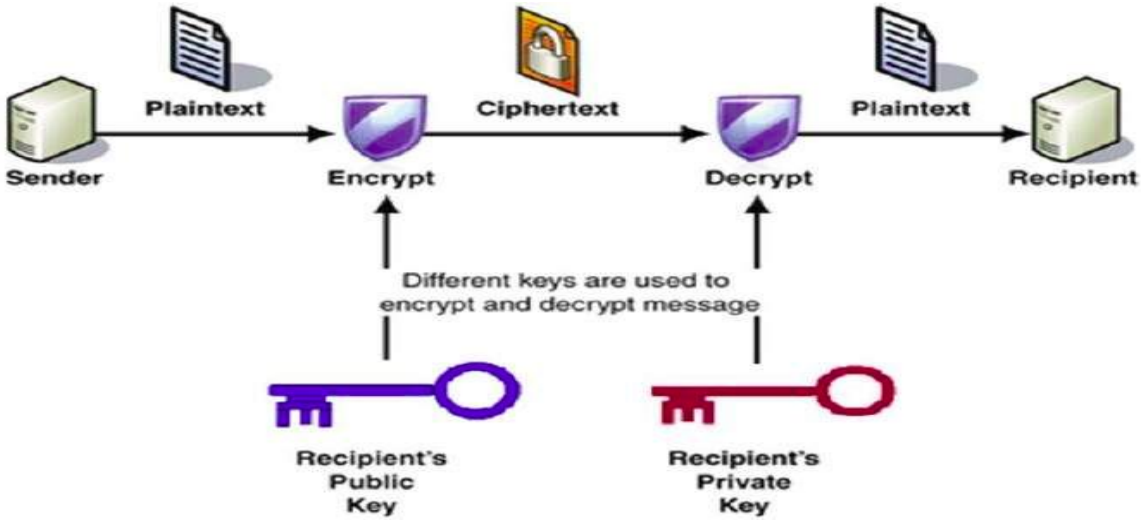
Contents

INTRODUCTION

A very important aspect in the world of software development is the security of data that flows through open communication channels. In our web applications, there is an intensive exchange of data via different protocols ,like http, between client applications which presented as browser, mobile and desktop applications and server side applications. The importance and confidentiality of data may be different depending on the specifics of the web application, and the possibility of interception by a third party increases with perfection of hacking techniques in the world of IT.
In order to make data secure , a secure cryptosystem called RSA is used, whose design and implementation were done from scratch.

RSA (Rivest–Shamir–Adleman) is one of the first Public Key Cryptosystem. Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1978. RSA is widely used for secure data Transmission. In such a Cryptosystem, the Encryption key is Public and it is different from the Decryption key which is kept secret (private). In RSA, this asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers, the "factoring problem".

The original message is encrypted upon passing through encryption function of RSA algorithm. The encrypted message is then transmitted over insecure radio frequency channel to the client . The client receives the encrypted message which is then passed through decryption function of RSA algorithm . Upon transmission over insecure RF channel the unauthorized person can access the encrypted message but due to absence of private key , the encrypted message cannot be decrypted .Hence , this project summarises the design and implementation of RSA cryptography in a microcontroller as the encryption / decryption function.



PROJECT PLANNING AND SCHEDULING

1. Project Development Model

Key Generation	
Select p, q	p, q both prime, p≠q
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1) \times (q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod n$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod n$

2. Project Plan and algorithm

The RSA algorithm involves four steps: key generation, key distribution, encryption and decryption.
The keys for the RSA algorithm are generated the following way:

- **a.** Choose two distinct prime numbers p and q. For security purposes, the integers p and q should be chosen at random, and should be similar in magnitude but differ in length by a few digits to make factoring harder.
- **b.** Compute n = pq.
  - **i.** n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
- **c.** Compute $\lambda(n) = \text{lcm}(\lambda(p), \lambda(q)) = \text{lcm}(p - 1, q - 1)$, where λ is Carmichael's totient function. This value is kept private.
- **d.** Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; i.e., e and $\lambda(n)$ are coprime.
  - **i.** This is more clearly stated as: solve for d given $d \cdot e \equiv 1 \pmod{\lambda(n)}$.

The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the private (or decryption) exponent d, which must be kept secret. p, q, and $\lambda(n)$ must also be kept secret because they can be used to calculate d.

ENCRYPTION:

$$c \equiv m^e \pmod{n}$$

Where, c= ciphertext
m= ASCII of the message
e= public key exponent.
n= p*q

DECRYPTION:

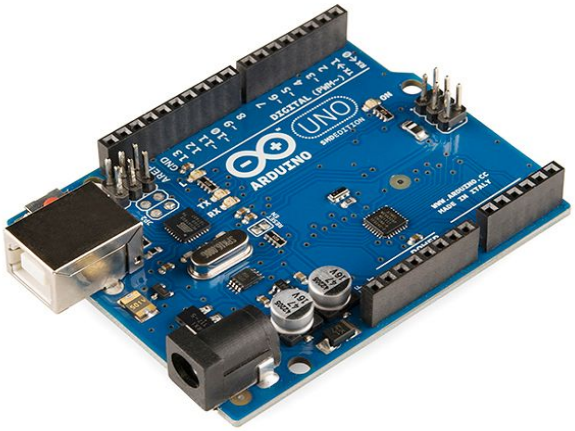
$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

where , d = private key exponent

SYSTEM REQUIREMENT STUDY

HARDWARE REQUIRED

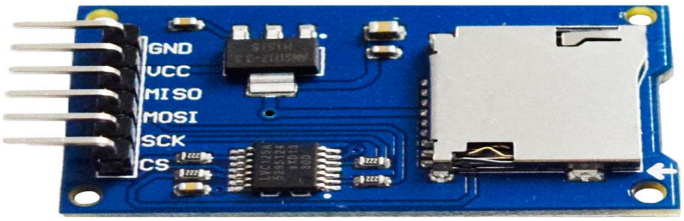
1. ARDUINO UNO MICROCONTROLLER



Arduino is an open source computer hardware and software company . In this project Arduino acts like a motherboard to the design . The whole project is controlled mainly through arduino . All other hardware are connected to arduino via serial port for communication and storage purpose . The Arduino is coded in such a way that it handles the other hardware functioning efficiently . Arduino (in this project) supports wireless communication through RF 433 Mhz (used as other hardware). The Arduino encrypts and decrypts the data . The two Arduino behave as encryption and decryption function respectively. The processing power for encryption and decryption is further provided by Arduino itself .

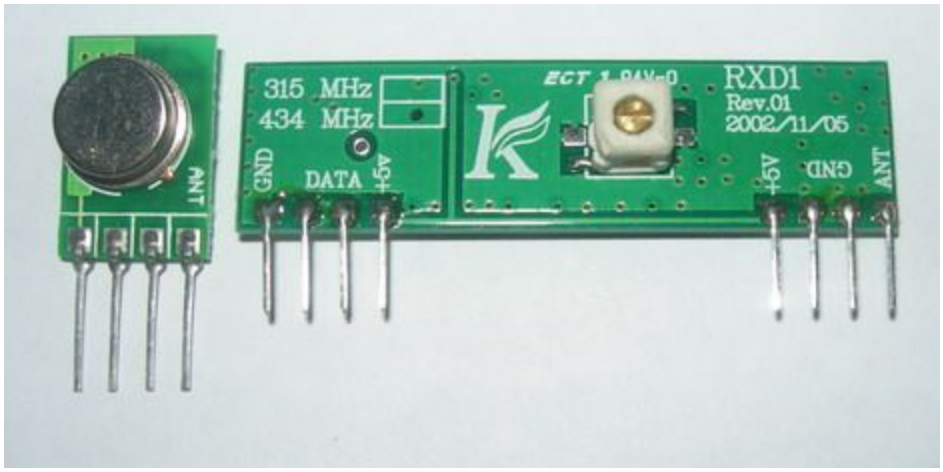
2. SD CARD MODULE

The SD Card Module provides mass storage , such as image or long text for transferring data to and from a standard SD card. In total , there were two SD CARD modules used , one at the transmitter end and other at the receiver end.



3. RF 433 MODULE

This Radio Frequency (RF) module employs Amplitude Shift Keying (ASK) with transmitter/receiver (Tx/Rx) pair operating at 434 MHz. The transmitter module takes serial input and transmits these signals through RF . In our project , the image file (.bmp) was stored at the transmitter end in SD card , which got converted to binary form and then to cipher text by the transmitter code , and it was then, transmitted to the receiver end. On receiving the code, the cipher text gets converted to binary and then to image form. Hence, we get a complete replica of the image sent.



SOFTWARE REQUIREMENT

1. Arduino Software (IDE)

The Arduino integrated development environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in the programming language Java. The Arduino IDE supports the languages C and C++ using special rules of code structuring. We used C language for our encryption as well as decryption process with modification.

A minimal Arduino program consist of only two functions:

- setup(): This function is called once when a sketch starts after power-up or reset. It is used to initialize variables, input and output pin modes, and other libraries needed in the sketch.
- loop(): After setup() function exits (ends), the loop() function is executed repeatedly in the main program. It controls the board until the board is powered off or is reset.

2. Radiohead library

It provides a complete object-oriented library for sending and receiving packetized messages via a variety of common data radios and other transports on a range of embedded microprocessors.

RadioHead consists of 2 main sets of classes: Drivers and Managers.

- Drivers provide low level access to a range of different packet radios and other packetized message transports.We used RH_ASK as driver.
- Managers provide high level message sending and receiving facilities for a range of different requirements.

3. SD Library

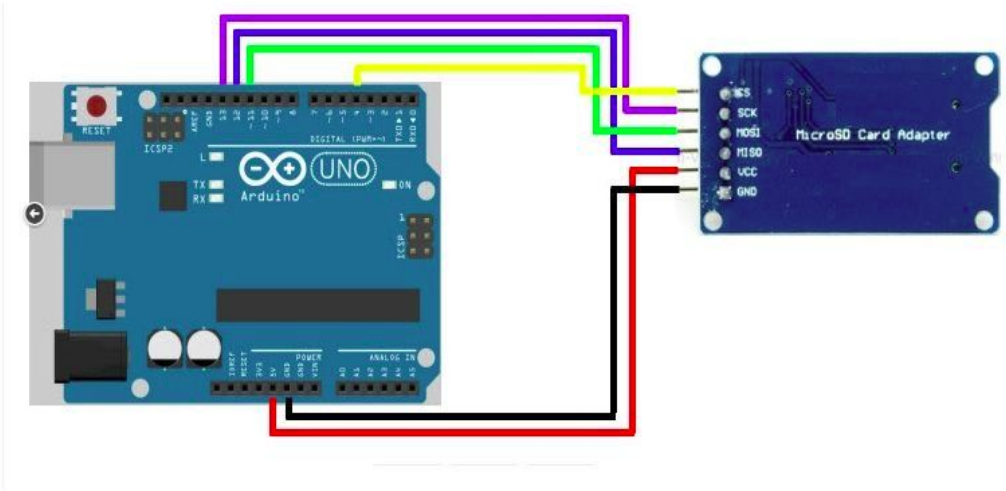
The SD library allows for reading from and writing to SD cards, e.g. on the Arduino Ethernet Shield. The communication between the microcontroller and the SD card uses SPI, which takes place on digital pins 11, 12, and 13 (on most Arduino boards) .

System Design and Implementation

SD Card Module

1. Connection with the microcontroller

SD MODULE		MICROCONTROLLER
CS	----------	PIN 4
SCK	----------	PIN 13
MOSI	----------	PIN 11
MISO	----------	PIN 12
VCC	----------	+5V
GND	----------	GND



RF-433 Mhz Module

1. Downloading Radiohead Library
   - **a.** Download the Radiohead library from the link given below: http://www.airspayce.com/mikem/arduino/RadioHead/RadioHead-1.41.zip
   - **b.** Unzip the RadioHead library
   - **c.** Install the RadioHead library in your Arduino IDE
   - **d.** Restart your Arduino IDE
2. Connecting Transmitter and Receiver with microcontroller.

TRANSMITTER MICROCONTROLLER

DATA ---------- PIN 7

VCC ---------- +5V

GND ---------- GND

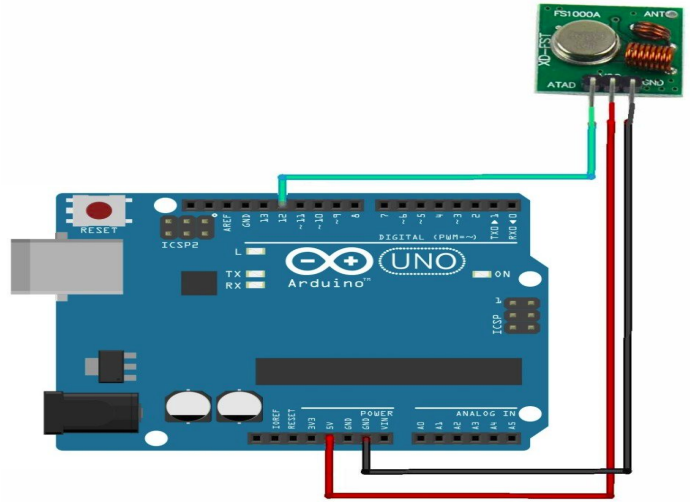
RECEIVER MICROCONTROLLER

DATA ---------- PIN 7

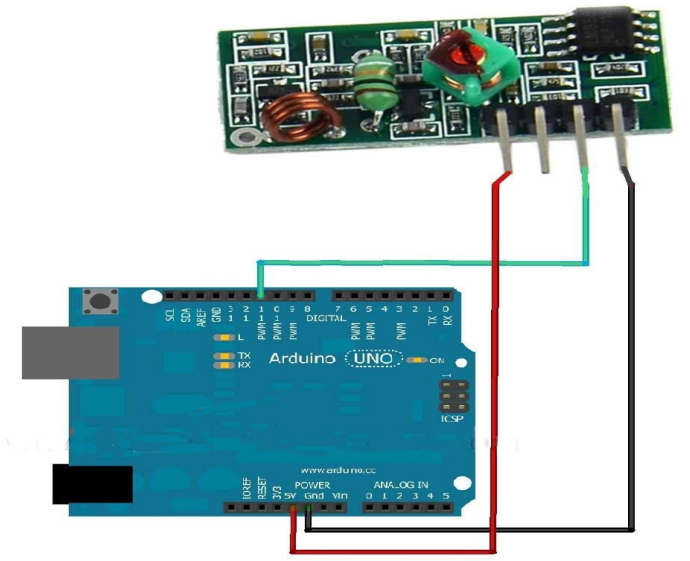
VCC ---------- +5V

GND ---------- GND

TRANSMITTER :



RECEIVER:



3. Sample code transmitter and receiver

TRANSMITTER :

```
#include <RH_ASK.h>
#include <SPI.h>
RH_ASK driver;

void setup()
{
   Serial.begin(9600);
   if (!driver.init())
   Serial.println("init failed");
}

void loop()
{
   const char *msg = "Hello World!";
   driver.send((uint8_t *)msg, strlen(msg));
   driver.waitPacketSent();
   delay(1000);
}
```

RECEIVER:

```
#include <RH_ASK.h>
#include <SPI.h>

RH_ASK driver;

void setup()
{
   Serial.begin(9600);
   if (!driver.init())
        Serial.println("init failed");
}

void loop()
{
   uint8_t buf[12];
```

```
  uint8_t buflen = sizeof(buf);
  if (driver.recv(buf, &buflen))
  {
    Serial.print("Message: ");
    Serial.println((char*)buf);
  }
}
```

4. Final Coding for the transmitter and receiver to transmit and receive text / image data.

RSA Transmitter Code Arduino:

```
#include <SPI.h>
#include<SD.h>
#include<RH_ASK.h>

RH_ASK driver(4700, 7, 7);

int gcd(int n1, int phi);
int countprime(int n);

void setup() {
  Serial.begin(9600);
  SD.begin(4);
  driver.init();

//IMAGE TO BINARY**************************************************
  Serial.print("Image to Binary: ");
  SD.remove("binary.txt");
  File image = SD.open("aa.bmp", FILE_READ);
  File binary = SD.open("binary.txt", FILE_WRITE);
  int i, c;

  while ((c = image.read()) != EOF)
  {
    for (i = 0; i <= 7; i++)
    {
      if (c & (1 << (7 - i)))
      {
        binary.print('1');
      }
```

```
    else
    {
      binary.print('0');
    }
  }
}
image.close();
binary.close();
Serial.print("DONE\n");


//*************************************************
int p = 7;
int q = 17;
int n = p * q;
int phi = (p - 1) * (q - 1);
int size = countprime(n);
int n1[size];

int count = 0;
for (int i = 2; i <= n; i++)
{
  int isprime = 0;
  for (int j = 2; j <= i / 2; j++)
  {
    if (i % j == 0)
    {
      isprime = 1;
      break;
    }
  }
  if (isprime == 0 && n != 1 )
  {
    n1[count] = i;
    count++;
  }
}

Serial.print("\nn is: ");
Serial.print(n);
Serial.print("\nphi is: ");
Serial.print(phi);

i = 0;
```

```
int cd = 0;
int e;
while (cd != 1)
{
  i = i + 1;
  cd = gcd(n1[i], phi);
}
e = n1[i];
Serial.print("\ne is: ");
Serial.print(e);

int val = 0;
int d = 0;
while (val != 1)
{
  d = d + 1;
  val = (d * e) % phi;
}
Serial.print("\nd is: ");
Serial.print(d);



int dec;
dec = e;
int bin_len = 0;
while (dec != 0)
{
  dec /= 2;
  bin_len++;
}
int en[bin_len];
dec = e;
i = 0;
while (dec > 0)
{
  en[i] = dec % 2;
  i++;
  dec /= 2;
}

Serial.print("\nBinary of e is: ");
for (i = bin_len - 1; i >= 0; i--)
{
```

```
  Serial.print(en[i]);
}
Serial.print("\nNos. of digits of e is: ");
Serial.print(bin_len);

//ENCRYPTION*************************************************
size = bin_len - 1;
SD.remove("cipher.txt");
binary = SD.open("binary.txt", FILE_READ);
File cipher = SD.open("cipher.txt", FILE_WRITE);

char ch;
Serial.print("\n\nEncryption: ");
while ((ch = binary.read()) != EOF)
{
  int c = 1;
  for (p = size; p >= 0; p--)
  {
    if (en[p] == 1)
    {
      c = fmod(fmod(pow(c, 2), n) * ch, n);
    }
    else if (en[p] == 0)
    {
      c = fmod(pow(c, 2), n);
    }
  }
  cipher.print(c);
  cipher.print(" ");
}
binary.close();
cipher.close();

Serial.print("DONE\n");


//TRANSMISSION**********************************************
Serial.print("\nTransmitting: \n");
cipher = SD.open("cipher.txt", FILE_READ);
int final = 0;
int msg;
delay(10000);
ch = cipher.read();
while (ch != EOF)
```

```
  {
   if (ch != ' ')
   {
    msg = ch - '0';
    final = final * 10 + msg;
    ch = cipher.read();
   }
   else
   {
    String combine = String(final);
    combine = combine + " ";
    static char *text = combine.c_str();
    driver.send((uint8_t *)text, strlen(text));
    driver.waitPacketSent();
    Serial.print(text);
    delay(150);
    ch = cipher.read();
    final = 0;
   }
  }

  Serial.print("Done");
  Serial.print(millis());
}

void loop() {
}

int gcd(int a, int b)
{
  if (a == 0 || b == 0)
    return 0;
  if (a == b)
    return a;
  if (a > b)
    return gcd(a - b, b);
  return gcd(a, b - a);
}

int countprime(int n)
{
  int j;
  int isprime;
  int count = 0;
```

```
  for (int i = 2; i <= n; i++)
  {
   isprime = 0;
   for (int j = 2; j <= i / 2; j++)
   {
    if (i % j == 0)
    {
     isprime = 1;
     break;
    }
   }
   if (isprime == 0 && n != 1 )
   {
    count++;
   }
  }
  return count;
}
```

RSA Receiver Code Arduino:

```
#include <SPI.h>
#include<SD.h>
#include<RH_ASK.h>
RH_ASK driver(4700,7,7);
File cipher;


void setup(){

  Serial.begin(9600);
  if (!driver.init())
    Serial.println("init failed");
SD.begin(4);
//*********************************************************************
  SD.remove("cipher.txt");
  Serial.println("Start");
  Serial.print("Message: ");
}

void loop() {
  uint8_t buf[RH_ASK_MAX_MESSAGE_LEN]={0};
```

```
  uint8_t buflen = sizeof(buf);

  char *temp;
  if (driver.recv(buf, &buflen)) // Non-blocking
  {
          cipher= SD.open("cipher.txt",FILE_WRITE);
          temp = ((char*)buf);
          cipher.print(temp);
          Serial.print(temp);
          cipher.close();

  }

}
```

RSA Decryption code :

```
#include <SPI.h>
#include<SD.h>

char bytefromtext(char* text);

File cipher;

void setup(){

  Serial.begin(9600);

  SD.begin(4);

//**********************************************************************
**************
  SD.remove("cipher.txt");
}

  void setup()
  {
 Serial.begin(9600);
 SD.begin(4);

//**********************************************************************
**************
```

```
Serial.println("Start");

int n=119;
int d=77;
int dec = d;
int bin_len = 0;
while (dec != 0)
{
dec /= 2;
bin_len++;
}

int bin11[bin_len];
dec = d;
int i = 0;
while (dec > 0)
{
bin11[i] = dec % 2;
i++;
dec /= 2;
}

Serial.print("\nBinary of d is: ");
for (i = bin_len - 1; i >= 0; i--)
{
Serial.print(bin11[i]);
}
Serial.print("\nNos. of digits of d is: ");
Serial.print(bin_len);


//DECRYPTION**************************************************
int size = bin_len - 1;
File cipher = SD.open("cipher.txt", FILE_READ);
File test;
SD.remove("test.txt");
test = SD.open("test.txt", FILE_WRITE);
char num;
Serial.print("\n\nDecryption: ");

int msg;
int final = 0;
num = cipher.read();
while (num != EOF)
```

```
{
if (num != ' ')
{
msg = num - '0';
final = final * 10 + msg;
num = cipher.read();
}
else
{
int c = 1;
for (int p = size; p >= 0; p--)
{
if (bin11[p] == 1)
{
c = fmod(fmod(pow(c, 2), n) * final, n);
}
else if (bin11[p] == 0)
{
c = fmod(pow(c, 2), n);
}
}
test.print((char)c);
num = cipher.read();
final = 0;
}

}
test.close();
cipher.close();
Serial.print("DONE\n");

                                                //                                                IMAGE
RECREATION**************************************************
Serial.print("\nImage Recreation: ");
SD.remove("new.bmp");
test = SD.open("test.txt", FILE_READ);
File image = SD.open("new.bmp", FILE_WRITE);
char rec;
char buf[8];
int j = 0;
while ((rec = test.read()) != EOF)
{
buf[j++] = rec;
if (j == 8)
```

```
 {
 image.print(bytefromtext(buf));
 j = 0;
 }
 }
 Serial.print("Done");

 test.close();
 image.close();
 }

void loop() { }

 char bytefromtext(char* text)
 {
 int i;
 char result = 0;
 for (i = 0; i < 8; i++)
 {
 if (text[i] == '1')
 {
 result |= (1 << (7 - i) );
 }
 }
 return result;
 }
```

RESULTS

Image used :

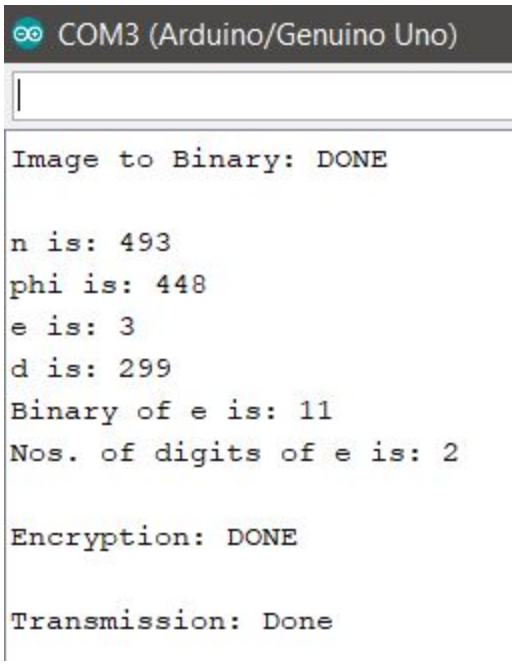
Cipher Text (only few bytes taken into account)

70 97 97 97 70 97 97 70 97 70 97 70 97 97 97 97 97 70 97 97 70 70 70 97 97 70 97 97 97 70 70 70 97 97 97 97
70 70 97 70 97 97 97 97 70 97 70 97 97 97 97 97 70 70 97 70 97 97 97 97 97 70 97 70 97 97 97 97 97 97 97 97
97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 70 70 97 70 97 70 97 97 70 97 97 70 97 70 97 97
70 97 97 97 97 70 97 97 97 70 97 97 97 70 97 70 97 97 70 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97
97 97 97 97 97 97 97 97 97 97 70 70 97 70 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97
97 97 97 97 97 97 70 97 70 97 97 97 97 97 97 97 97 70 97 97 97 97 97 97 97 97 70 70 97 97 97 97 97 97 97 97 97
97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 70 97 70 70 97 97 97 70 97 70 70 97 70 70 97 97 70 70 97 70
70 97 97 70 97 97 97 97 70 97 70 70 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97 97
97 97 97 97 97 97 97 70 97 70 70 70 97 97 70 70 97 70 97 70 97 97 70 97 97 70 97 97 97 70 70 70 97**… about 6000 more bytes.**

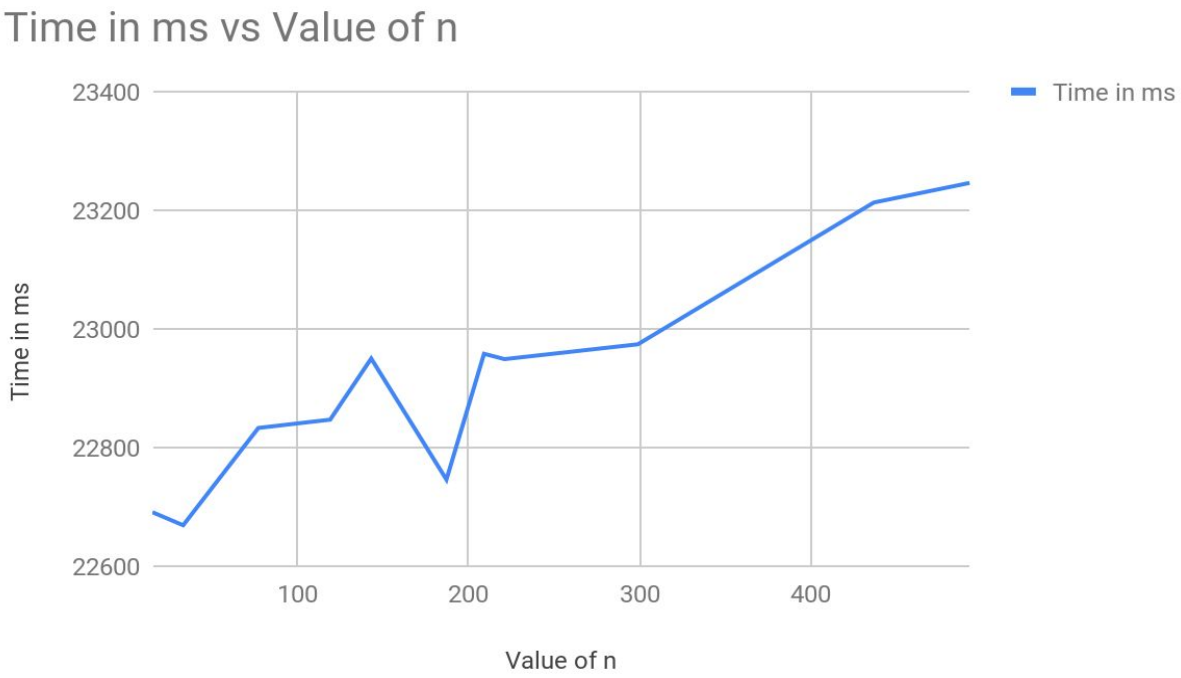
Image Binary

1000100101010000010011100100011100001101000010100001101000001010000000000000000000000000000110101001
0010100100001000100010100100000000000000000000000000011010000000000000000000000000000010100000001000000
0001100000000000000000000000000010110001011011001101100100000101100000000000000000000000000000010111001
1010100100100011101000010000000001010111011001110000111001110100100000000000000000000000000001000110
0111010000010100110101000001000000000000000001011000110001111000010111111110001100001000001010000000000
0000000000000000000010010111000001001000010110010111001100000000000000000001001001110100000000000000000
0000010010011101000000000111011110011001100001111101111000000000000000000000000000010010101010010010100
0010001000001010101000110100001000011111011011001010101000001000010101100000000100000000011000000010 0
1001001111111110111111111100111101101101101000011001111011110110000100100100001010101100101011000111 0
0011110011110100010010100001101010000001010001100111110100011110111100010110011010111100010000110100 1
1000110001101101110100001000010111011011010010000011011011100111111011100110000111110111001100010011 1
0101100001011101010111111100101000110001101000111000011010010000101010101111001110110011100110110011 0
1001111001011101100001010001010000111110100011111100110111110110010010011011100100100111001011101100 1
0011100110101110101000011010100000011000010111011001111001101101001011001011101000111110011010100100 1
0111100101110110001110100111011111111010000111010100111011110100110000010100011011000100101111001011 0
1100000111001101011100100001011101000111010001110101011000111101000111111001011010100110100000110111
1101010001010000111101001011110101001010111101010111110000101001110001110100111011111111100001101011 0
0101010000011010011010010001101010000000000010100101111110000011101001010010100011110001111000110100 1
1000100100011111001010101100011101001111101100010000100110011000001111111011111110111001011110000111 1
1110000101111011110011001011101000000000000000000000000000000000000010010010100010101001110010001001010
1110010000100110000010000010

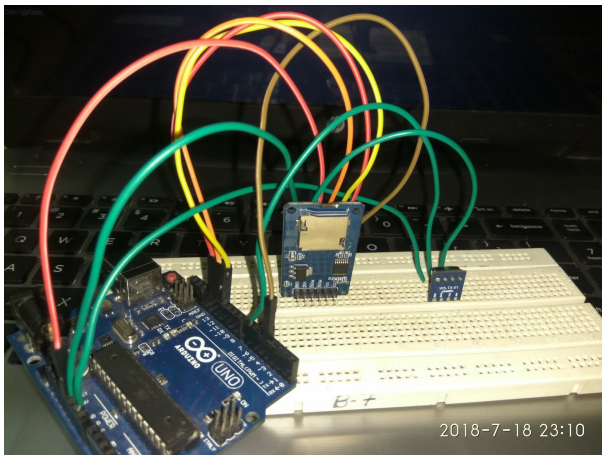
Transmitter Output

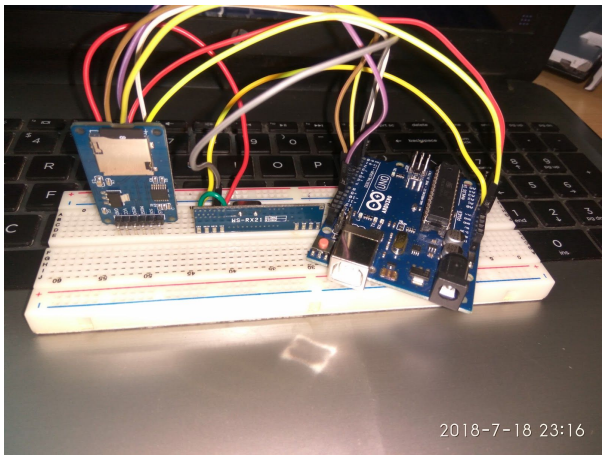


Encryption time for different values of n:



Project Images





Tx Side **Rx Side**

Other Devices for Better Performance

1. **NRF24L01L Long Range Wireless Module**

   Features:

   - nRF24L01+ is a single-chip radio transceiver for the worldwide 2.4-2.5 GHz ISM band, suitable for Arduino and Raspberry Pi.
   - Supports 6-channel receiving, with a transmission speed of up to 2Mbps.
   - Multi-frequency point: With 125 frequency points, it can meet multipoint communication and frequency hopping communication requirements.
   - Ultra-small: 24*15mm, with built-in 2.4GHz antenna.
   - Easy to develop: the link layer is fully integrated on the module, making it easy for development.

2. **Raspberry Pi for high bits transfer and higher security**

   A Raspberry Pi is a general-purpose computer, usually with a Linux operating system, and the ability to run multiple programs. It is more complicated to use than an Arduino.Raspberry Pi is best used when you need a full-fledged computer: driving a more complicated robot, performing multiple tasks, doing intense calculations (as for Bitcoin or encryption).

Limitations

- If transmitter and receiver placed about 100cm far away then it results in some packet loss.
- Currently transmission speed is kept at 4700 bps and a delay of 150 ms between each cipher text transmission. If speed increased more than 5000 bps then there is significant loss of packets. Removing delay has same impact.
- Decryption of cipher text and reconstruction of image can only be done if a huge delay more than 1500 ms is kept during cipher text transmission. Else transmitter and receiver lose sync because arduino is busy processing decryption and reconstruction rather than receiving.
- Processing power of arduino is very low.
- RadioHead library can't send more than 60 byte at same time.
- Transmission time and size of cipher text directly proportional. 6kb cipher text takes 6 minutes.

Conclusion

This Project is a prototype of how image can be encrypted and transmitted by using RF Module 433 Mhz. In this project we got to work in currently biggest emerging field of IOT security. We learnt about RSA cryptosystem, arduino, RF and SD module. We were able to convert image to binary and then encrypt it using RSA. Then this cipher text was transmitted by RF 433 module. In receiver side this cipher text was decoded and image was constructed again.The same steps were followed for text also.
It was a learning experience and we shall extend our work in the near future.

References

- RSA Wikipedia Page
- Rivest, Ronald. "The Early Days of RSA -- History and Lessons"
- Randomnerdtutorials for RF 433 Mhz.
- Arduino examples on SD Card .