Muhammad Ali Farooq | 331879

The Vignere Cipher Used to Demonstrate Secure Encrypted Communication between Two Arduinos.

Semester Project

FoP

**Project Summary:**

In this project, I have extended the bounds of the Vigenère cipher and used it to demonstrate encrypted transmission between two Arduino MEGA 2560’s, over both wired serial and radio communication

The Vigenère cipher is an encryption method, used to encrypt alphabetic text through a series of interwoven Caesar ciphers (each of which is based on the letters of a key). The cipher forms a type of poly-alphabetic substitution.

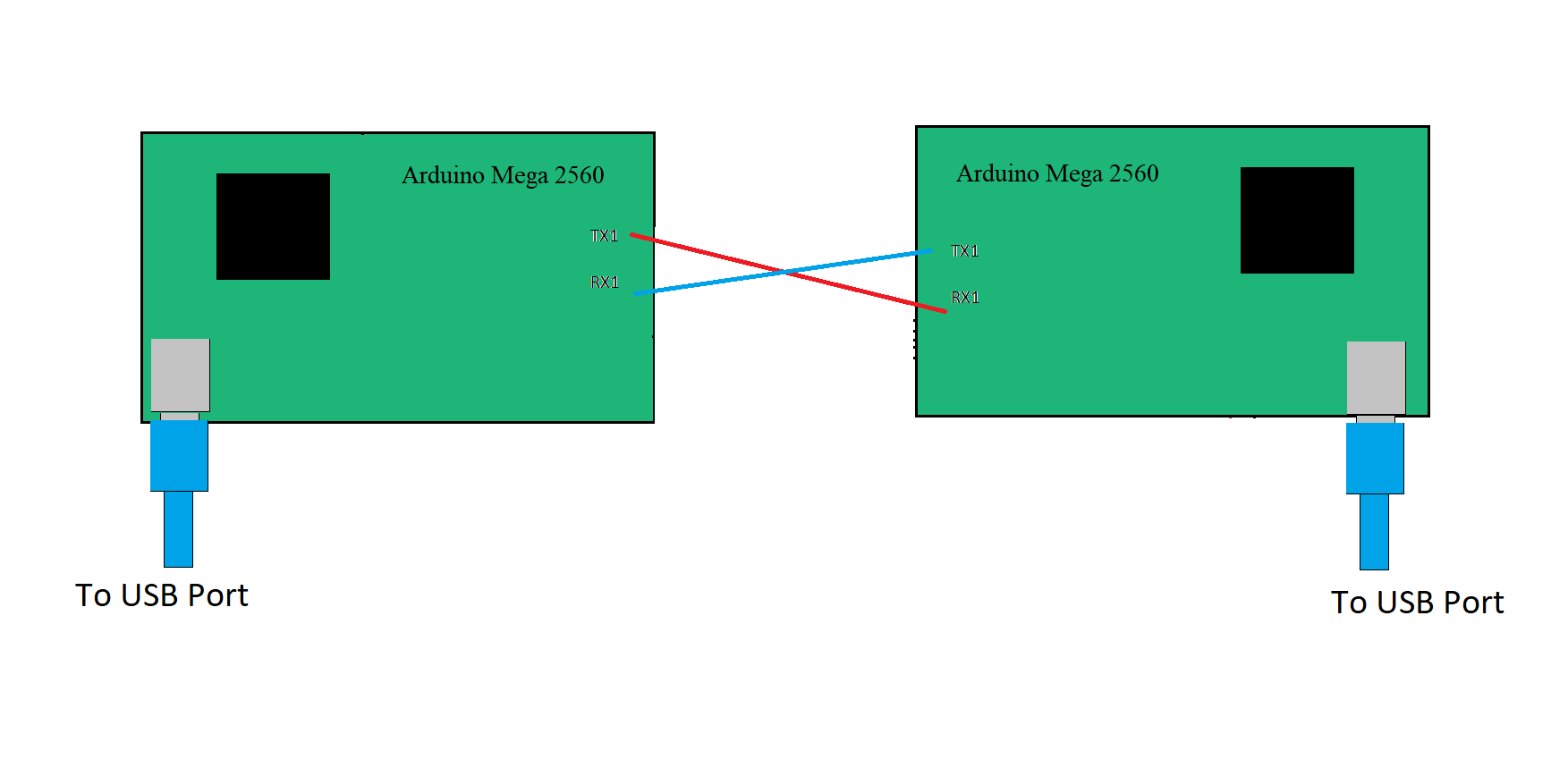
I have extended the bounds of the Vigenère cipher to include all printable ASCII characters (that is, characters within the integer range 33-126). Its implementation has also been modified to ignore whitespaces.

**Serial Implementation:**

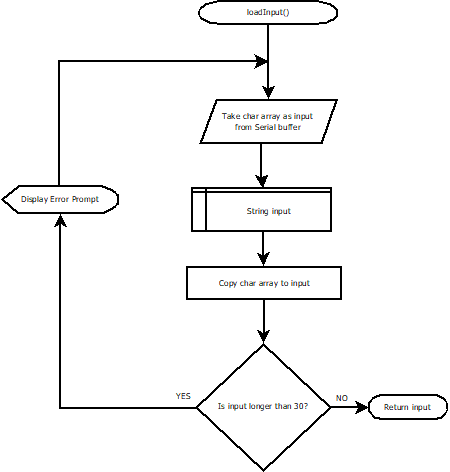
The Arduinos communicate with each other via TTL interface over their Serial1 port.

It is a very simple arrangement, one created to demonstrate just the use of the ciphering and deciphering functions.

**Schematic:**

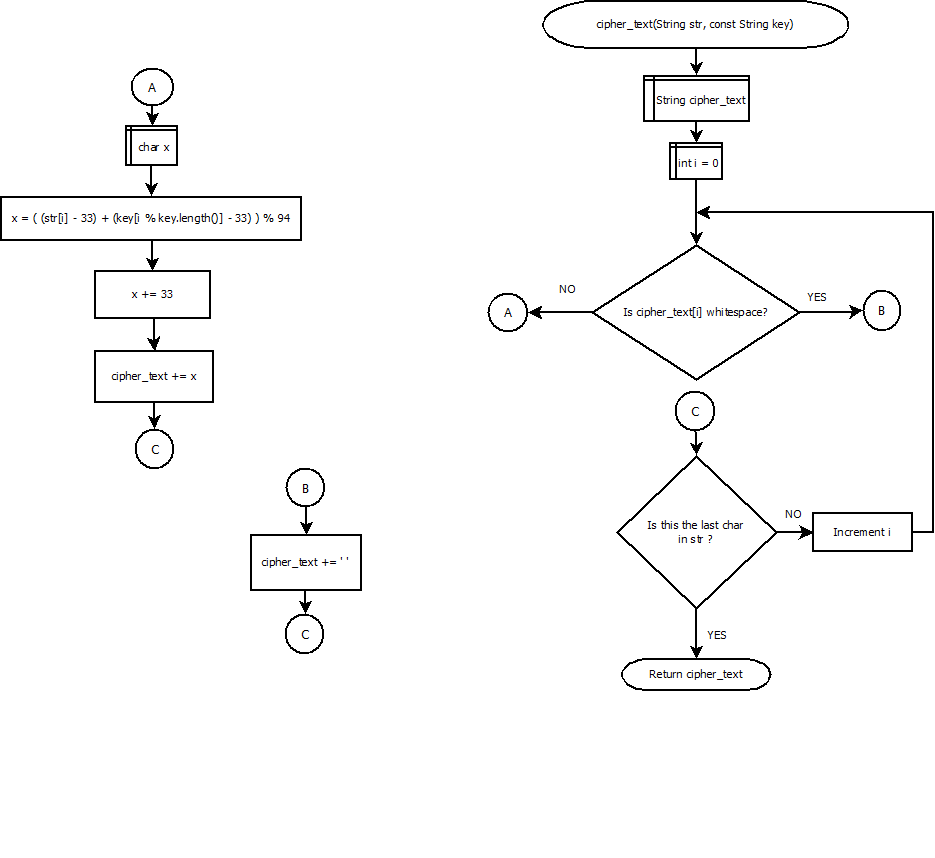


**Function loadInput():**

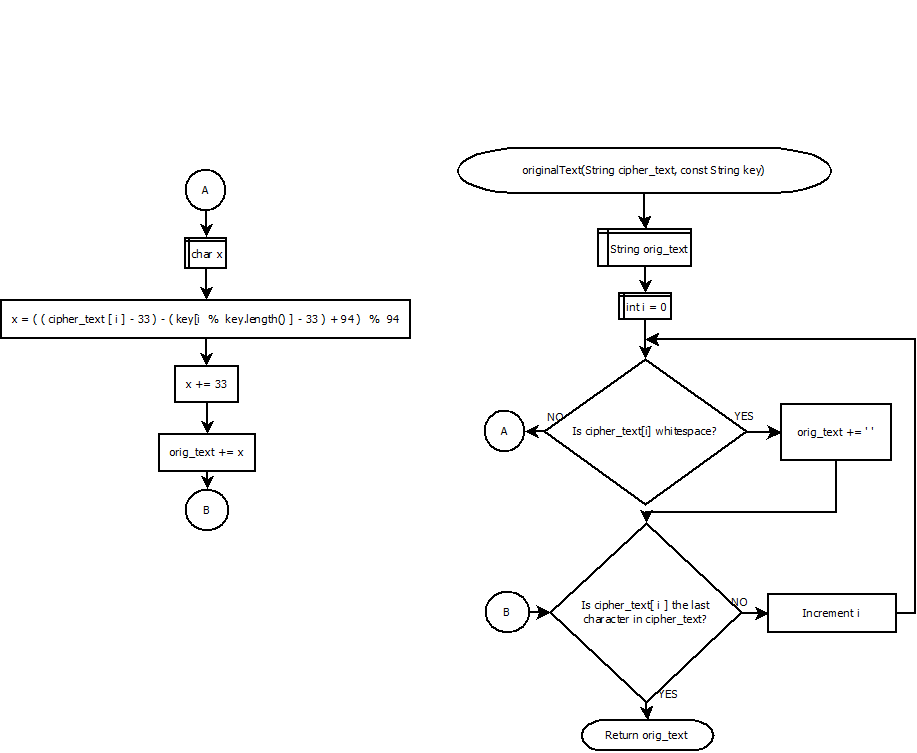
****

This function checks the length of the entered string. If it is not within acceptable bounds, it prompts the user to enter the string again.

**Function cipher\_text():**

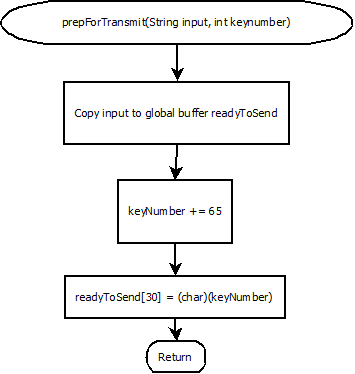
‘

This is the function responsible for encrypting the message. Through the use of a key (used as a circular buffer), which is stored in a global array, it encrypts the message. It ignores whitespaces, and includes them as is in the encrypted message.

**Function originalText():**

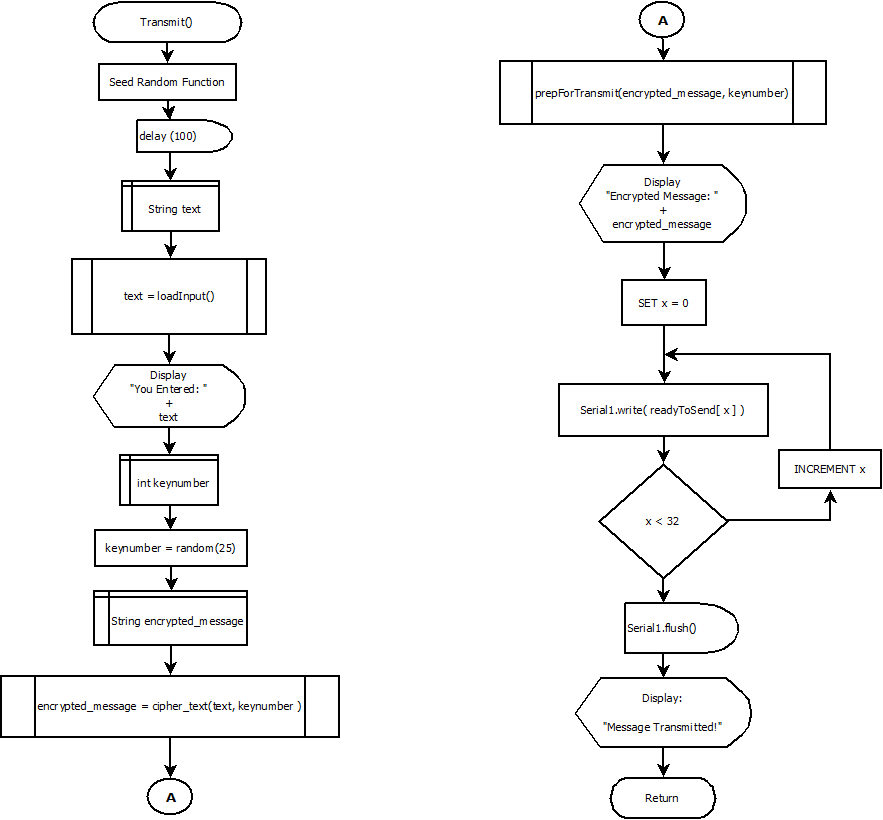
This function is responsible for deciphering a received message. It also uses a stored key, similar to the function for encryption.

**Function prepForTransmit():**



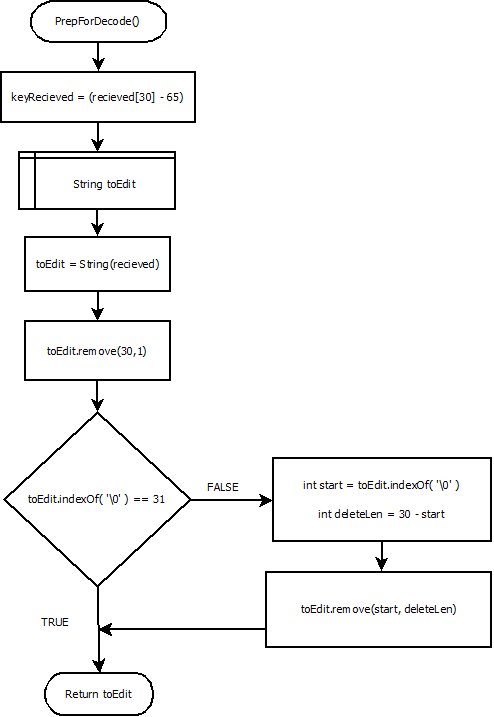
This function appends the index of the chosen key in the global key array to the encrypted message.

**Function Transmit():**



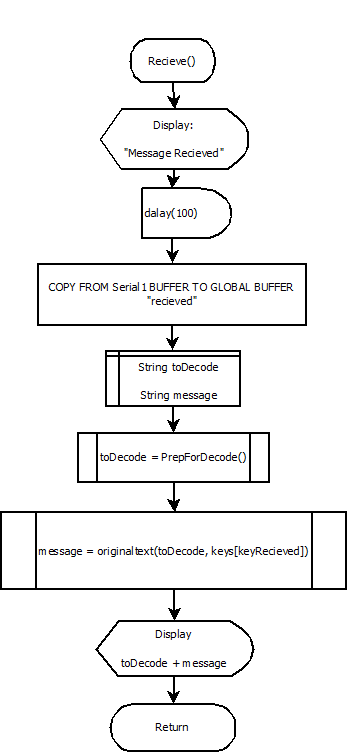
This function is responsible for taking input from the user, and calling the functions loadInput(), cipher\_text(), and prepForTransmit(). It then transmits the message character by character over Serial port 1.

**Function PrepForDecode():**



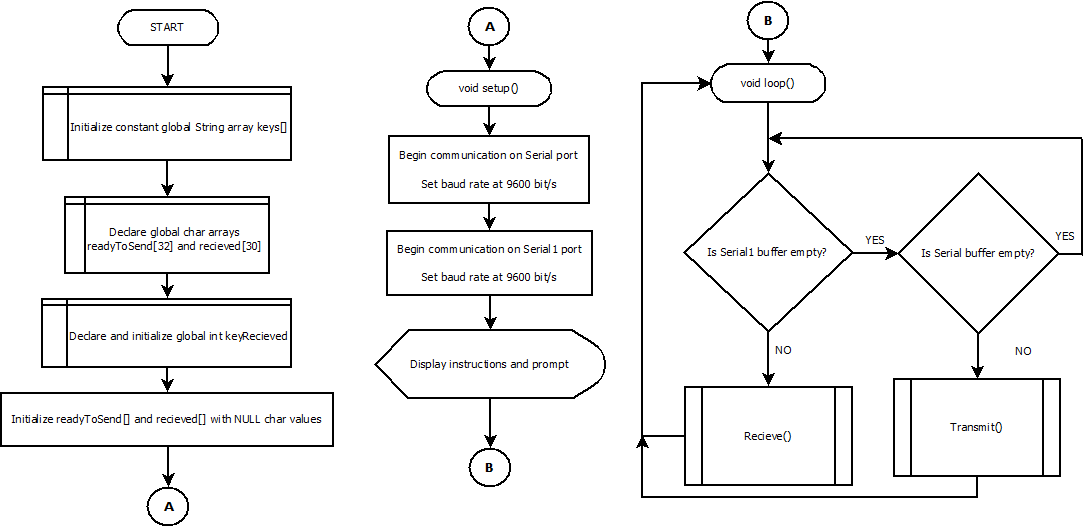
This function removes the key index number from the received message, and stores it in a global variable. It also prunes the null characters from the received message, making it ready for decryption.

**Function Receive():**



This function handles the preparation and decryption of received messages, by calling the functions PrepForDecode() and originaltext().

**Main Code Structure:**



**Code:**

const String keys[] = {

"TTZUZRZJHP", "SUENIFKGAT", "CEXGWKMCVQ", "SIACAKRFM", "CSMKSWSNLH",

"IPIJWROJBB", "SZSGAMYJGJ", "BIHLBNUIJS", "UTFGSQAISS", "SSPXJXURJV",

"XPDRYPOFAC", "DROIFANPZE", "DKTZXUQLAM", "UDHAWMAJTG", "HKTQMEPWXS",

"IRUQAHZSBP", "ZHZAAGTMTN", "EHORTISSLQ", "NYZZQALWTP", "PEEGDJLQLQ",

"UNQGHTXFSG", "NOBSTVFLAX", "GQJYEVUNCX", "UTJDNOCLSP", "XQNGBDOBEN",

"FURHLIZSIN", "AAAPEIGRHI", "QWQFCCVFPV", "KZEFJHUJWB", "NDJNLHURJO"

};

char readyToSend[32] = {'\0'}; //global array, which is to be transmitted via RF24

char recieved[32] = {'\0'}; // global array, which is to be recieved via RF24

int keyRecieved = 0; //global key number, which is to be embedded as the last char recieved[]

String loadInput() { //this functions takes the input string and checks it's validity

bool errorFlag = false;

String(inputStr);

do {

errorFlag = false;

while (Serial.available () == 0) {}

inputStr = Serial.readString();

inputStr.remove((inputStr.length() - 2), 2);

int inputLength = inputStr.length();

if (inputLength > 30) {

Serial.println("Input longer than 30 chars.");

errorFlag = true;

}

} while (errorFlag);

return inputStr;

}

String cipherText(String str, const String key) //this function encrypts the message, and returns the encrypted version

{

String cipher\_text;

for (int i = 0; i < str.length(); i++)

{

if (isWhitespace(str[i])) {

cipher\_text += ' ';

}

else {

char x = ((int)(str[i] - 33) + (int)(key[i % key.length()] - 33)) % 94;

x += 33;

cipher\_text += x;

}

}

return cipher\_text;

}

String originalText( String cipher\_text, const String key) //this function decrypts the message, and returns the decrypted version

{

String orig\_text;

for (int i = 0 ; i < cipher\_text.length(); i++)

{

if (isWhitespace(cipher\_text[i])) {

orig\_text += ' '; }

else {

// converting in range 0-94, then applying reverse cipher

char x = ((cipher\_text[i] - 33) - (key[i % key.length()] - 33) + 94) % 94;

// convert into (ASCII)

x += 33;

orig\_text += x; }

}

return orig\_text;

}

void prepForTransmit(String input, int keyNumber) {

input.toCharArray(readyToSend, 32);

int zeroesStart = input.length();

char keyNumberAsChar = (char)(keyNumber + 65);

readyToSend[30] = keyNumberAsChar;

return;

}

String PrepForDecode() { //this function makes extracts key, then removes padding

keyRecieved = (recieved[30] - 65);

String toEdit = String(recieved);

toEdit.remove(30, 1);

if (toEdit.indexOf('\0') != 31) {

toEdit.remove(toEdit.indexOf('\0'), (30 - toEdit.indexOf('\0')));

}

else { }

return toEdit;

}

void Transmit() {

randomSeed(analogRead(A0)); //SEED RANDOM FUNCTION

delay(100);

String(text) = loadInput(); //GET INPUT

Serial.print("You entered: ");

Serial.println(text); //PRINT INPUT <--------------------------------------

int keynumber = random(25); // |

String encrypted\_message = cipherText( text, keys[keynumber]); //ENCRYPT THIs \_|

Serial.print("Using key number ");

Serial.println(keynumber);

prepForTransmit(encrypted\_message, keynumber); // PADDING AND ADD KEYNUMBER

Serial.print("Encrypted Message is: ");

Serial.println(readyToSend);

for (int i = 0; i < 32; i++) {

Serial1.write(readyToSend[i]);

}

Serial1.flush();

Serial.println("Encrypted Message has been transmitted");

Serial.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

return;

}

void Recieve() {

Serial.println("A message has been recieved!");

delay(100);

Serial1.readBytes(recieved, sizeof(recieved));

Serial.print("Encrypted Message recieved: ");

Serial.println(recieved);

String toDecode = PrepForDecode(); // PREP RECIEVED STRING FOR DECODING (REMOVE PADDING, EXTRACT KEY)

Serial.print("Deciphered Message:");

Serial.println(originalText( toDecode , keys[keyRecieved])); // PRINT DECODED STRING

Serial.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

return;

}

void setup() {

Serial.begin(9600); // init serial comms

Serial1.begin(9600);

Serial.println("To transmit a coded message, simply enter your message. Maximum length is 30 chars");

Serial.println("If you recieve a message, you will be prompted immediately.");

Serial.println("\nDO NOT ENTER A MESSAGE IF A MESSAGE IS BEING RECIEVED\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

void loop() { //main program

while ((Serial.available () == 0) && (Serial1.available () == 0)) {}

if (Serial1.available()) {

Recieve();

}

else if (Serial.available()) {

Transmit();

}

}

**Radio Implementation:**

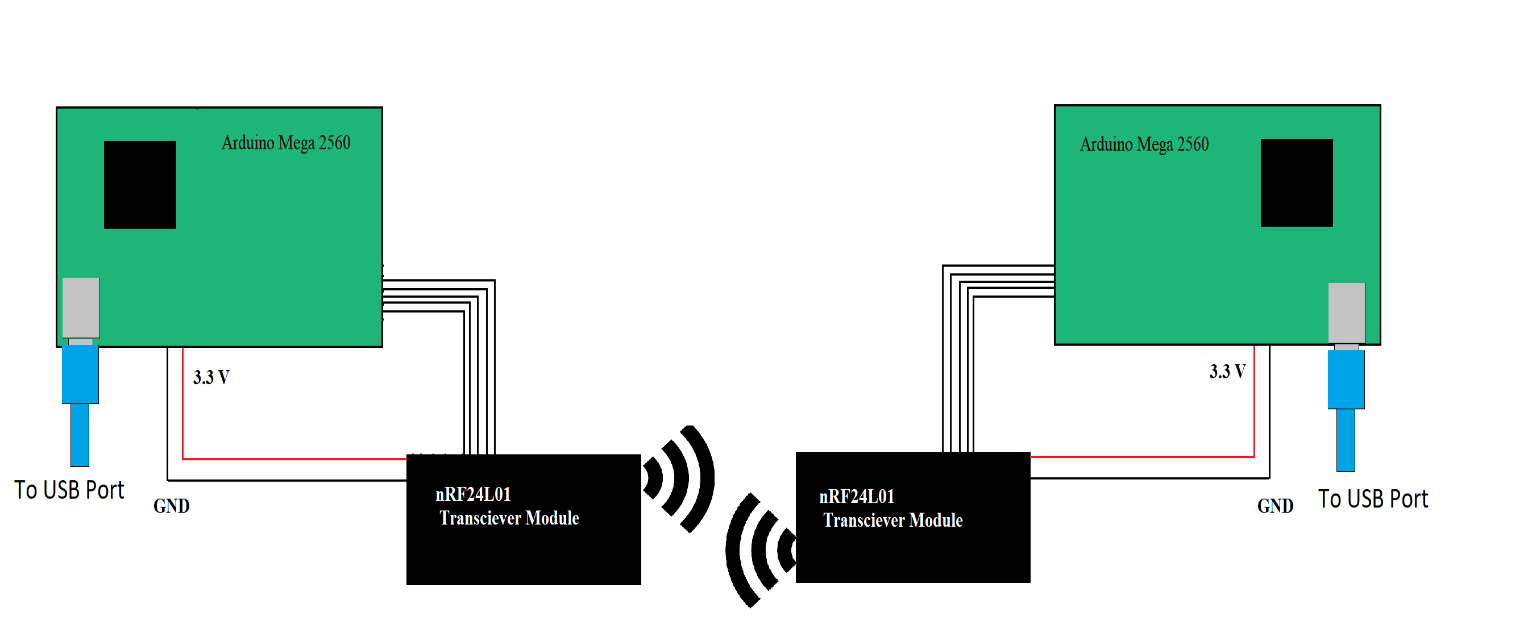
For radio communication, I used nRF24l01 transceiver modules, with the RF24 library developed by github user TMrh20.

These modules are half-duplex, operating on a 2.4 GHz band, and have a maximum data transfer rate of 2 Mbps. They use Enhanced ShockBurst protocol.

The modules communicate with the Arduino via SPI interface.

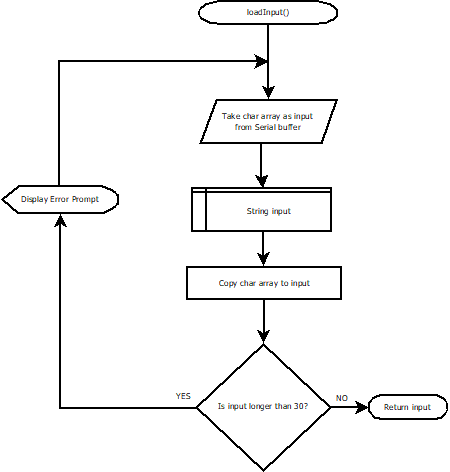
I had to attach a 10uF capacitor to each of the modules due to power draw issues.

**Schematic:**

****

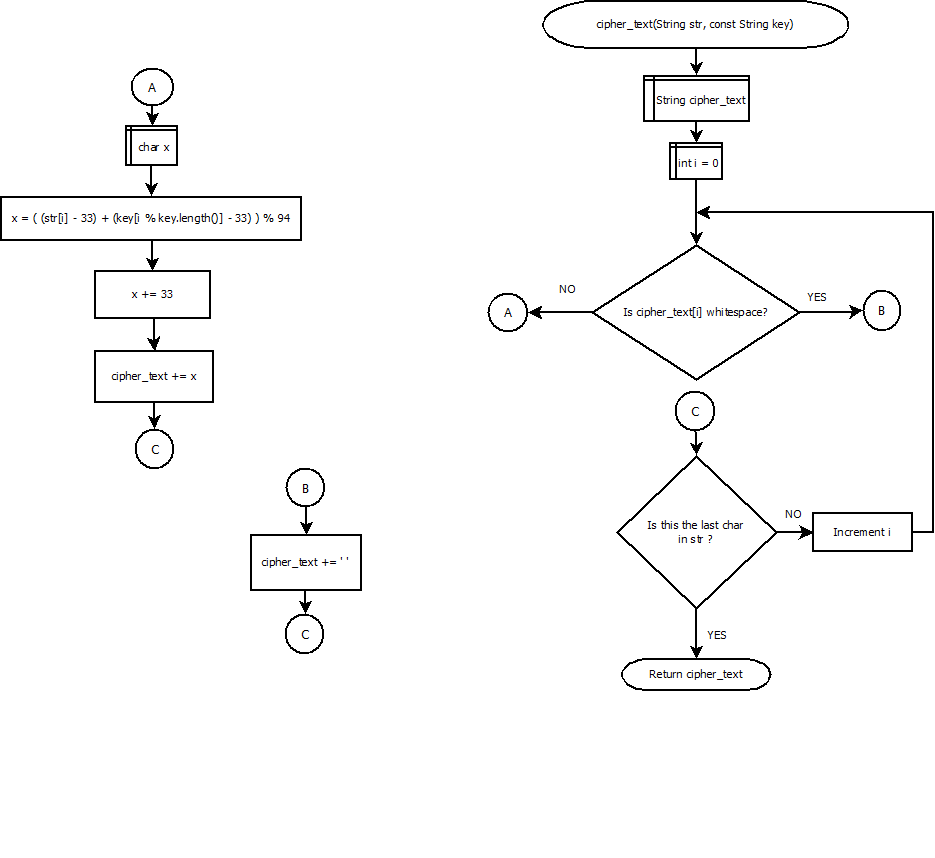
**Flowcharts:**

**Function loadInput():**

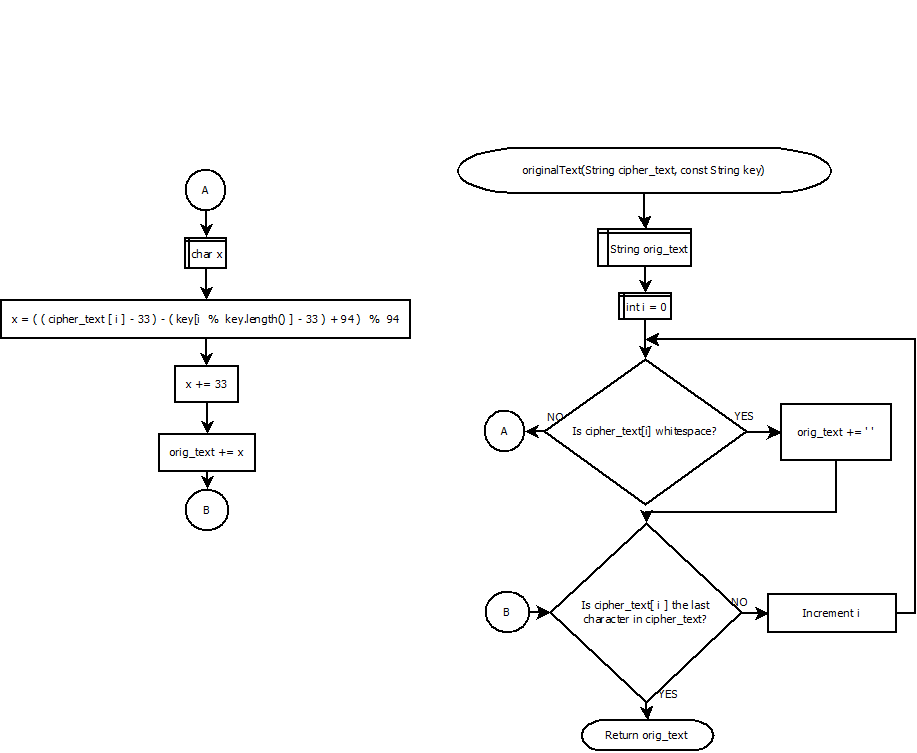


This function checks the length of the entered string. If it is not within acceptable bounds, it prompts the user to enter the string again.

**Function cipher\_text():**

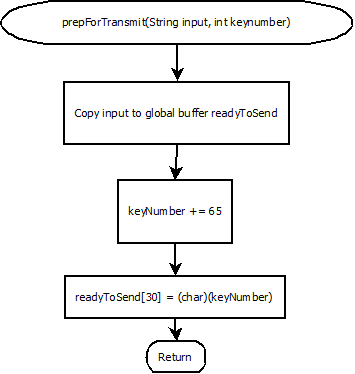
‘

This is the function responsible for encrypting the message. Through the use of a key (used as a circular buffer), which is stored in a global array, it encrypts the message. It ignores whitespaces, and includes them as is in the encrypted message.

**Function originalText():**

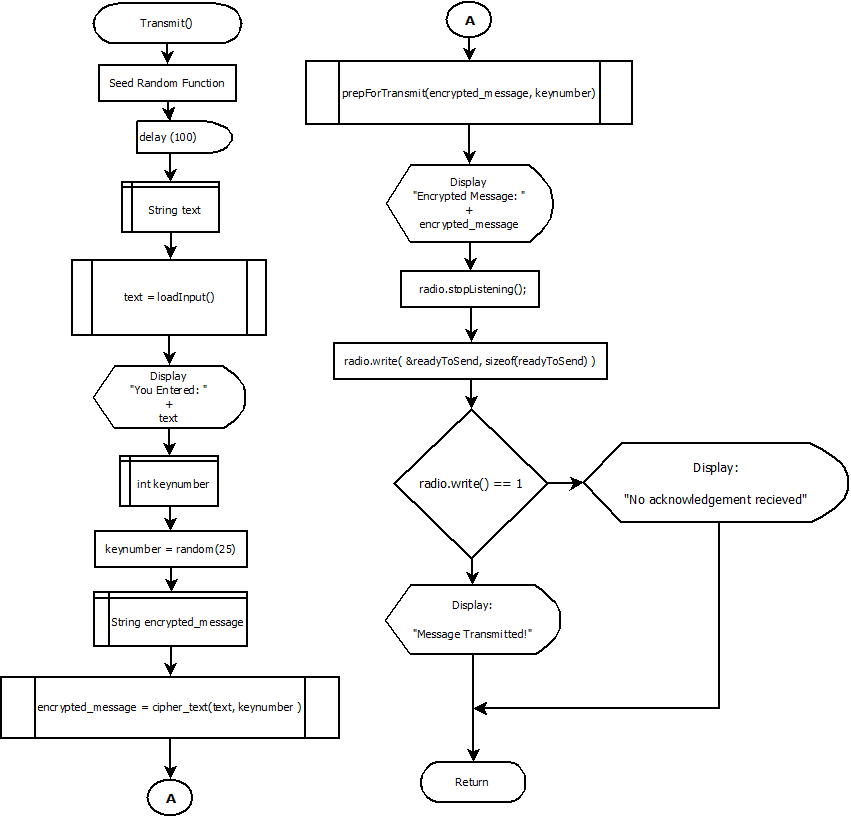
This function is responsible for deciphering a received message. It also uses a stored key, similar to the function for encryption.

**Function prepForTransmit():**



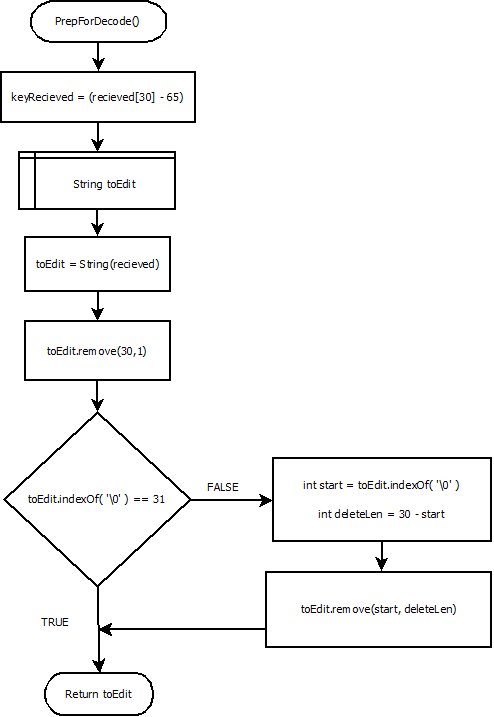
This function appends the index of the chosen key in the global key array to the encrypted message.

**Function Transmit():**



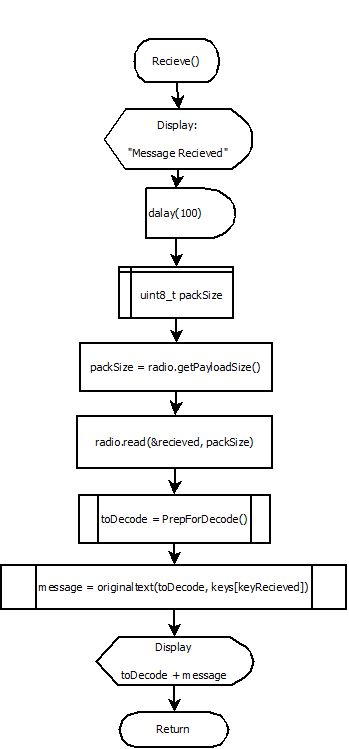
This function is responsible for taking input from the user, and calling the functions loadInput(), cipher\_text(), and prepForTransmit(). It then transmits the message over radio.

**Function PrepForDecode():**



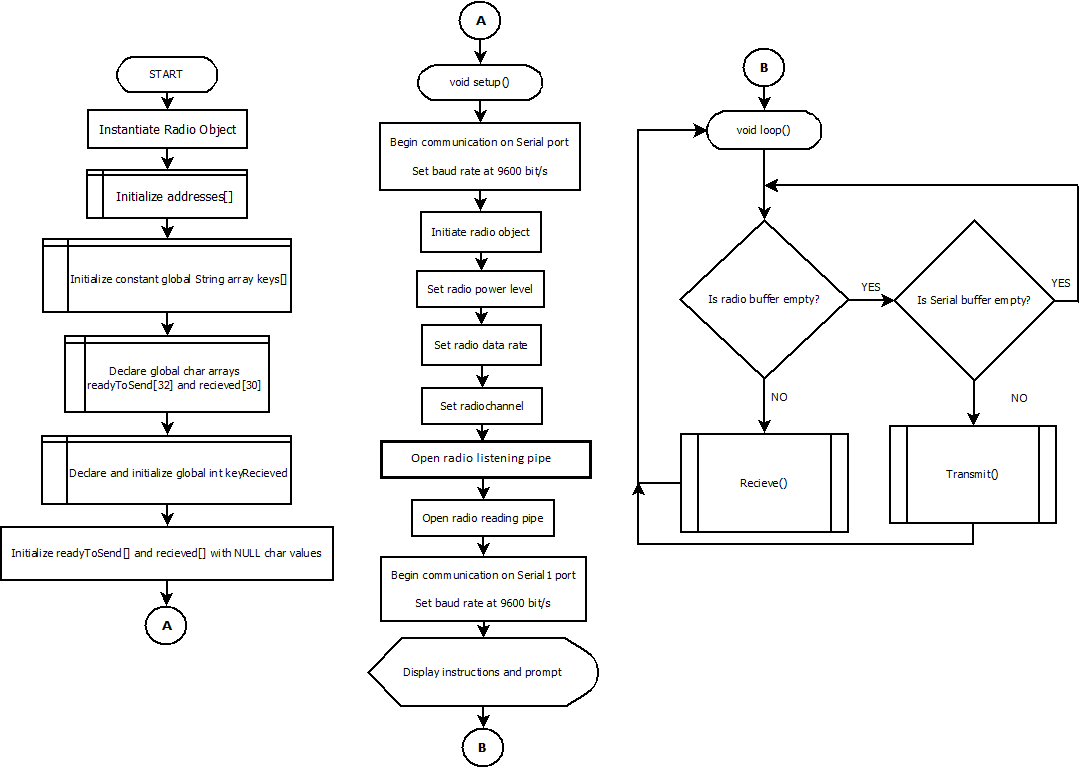
This function removes the key index number from the received message, and stores it in a global variable. It also prunes the null characters from the received message, making it ready for decryption.

**Function Receive():**



This function recieves the message from RF24 FIFO buffers,and performs preparation and decryption of received messages, by calling the functions PrepForDecode() and originaltext().

**Main Code Structure:**



**Main Code:**

#include "Arduino.h"

#include <SPI.h>

#include <RF24.h>

RF24 radio(48, 49); //radio object created, with CE => 48 and CSN => 49

byte addresses[][6] = {"1Node", "2Node"};

const String keys[] = {

"TTZUZRZJHP", "SUENIFKGAT", "CEXGWKMCVQ", "SIACAKRFM", "CSMKSWSNLH",

"IPIJWROJBB", "SZSGAMYJGJ", "BIHLBNUIJS", "UTFGSQAISS", "SSPXJXURJV",

"XPDRYPOFAC", "DROIFANPZE", "DKTZXUQLAM", "UDHAWMAJTG", "HKTQMEPWXS",

"IRUQAHZSBP", "ZHZAAGTMTN", "EHORTISSLQ", "NYZZQALWTP", "PEEGDJLQLQ",

"UNQGHTXFSG", "NOBSTVFLAX", "GQJYEVUNCX", "UTJDNOCLSP", "XQNGBDOBEN",

"FURHLIZSIN", "AAAPEIGRHI", "QWQFCCVFPV", "KZEFJHUJWB", "NDJNLHURJO"

};

char readyToSend[32] = {'\0'}; //global array, which is to be transmitted via RF24

char recieved[32] = {'\0'}; // global array, which is to be recieved via RF24

int keyRecieved = 0; //global key number, which is to be embedded as the last char recieved[]

String loadInput() { //this functions takes the input string and checks it's validity

bool errorFlag = false;

String(inputStr);

do {

errorFlag = false;

while (Serial.available () == 0) {}

inputStr = Serial.readString();

inputStr.remove((inputStr.length() - 2), 2);

int inputLength = inputStr.length();

if (inputLength > 30) {

Serial.println("Input longer than 30 chars.");

errorFlag = true;

}

} while (errorFlag);

return inputStr;

}

String cipherText(String str, const String key) //this function encrypts the message, and returns the encrypted version

{

String cipher\_text;

for (int i = 0; i < str.length(); i++)

{

if (isWhitespace(str[i])) {

cipher\_text += ' ';

}

else {

char x = ((int)(str[i] - 33) + (int)(key[i % key.length()] - 33)) % 94;

x += 33;

cipher\_text += x;

}

}

return cipher\_text;

}

String originalText( String cipher\_text, const String key) //this function decrypts the message, and returns the decrypted version

{

String orig\_text;

for (int i = 0 ; i < cipher\_text.length(); i++)

{

if (isWhitespace(cipher\_text[i])) {

orig\_text += ' '; }

else {

// converting in range 0-94, then applying reverse cipher

char x = ((cipher\_text[i] - 33) - (key[i % key.length()] - 33) + 94) % 94;

// convert into (ASCII)

x += 33;

orig\_text += x; }

}

return orig\_text;

}

void prepForTransmit(String input, int keyNumber) {

input.toCharArray(readyToSend, 32);

int zeroesStart = input.length();

char keyNumberAsChar = (char)(keyNumber + 65);

readyToSend[30] = keyNumberAsChar;

return;

}

String PrepForDecode() { //this function makes extracts key, then removes padding

keyRecieved = (recieved[30] - 65);

String toEdit = String(recieved);

toEdit.remove(30, 1);

if (toEdit.indexOf('\0') != 31) {

toEdit.remove(toEdit.indexOf('\0'), (30 - toEdit.indexOf('\0')));

}

else { }

return toEdit;

}

void Transmit() {

randomSeed(analogRead(A0)); //SEED RANDOM FUNCTION

delay(100);

String(text) = loadInput(); //GET INPUT

Serial.print("You entered: ");

Serial.println(text); //PRINT INPUT

int keynumber = random(25);

String encrypted\_message = cipherText( text, keys[keynumber]); //ENCRYPT THIs \_|

Serial.print("Using key number ");

Serial.println(keynumber);

prepForTransmit(encrypted\_message, keynumber); // PADDING AND ADD KEYNUMBER

Serial.print("Encrypted Message is: ");

Serial.println(readyToSend);

radio.stopListening();

if (!radio.write( &readyToSend, sizeof(readyToSend) )) {

Serial.println("No acknowledgement of transmission.");

}

Serial.println("Encrypted Message has been transmitted");

radio.startListening();

Serial.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

return;

}

void Recieve() {

Serial.println("A message has been recieved!");

delay(100);

uint8\_t packSize = radio.getPayloadSize();

radio.read(&recieved, packSize); // get packet

Serial.print("Encrypted Message recieved: ");

Serial.println(recieved);

String toDecode = PrepForDecode(); // PREP RECIEVED STRING FOR DECODING (REMOVE PADDING, EXTRACT KEY)

Serial.print("Deciphered Message:");

Serial.println(originalText( toDecode , keys[keyRecieved])); // PRINT DECODED STRING

Serial.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

return;

}

void setup() {

Serial.begin(9600);

radio.begin(); // Initiate the radio object

radio.setPALevel(RF24\_PA\_MIN); // Set the transmit power to lowest available to prevent power supply related issues

radio.setDataRate(RF24\_2MBPS); // Set the speed of the transmission to the quickest available

radio.setChannel(124); // Use a channel unlikely to be used by Wifi, Microwave ovens etc

// Open a writing and reading pipe on each radio, with opposite addresses

radio.openWritingPipe(addresses[1]);

radio.openReadingPipe(1, addresses[0]);

Serial.println("To transmit a coded message, simply enter your message. Maximum length is 30 chars");

Serial.println("If you recieve a message, you will be prompted immediately.");

Serial.println("\nDO NOT ENTER A MESSAGE IF A MESSAGE IS BEING RECIEVED\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

void loop() { //main program

while (!(radio.available()) && !(Serial.available())) {}

if (radio.available()) {

Recieve();

}

else if (Serial.available()) {

Transmit();

}

}