Centro Regional Universitario Córdoba IUA
Universidad de la Defensa Nacional

Securización de Sistema de Mando a Distancia con RF

Rodrigo Javier Fernández

Trabajo Final

2018

INDICE

1. Introducción

Desde hace varios años los controles remotos por radio frecuencia han sido ampliamente utilizados para controlar dispositivos de uso cotidiano, como alarmas hogareñas, alarmas de automóviles con cierres centralizados o portones eléctricos. Su bajo costo, sencilla implementación y uso los convierten en una opción atractiva para fabricantes que requieren de un mando a distancia para sus dispositivos. Estos controles ofrecen gran comodidad a quienes los utilizan, pero ocultan un potencial problema del que la mayoría no son conscientes, la seguridad. Debemos recordar que la mayoría de estos dispositivos funcionan en cierta forma como llaves de acceso a vehículos, a distintas instalaciones o a nuestros propios hogares. En nuestra cotidianidad donde la inseguridad es una realidad, con delincuentes que cada vez utilizan más la tecnología, no podemos darnos el lujo de dejar desprotegidos estos dispositivos. La mayoría de estos controles son vulnerables a ataques de reproducción [1] que pueden ser realizados muy fácilmente sin demasiados conocimientos previos o hasta se pueden conseguir muy fácilmente y a bajo costo online dispositivos clonadores de controles. Los que no están incluidos en ese grupo son más seguros y costosos, pero también tienen una vulnerabilidad [2] reportada que puede ser explotada con un requerimiento económico mínimo, que puede estar al alcance de cualquier persona.

2. Problema

Los controles remotos por radio frecuencia en su mayoría envía una señal, un código por cada comando que le envían al dispositivo a controlar. Este código no solamente no posee la longitud necesaria para ser considerado seguro, 24 bits por ejemplo, sino que además se repite cada vez que se desea realizar la misma función. Esta técnica es aún utilizada, dado su simpleza y la posibilidad de implementarla utilizando controladores muy económicos. Esto hace de estos controles un blanco inmejorable para ataques de reproducción, o ataques de fuerza bruta [3]. En los ataques de reproducción, un código válido es enviado por un atacante y se toma como válido, ante la imposibilidad del receptor del verificar el origen de este. Al enviar el mismo código siempre, no tiene sentido codificar el

mensaje ya que atacante no debe entenderlo, solo deberá reproducirlo sin que el receptor puede distinguir desde que dispositivo fue enviado. Al no tener una longitud de clave adecuada, estos dispositivos pueden ser fácilmente blanco de ataques de fuerza bruta en donde en cuestión de pocos minutos pueden ser vulnerados. Utilizando una técnica más avanzada como la utilización de la secuencia de De Bruijn [4], que en caso de que no se utilice un preámbulo que delimite la trama a recibir, se puede acortar el espacio de claves logrando así romper el código en pocos segundos [5]. Como alternativa a este problema se desarrolló una técnica alternativa, Rolling code [6] [7], que consiste en enviar un código distinto conocido por el receptor y el emisor cada vez. De esta forma se logra no enviar siempre el mismo código, solucionando el problema anterior. Esta alternativa fue la seleccionada por lo fabricantes de automóviles para sus mandos a distancia para abrir y cerrar un vehiculó o inclusive encenderlo. Sin embargo, nada es invulnerable y en 2015 se demostró que Rolling code es susceptible a un ataque del tipo man in the middle [8]. En donde el atacante transmite ruido al vehículo en una frecuencia apenas superior a que utiliza el emisor. El receptor escucha en un rango de frecuencia un apenas mayor a la que utiliza el emisor. De este modo logra que el receptor escuche el código que envía el emisor y el ruido que el emitió, logrando así que no pueda interpretar el código enviado por el emisor. Mientras tanto en atacante escucha en la una banda de frecuencia apenas más chica que la del receptor y allí puede no ser afectado por el ruido que él mismo emite. Con esto logra que el receptor no reciba el código que envía el emisor mientras que el atacante si lo hace y lo guarda. Al no realizar ninguna acción el vehículo, el usuario vuelve a presionar el botón del mando, así el emisor envía el siguiente código, el cual el atacante guarda para luego transmitir el primer código que envió. De esta forma las puertas se destraban sin que el emisor note que el atacante tiene en su poder el próximo código válido. Este ataque explota la vulnerabilidad de que las claves válidas no tienen caducidad y así pueden ser utilizadas por el atacante en cualquier momento.

3. Objeto de Estudio

El objeto de estudio de este trabajo es un sistema de mando a distancia que de forma segura controle algún dispositivo electrónico como un portón o una cerradura eléctrica. El mismo está compuesto de un emisor y receptor. Los dispositivos mencionados deben ser construidos con hardware genérico de fácil

acceso en el mercado local y bajo costo, que implemente soluciones a las vulnerabilidades descriptas anteriormente.

4. Campo de Acción

El campo de acción se encuentra limitado a un prototipo para el ámbito doméstico, esperándose que este prototipo sea una primera aproximación a posible un modelo comercial.

5. Objetivo General

El objetivo de este trabajo es el desarrollo de un sistema de mando a distancia seguro, construido con hardware genérico, resistente aún a ataques realizados con equipos con alto poder de procesamiento como computadoras personales. Se pretende así construir un control remoto y su correspondiente receptor. Ambos dispositivos deberán ser económicos y de fácil construcción. El control remoto a construir tendrá dos botones para enviar dos comandos. Ambos comandos estarán acompañados de una clave independiente entre los botones, que se calculará en base a indice independiente correspondiente al botón mencionado que se cambiará con cada pulsación. El receptor debe ser capaz de trabajar con múltiples controles remotos. Se requiere también que el sistema implementado mitigue las vulnerabilidades y resista los ataques mencionados en la Sección 2.

a. Objetivos Específicos

- Se deben utilizar claves de 256 bits de longitud.
- Se debe contar con algún mecanismo para evitar ataques del tipo man in the middle y replay.
- Se debe poder bloquear ataques del tipo de fuerza fruta.
- Se deberá utilizar únicamente hardware genérico, accesible y económico.
- El emisor y el receptor deben almacenar los índices correspondientes a cada botón de manera persistente.
- El receptor debe poder trabajar con 45 controles distintos.
- El control debe tener un alcance de al menos 20 mts.

6. Delimitación del proyecto

El proyecto está limitado a construcción de un prototipo de control remoto y receptor que permita observar su funcionamiento y como alcanza los objetivos planteados. El receptor no controlará ningún dispositivo, solo se limitará a demostrar el correcto procesamiento de los códigos recibidos. No se construirá una versión final.

7. Desarrollo

A. Hardware Utilizado

Para la construcción de nuestro emisor y receptor se utilizara el siguiente hardware: Arduino Pro Mini [9] x 2, Kit Modulo Rf Emisor Y Receptor 433mhz [10], Adaptador Conversor USB TTL Serie, Fuente Para Protoboard. La placa Arduino Pro-Mini que se utilizará para el emisor cuenta con una memoria EEPROM de 1 Kbyte (se necesitan 16 bytes para guardar los 2 índices). El receptor también se construirá con un Arduino Pro-Mini que cuenta con 1 Kbyte, suficiente para almacenar los 720 bytes necesarios para almacenar los índices de 45 controles remotos. Se destaca el tamaño de la memoria ROM programable dado que es necesaria para guardar los datos de forma persistente aún si se interrumpe la alimentación de energía. Cabe recordar que las memorias EEPROM están garantizadas para realizar 100.000 escrituras, a un promedio de 4 pulsaciones por día, duraría al menos 68 años. Tanto el emisor como el receptor poseen un microcontrolador con una velocidad de 16 MHz, capaz de realizar el proceso criptográfico necesario en 80 milisegundos. Los módulos Emisor y Receptor RF 433mhz poseen un alcance mínimo de 20 mts y máximo de 200.

En la figura 1, podemos observar el emisor. En la imagen se ve una fuente de alimentación de 5v, un pulsador para cada comando, un módulo transmisor RF de 433 MHz, una placa Arduino Pro-Mini de 5v.

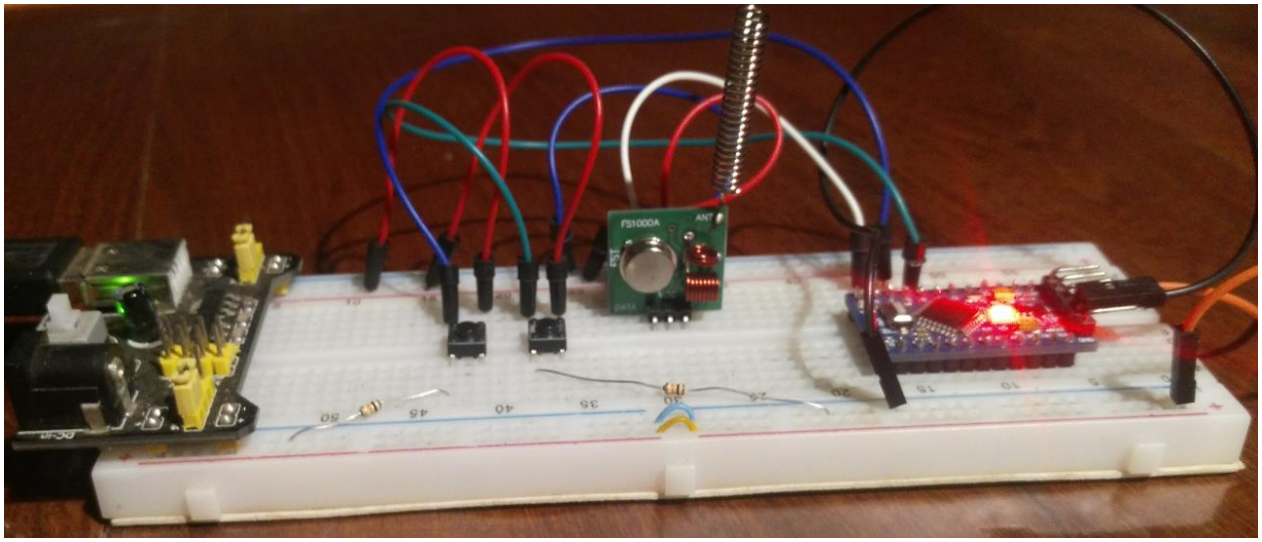

Fig. 1 Emisor

El receptor que se muestra en la figura 2, es un tanto más sencillo. Consta de una placa Arduino Pro-Mini, una fuente de alimentación de 5v y un módulo receptor RF de 433 MHz. Esta placa consta con un microprocesador con memorias ROM y RAM, suficiente para almacenar claves suficientes para 45 controles distintos. Se eligió esta placa para el receptor dado su tamaño pequeño, pero se puede optar por alguna otra placa más grande cuyo procesador puede trabajar con 250 controles.

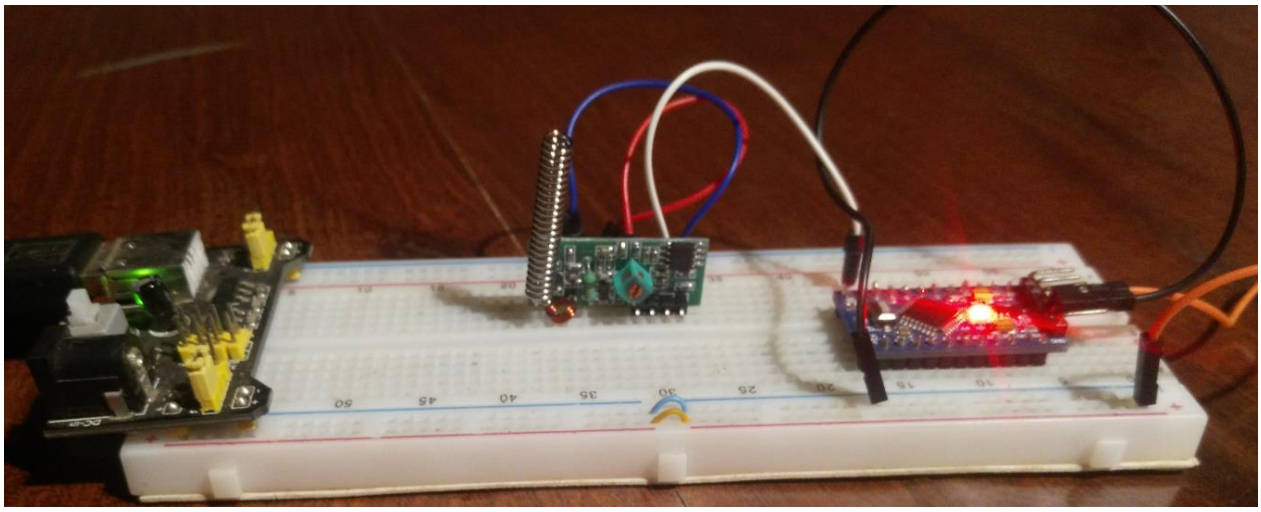

Fig. 2 Receptor

B. Entorno de Desarrollo y Librerías

Para el desarrollo del software se utilizará Arduino IDE 1.8.4, un popular entorno de desarrollo que utiliza un lenguaje similar a C. Se utilizará la librería VirtualWire para el control de los módulos RF, la misma permite trasmitir hasta 216 bits por trama. Dado que el tamaño de la clave es de 256 bits y que puedo trasmitir hasta 216 bit por trama, deberé transmitir 2 tramas por cada comando que desee enviar. Para las operaciones criptográficas se utilizará la librería sha256.h la cual calcula un HMAC-SHA256 con una clave de 256 en 80 milisegundos corriendo en los procesadores ATmega328.

C. Estructura de Trama

Con esta información puedo proceder a diseñar la trama que utilizaré para enviar los comandos. Primero seleccioné el tipo de dato para el índice de cada botón, unsigned long de 32 bits, esto me permite disponer de 2^32 números posibles antes de que empiecen a repetirse la clave enviada. Utilizaré 8 bits para el id del emisor, otro para el comando a ejecutar, uno para la secuencia de trama dado que transmitiré dos tramas por comando. Finalmente, la trama estará compuesta por los 128 bits correspondientes a una de las mitades de la clave a transmitir. De esta forma nuestra trama tendrá 184 bits, por debajo de los 216 bits que puede transmitir por trama la librería que controla el transmisor/receptor RF.

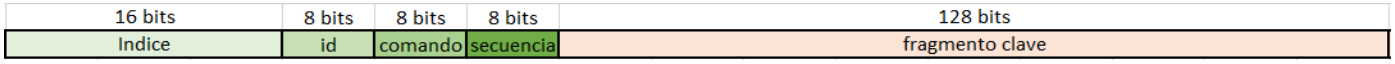

Fig 3 Estructura de la trama

D. Aspectos de Seguridad y Operativos

El próximo paso en definir el mecanismo criptográfico que utilizaré para poder generar una clave de 256 bits de forma segura. Utilizaré la función hash SHA256 que provee la longitud de clave deseada. Para proveer mayor protección contra ataques de extensión de longitud la función hash se implementará junto con HMAC. Se probó que los procesadores seleccionados tardaron 80 milisegundos en procesar un mensaje de 4 bytes junto a una clave de 256 bits usando HMAC-SHA256. El próximo punto a definir es como proteger a mi sistema de un ataque de un ataque del tipo hombre en el medio [2], la solución sería que los códigos

generados por el emisor tuvieran una caducidad, de modo que solo fueran válidos por un período breve luego de ser generados. En este caso por una cuestión de mantener el prototipo con el menor tamaño posible y además por el costo, el emisor usa solo un emisor RF. Por lo tanto, solo es posible una comunicación unidireccional desde el emisor al receptor. Esto imposibilita cualquier sincronización entre ambos dispositivos y así la viabilidad de poder validar la caducidad de los mensajes. Si bien no puedo resolver el problema de fondo, si puedo mitigar el ataque descripto. En mi control tendré dos botones, uno para abrir y otro para cerrar. Cada uno de estos comandos tiene un índice que se incrementa en una unidad al presionar botón asociado al mismo. Si bien no puedo sincronizar la hora entre el emisor y el receptor, puedo medir el paso del tiempo en ambos dispositivos. En el emisor desde que presiono un botón y en el emisor desde que recibo un código de autenticación válido. Esto me permite que después de un tiempo determinado, en este caso seleccione 5 segundos, incrementar el valor del índice del botón enviado/comando recibido. Elegí 105 como valor de incremento, suficientemente grande e impar, dado que la cantidad número máximo del índice es par y cuando se da la vuelta no se repiten los números que ya se utilizaron. De este modo, se invalidarían las claves anteriores, si alguna hubiera sido interceptada y almacenada por un atacante con un ataque del tipo man in the middle [8] . De este modo, si un atacante de apoderará de la siguiente clave válida para abrir como se explicó en la sección 2, luego de 5 segundos después de ser aceptada la última, se invalidarán las próximas 105 incluida la interceptada. Para asegurarme de que esto sea así, incrementaré el índice en un número suficientemente grande para que, en caso de un ataque, invalidar las claves interceptadas aún si el usuario oprima el botón varias veces.

Finalmente, para bloquear ataques de fuerza bruta, contaré los pares de tramas cuyos encabezados sean válidos pero las claves no. Después de la décima solicitud inválida recibida, el receptor esperará un segundo antes de procesar alguna otra solicitud.

Dado que solo hay comunicación unidireccional, se puede perder fácilmente la sincronización entre los índices de un botón de ambos dispositivos. Para solucionar este problema, el receptor deberá aceptar como válidos un número razonable de índices siguientes al que él tiene registrado. El número que elegí es 10500, 100 veces el incremento del índice del otro botón que se presionó.

E. Arquitectura

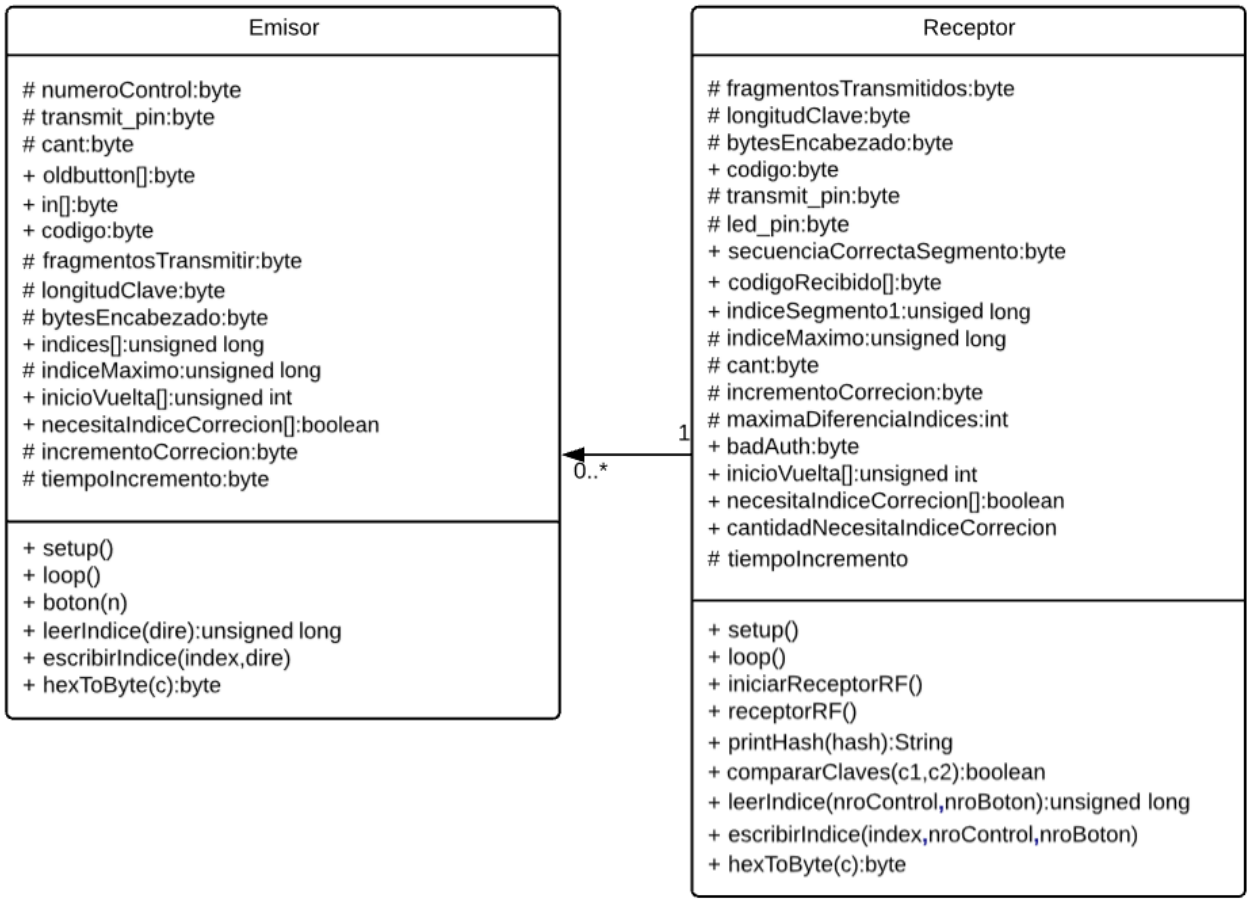

Fig. 4 Diagrama de Clases

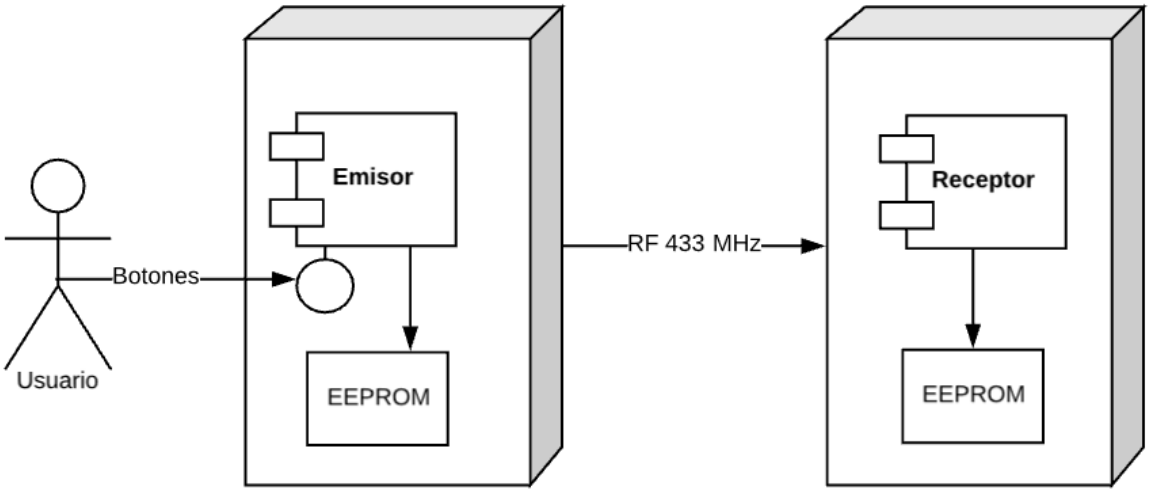

Fig. 5 Diagrama de Despliegue

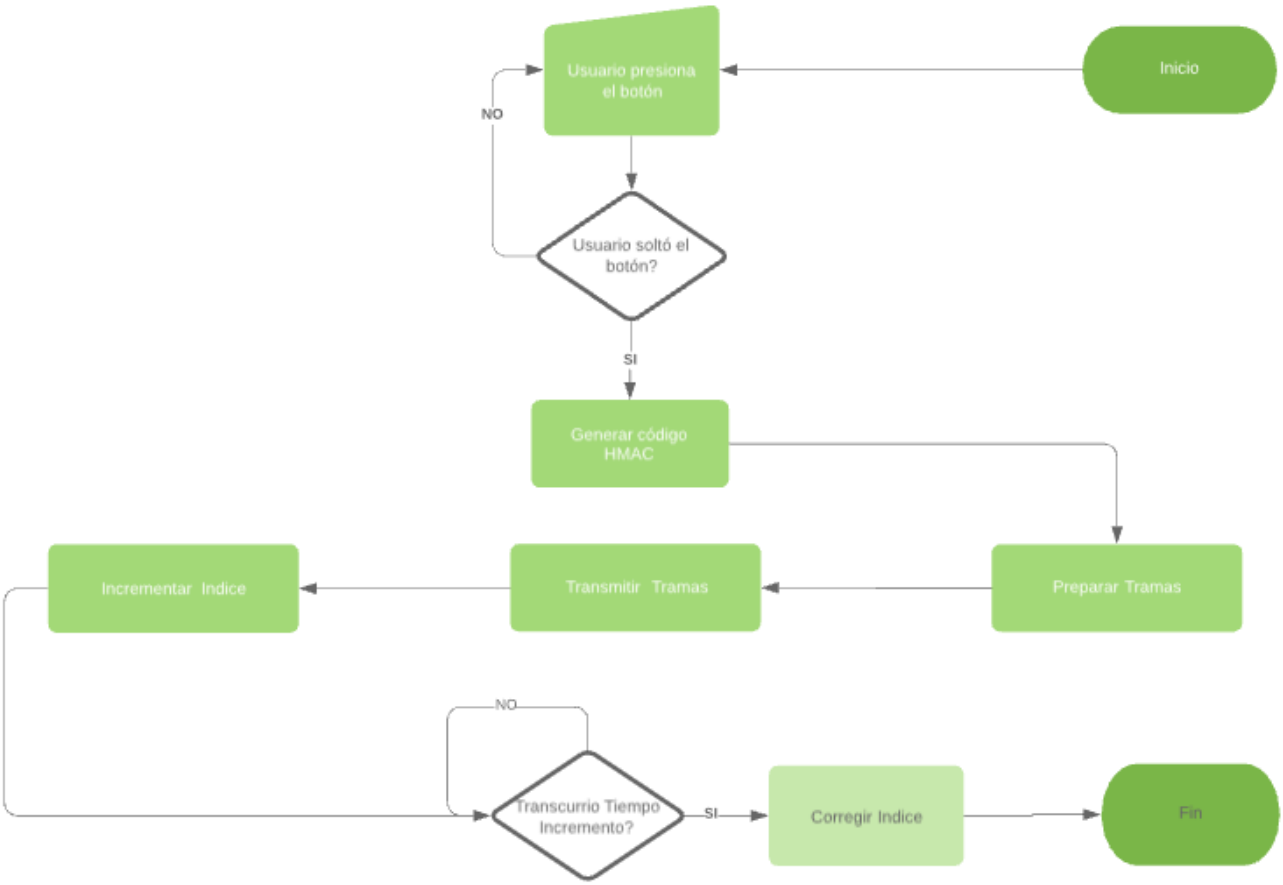

Fig. 6 Diagrama Flujo Emisor

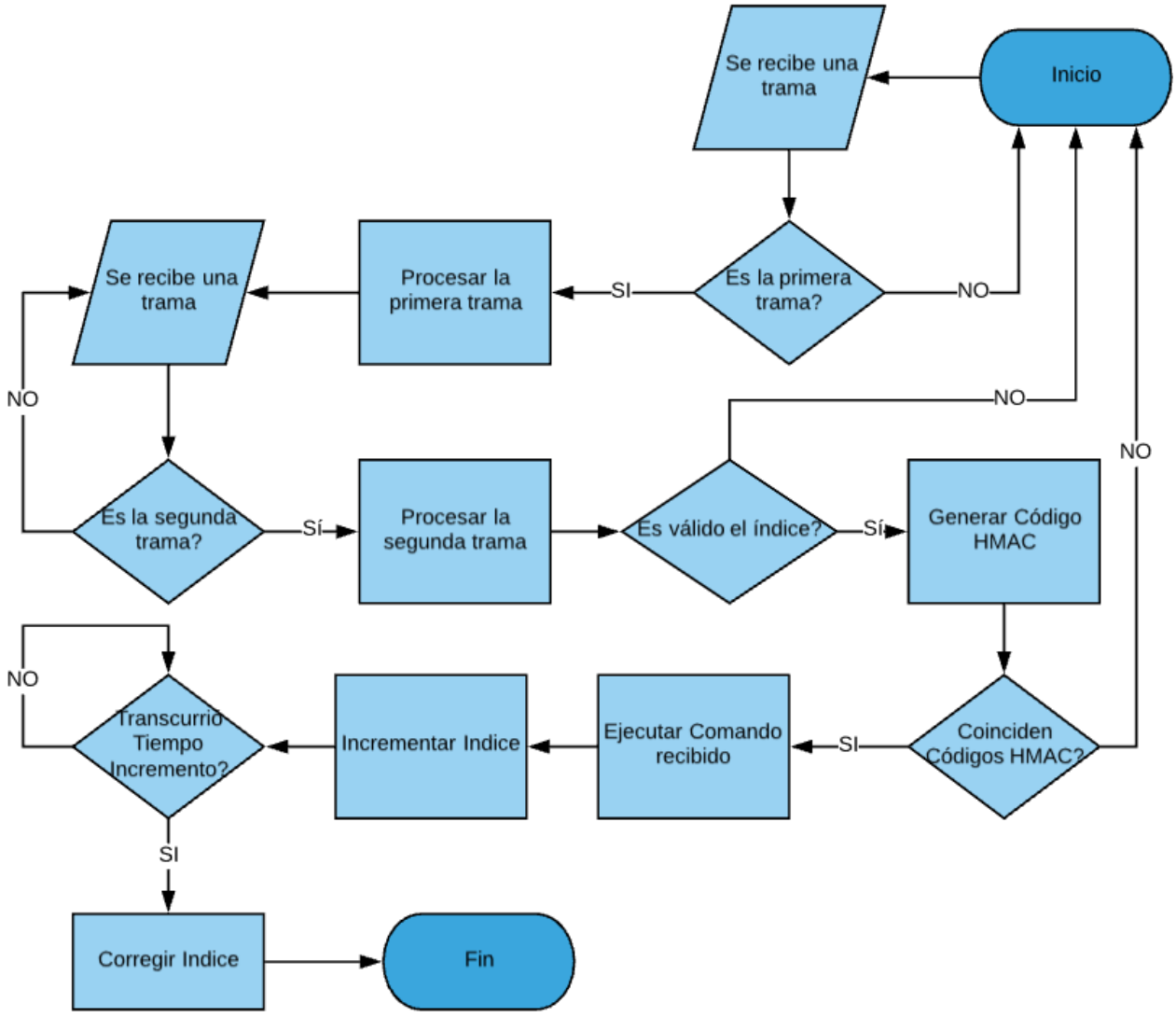

Fig. 7 Diagrama de Flujo Receptor

F. Código Fuente

Código Emisor

```
#include <EEPROM.h>
#include <avr/pgmspace.h>
#include <VirtualWire.h>
#include <sha256.h>
#include <Arduino.h>
#include <printf.h>

const byte numeroControl=45;
```

```
const byte transmit_pin = 8;
const byte  cant=2; //cantidad de botones del control, deben ser múltiplo de 2 .
Los botones pares activan y los impares desactivan
byte button[cant];
byte oldbutton[cant];
byte in[cant];
uint8_t* codigo="{FXH*Ua@EJ3{GHGTK(H_JfP+zUfvG?xT";
const byte fragmentosTransmitir=2;
const byte longitudClave=(32/fragmentosTransmitir); // en bytes busco una clave de
256 bits
const byte bytesEncabezado=7;  //son 7 los primeros bytes que uso para indicar
numero de índice (4), dispositivo (1) y comando a ejecutar (1), secuencia se
segmento transmitido (1)
unsigned long índices[cant]; // índices para botones del control, numero de comando
coincide con numero de casillero
const unsigned long indiceMaximo=4294967294; //el numero máximo tiene que ser par y
el incremento de la corrección impar para que se usen todos los números
unsigned int inicioVuelta[cant]; //uso esta variables para medir el tiempo antes de
corregir con el incremento los índices
boolean necesitaIndiceCorrecion[cant]; //uso estas variables para saber si necesito
medir tiempo para realizar una corrección de índices
const byte incrementoCorrecion=105; //cantidad de unidades que se incrementara el
índice correspondiente luego que caduque la el ultimo índice
const byte tiempoIncremento=5; //cantidad de segundo después de las que se realizar
la corrección del índice del botón presionado


void setup()
{ Serial.begin(9600);////////////////////////////////////////////////
  Serial.println("Hola soy el emisor- Control Remoto");///////////////
  pinMode(13, OUTPUT); ////////////////////////////////////////////////
  printf_begin();
  //pines para los botones
  in[0]=5;  ////pin para cerrar
  in[1]=6;  ////pin para abrir
  for(byte i=0;i<=cant;i++)
     { pinMode(in[i],INPUT);
     }


  //inicio arrays de control
  for(byte i=0;i<cant;i++)
     { indices[i] = leerIndice(i);
       inicioVuelta[i]=millis();
       necesitaIndiceCorrecion[i]=false;
     }
  //Inicializo el módulo RF
  vw_set_tx_pin(transmit_pin);
  vw_setup(2000);       // Bits per sec  */
  //Inicializo el array de estados viejos de los botones
  for(byte i=0;i<=cant;i++) {oldbutton[i]=0;}
}

void loop()
{ for(byte i=0;i<=cant-1;i++)
      {  boton(i);
```

```
        if(necesitaIndiceCorrecion[i])
          { int vuelta=(int)(((unsigned int)millis()-inicioVuelta[i])/1000);
            if(vuelta >=tiempoIncremento)
              { if(indices[i] > indiceMaximo-incrementoCorrecion)
                  { indices[i]= indices[i]+incrementoCorrecion-indiceMaximo;
                  }
                else
                  { indices[i]=indices[i]+incrementoCorrecion;
                  }
                escribirIndice((unsigned long)indices[i],(int)i);
                Serial.print("Indice del boton ");//////////////////////////////
                Serial.print(i); ////////////////////////////////////////////////
                Serial.print(" despues de que caduco: ");//////////////////////
                Serial.print(indices[i]);//////////////////////////////////////
                Serial.println("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");/////////
                Serial.println("");/////////////////////////////////////////////
                necesitaIndiceCorrecion[i]=false;
              }
          }
      }
}


void boton(byte n)
 { byte pos=n;
   button[pos] = digitalRead(in[pos]);

   if(button[pos] && !oldbutton[pos]) // lo mismo que si (button == high &&
   oldbutton == low)
     { //se ha presionado un boton
      necesitaIndiceCorrecion[pos]=true;
      inicioVuelta[pos]=millis();
      //genero el codigo de autenticación con mi clave y la combinación de el
      índice correspondiente concatenado con el nroControl y nroBoton
      Sha256.initHmac((uint8_t*)codigo,longitudClave);
      Sha256.print((String)indices[pos]+"-"+(String)numeroControl+(String)pos);
      String hmac=printHash(Sha256.resultHmac()); //para verificar el hmac
      https://codebeautify.org/hmac-generator
      //el hmac que tengo obtuve esta HEX (64 caracteres) y necesito pasarlo a
      bytes (32 caracteres) para transmitirlo
      byte conv[longitudClave];
      for(byte i=0; i<longitudClave;i++)
         { conv[i]=  hexToByte(hmac[i*2])<<4 | hexToByte(hmac[i*2+1] );
         }
      //TRANSMITIR DESDE ACA
      Serial.println(""); ////////////////////////////////////
      Serial.println("Principio Transmision---------"); ////////
      for(byte j=0;j<fragmentosTransmitir;j++)
         { byte tempTransmitir[((longitudClave)/ fragmentosTransmitir)
+bytesEncabezado]; // 27 es la máxima longitud en bytes a transmitir para la
librería VirtualWire
           tempTransmitir[0] = (byte) indices[pos];
           tempTransmitir[1] = (byte) (indices[pos] >> 8);
           tempTransmitir[2] = (byte) (indices[pos] >> 16);
           tempTransmitir[3] = (byte) (indices[pos] >> 24);
           tempTransmitir[4]=numeroControl;
```

```
            tempTransmitir[5]= pos; // comando a ejecutar, se reserva para el futuro
            donde pueda haber muchos más comandos
            tempTransmitir[6]= j; //numero de secuencia del fragmento transmitido
            for(byte k=0;k<(longitudClave/fragmentosTransmitir);k++)
               { tempTransmitir [k+bytesEncabezado]= conv[k+((longitudClave
/fragmentosTransmitir)*j)];
               }
            vw_send((uint8_t *)tempTransmitir, ((longitudClave/fragmentosTransmitir)
+bytesEncabezado));
            vw_wait_tx();
            delay(100);
            for(byte i=0;i<((longitudClave/fragmentosTransmitir)+bytesEncabezado)
;i++)////////////////////////////////////
               { Serial.print(tempTransmitir[i]); ////////////
                 Serial.print(" "); ////////////////////////
               } ////////////////////////////////////////////
            Serial.println(""); ////////////////////////////
          }
       Serial.println("Fin Transmision---------"); /////////
       //TRANSMITIR HASTA ACA
       digitalWrite(13, LOW);////////////////////////////////
       Serial.print("Indice del boton ");
       Serial.print(pos);
       Serial.print(" presionado antes de la correcion:");////
       Serial.println(indices[pos]);//////////////////////////
       Serial.println("");////////////////////////////////////
       Serial.print("Codigo para hmac:");//////////////////////
       for(byte i=0;i<longitudClave;i++)//////////////////////
          { Serial.print((char)codigo[i]);////////////////////
          }//////////////////////////////////////////////////
       Serial.println("");////////////////////////////////////
       Serial.print("Entrada para hmac:");//////////////////////
      Serial.println((String)indices[pos]+ "-
      "+(String)numeroControl+(String)pos);/////////////////
      Serial.print("HMAC:");//////////////////////////////////
      Serial.println(hmac);/////////////////////////////////
      //incremento en índice en 1 para el boton presionado . sí me paso del limite
      empiezo de 0
       if((indices[pos] ==indiceMaximo))
         { indices[pos]=0;
         }
       else
         { indices[pos]++;
         }

       Serial.print("Indice boton presionado despues de la correcion:");///////////
       Serial.println(indices[pos]);////////////////////////////////////////////
       //guardo el índice del boton presionado (abrir/cerrar portón por ej.)
       escribirIndice((unsigned long)indices[pos],(int)pos);
       oldbutton[pos] = 1;
   }
 else if(!button[pos] && oldbutton[pos]) // lo mismo que si(button == low &&
oldbutton == high)
   { // se soltó el boton
     oldbutton[pos] = 0;
     delay(100);
```

```
    }
}

String printHash(uint8_t* hash) {
  int i;
  String resu;
  for (i=0; i<32; i++) {
    resu+=(String)("0123456789abcdef"[hash[i]>>4]);
    resu+=(String)("0123456789abcdef"[hash[i]&0xf]);
  }
  return resu;
}

unsigned long leerIndice(byte dire)
{ byte descompuesto[4];
  //la librería eeprom lee de a un byte, mi índice tiene 4 byte por lo que tengo
  que leer los bytes separándolos primero
  descompuesto[0]=EEPROM.read(dire*4+1); // el 4 es por la cantidad de bytes del
  índice a guardar que es unsigned long (4 bytes) y yo guardo de a un byte
  descompuesto[1]=EEPROM.read(dire*4+2); // el 4 es por la cantidad de bytes del
  índice a guardar que es unsigned long (4 bytes) y yo guardo de a un byte
  descompuesto[2]=EEPROM.read(dire*4+3); // el 4 es por la cantidad de bytes del
  índice a guardar que es unsigned long (4 bytes) y yo guardo de a un byte
  descompuesto[3]=EEPROM.read(dire*4+4);  // el 4 es por la cantidad de bytes del
  índice a guardar que es unsigned long (4 bytes) y yo guardo de a un byte
  //combino los bytes leídos desde la EEPROM en una unsigned long
  unsigned long index;
  index = ((unsigned long )descompuesto[3]) << 24;
  index |= ((unsigned long )descompuesto[2]) << 16;
  index |= ((unsigned long )descompuesto[1]) << 8;
  index |= descompuesto[0];
  return index;
}

void escribirIndice(unsigned long index, byte dire)
{ byte descompuesto[4];
  //la librería eeprom guarda de a un byte, mi índice tiene 4 byte por lo que tengo
  que separa los bytes  primero
  descompuesto[0] = (byte) index;
  descompuesto[1] = (byte) (index >> 8);
  descompuesto[2] = (byte) (index >> 16);
  descompuesto[3] = (byte) (index >> 24);
  EEPROM.write((dire*4+1), descompuesto[0]); // el 4 es por la cantidad de bytes
  del índice a guardar que es unsigned long (4 bytes) y yo guardo de a un byte
  EEPROM.write((dire*4+2), descompuesto[1]); // el 4 es por la cantidad de bytes
  del índice a guardar que es unsigned long (4 bytes) y yo guardo de a un byte
  EEPROM.write((dire*4+3), descompuesto[2]); // el 4 es por la cantidad de bytes
  del índice a guardar que es unsigned long (4 bytes) y yo guardo de a un byte
  EEPROM.write((dire*4+4), descompuesto[3]); // el 4 es por la cantidad de bytes
  del índice a guardar que es unsigned long (4 bytes) y yo guardo de a un byte.
}

byte hexToByte(char c)
{
  // Consigue el valor  0...15 de la variable c
  // Procesa el digito, si c<'A', sino letra
```

```
  return c<'A'? c & 0xF : 9 + (c & 0xF);
}
```

Código Receptor

```
#include <EEPROM.h>
#include <avr/pgmspace.h>
#include <VirtualWire.h>
#include <sha256.h>
#include <Arduino.h>
#include <printf.h>

const byte fragmentosTransmitidos=2;
 const byte longitudClave=(32/fragmentosTransmitir); // en bytes busco una clave de
256 bits
const byte bytesEncabezado=7;  //son 7 los primeros bytes que uso para indicar
numero de índice (4), dispositivo (1) y comando a ejecutar (1), secuencia se
segmento transmitido (1)
uint8_t* codigo="{FXH*Ua@EJ3{GHGTK(H_JfP+zUfvG?xT";   //generada con
https://pinetools.com/random-string-generator usando los 4 grupos de caracteres
const int receive_pin = 2;
const int led_pin = 13;
const byte cantidadControles=50;
byte secuenciaCorrectaSegmento;
byte codigoRecibido[longitudClave];
unsigned long indiceSegmento1;
const unsigned long indiceMaximo=4294967294;
//unsigned long indices[cantidadControles];   borrar si se puede
const byte  cant=2; //cantidad de botones del control, deben ser múltiplo de 2 .
Los botones pares activan y los impares desactivan
const byte incrementoCorrecion=105;
const int maximaDiferenciaIndices=10500 ;
byte badAuth=0;
unsigned int inicioVuelta[cantidadControles*cant]; //uso estas variables para
medir el tiempo antes de corregir con el incremento los indices
boolean necesitaIndiceCorrecion[cantidadControles*cant]; //uso estas variables
para saber si necesito medir tiempo para realizar una corrección de indices
byte cantidadNecesitaIndiceCorrecion=0;
const byte tiempoIncremento=5; //cantidad de segundo despues de las que se
realizar la corrección del índice del boton presionado


void setup()
{ pinMode(led_pin,OUTPUT);
  Serial.begin(9600);
  Serial.println("Hola, soy el receptor");
  iniciarReceptorRF();
  for(byte i=0;i<cant;i++)
     { inicioVuelta[i]=millis();
       necesitaIndiceCorrecion[i]=false;
     }
}
```

```
void loop()
{ //reviso que no haya indices que caduque y si los hay, los actualizo
  if(cantidadNecesitaIndiceCorrecion>0)
    { for(byte i=0;i<cantidadControles*cant;i++)
        {  if(necesitaIndiceCorrecion[i])
             { int vuelta=(int)(((unsigned int)millis()-inicioVuelta[i])/1000);
               if(vuelta >=tiempoIncremento)
                 { byte nroControl=(int)(i/cant);
                   byte nroBoton=i % cant;
                   unsigned long indiceLocalACorregir=leerIndice(nroControl,
                   nroBoton);
                   if(indiceLocalACorregir > indiceMaximo-incrementoCorrecion)
                     { indiceLocalACorregir= indiceLocalACorregir+
                       incrementoCorrecion-indiceMaximo;
                     }
                   else
                     { indiceLocalACorregir=indiceLocalACorregir+
                       incrementoCorrecion;
                     }
                   escribirIndice(indiceLocalACorregir,nroControl,nroBoton);
                   Serial.print("Indice del boton ");//////////////
                   Serial.print(nroBoton); ////////////////////////
                   Serial.print(" del control ");//////////////////
                   Serial.print(nroControl); //////////////////////
                   Serial.print(" despues de que caduco: ");///////
                   Serial.print(leerIndice(nroControl,nroBoton/////
                   Serial.println("\\\\\\\\\\\\\\\\\");/////////////
                   Serial.println("");//////////////////////////////
                   necesitaIndiceCorrecion[i]=false;
                   cantidadNecesitaIndiceCorrecion--;
                 }
             }
        }
    }
  receptorRF();
}


void iniciarReceptorRF()
 { vw_set_rx_pin(receive_pin);
   vw_setup(2000);   // Bits per sec
   vw_rx_start();
   secuenciaCorrectaSegmento=0;

 }

void receptorRF()
{ uint8_t buf[VW_MAX_MESSAGE_LEN];
  uint8_t buflen = VW_MAX_MESSAGE_LEN;
  if(vw_get_message(buf, &buflen)) // Non-blocking
    { int i;
      digitalWrite(led_pin, HIGH); // Flash a light to show received good message
      Serial.println("Algo llego");////////////////////////
      //Verifico que el número de segmento sea el correcto
      if(buf[6] == 0)
        { secuenciaCorrectaSegmento=1;
```

```
      indiceSegmento1 = ((unsigned long )buf[3]) << 24;
      indiceSegmento1 |= ((unsigned long )buf[2]) << 16;
      indiceSegmento1 |= ((unsigned long )buf[1]) << 8;
      indiceSegmento1 |= buf[0];
      for(byte i=0; i < (longitudClave/fragmentosTransmitidos); i++) //consigo
      el primer fragmento del codigo que transmitió el control remoto
         { codigoRecibido[i]=buf[i+bytesEncabezado];
         }
      Serial.println("Fragmento UNO recibido--------------------");/////
   }
else
   { if(buf[6] == 1 && secuenciaCorrectaSegmento == 1)
       { Serial.println("Fragmento DOS recibido+++++++++++++++");//////
         Serial.println("");//////////////////////////////////////////////
         unsigned long indiceSegmento2;
         indiceSegmento2 = ((unsigned long )buf[3]) << 24;
         indiceSegmento2 |= ((unsigned long )buf[2]) << 16;
         indiceSegmento2 |= ((unsigned long )buf[1]) << 8;
         indiceSegmento2 |= buf[0];
          secuenciaCorrectaSegmento=0;
         if(indiceSegmento1 == indiceSegmento2 )
           { byte numeroControl=buf[4];
             unsigned long indiceLocal= leerIndice((byte)buf[4],
             (byte)buf[5]);
             Serial.print("indice Local      ");//////////////////////
             Serial.println(indiceLocal);//////////////////////////////
             Serial.print("indice Recibido   ");//////////////////////
             Serial.println(indiceSegmento1);//////////////////////////
             //Controlo que el indice local sea menor al recibido en no mas
             de la diferencia máxima aceptada o  que el indice recibido que
             dio la vuelta y arranco desde cero, no sea mayor a la diferencia
             máxima
             if( (indiceLocal <=indiceSegmento1) && (((indiceSegmento1-
            indiceLocal)<= maximaDiferenciaIndices) || (indiceMaximo-
            indiceLocal+indiceSegmento1 <= maximaDiferenciaIndices)) )
               { //consigo el segundo segmento del codigo que transmitió el
                 control remoto
                 for(byte s=(longitudClave/fragmentosTransmitidos);
                 s <longitudClave ; s++)
                    { codigoRecibido[s]=buf[s-(longitudClave/
                      fragmentosTransmitidos) +bytesEncabezado];
                    }
                 Serial.print("indice Aceptado    ");///////////////
                 Serial.println(indiceSegmento1);////////////////////
                 Serial.println("");//////////////////////////////////
                 //genero el codigo de autenticación con mi clave y la
                 combinación del indice correspondiente concatenado con el
                 nroControl y nroBoton
                 Sha256.initHmac((uint8_t*)codigo,longitudClave);
                 Sha256.print((String)indiceSegmento1+"-"+(String)
                 numeroControl+ (String)buf[5]); //uso el indice recibido y
                 no el local porque pueden ser distintos y hasta validar la
                 clave no los igualo
                 String hmac=printHash(Sha256.resultHmac());//para verificar
                 el hmac https://codebeautify.org/hmac-generator
                 //el hmac que tengo obtuve esta HEX (64 caracteres) y
```

```
necesito pasarlo a bytes (32 caracteres) para transmitirlo
byte codigoLocal[longitudClave];
for(byte i=0; i<longitudClave;i++)
   { codigoLocal[i]= (byte) (hexToByte(hmac[i*2])<<4 |
     hexToByte(hmac[i*2+1] ));
   }
Serial.println("");//////////////////////////
Serial.print("Codigo para hmac:");/////////////
for(byte i=0;i<longitudClave;i++)//////////////
   { Serial.print((char)codigo[i//////////////
     Serial.println("");////////////////////////
     Serial.print("Entrada para hmac:");///////
     Serial.println((String) indiceSegmento1+"-
     "+(String)numeroControl+ (String)buf[5]);/
Serial.print("HMAC:");//////////////////////////
Serial.println(hmac);//////////////////////////
Serial.print("Codigo   Local :");//////////////
for(int i=0;i<longitudClave;i++)///////////////
   {Serial.print(codigoLocal[i]);///////////////
    Serial.print(" ");//////////////////////////
   }/////////////////////////////////////////////
Serial.println(" ");//////////////////////////
Serial.print("Codigo Recibido:");//////////////
for(int i=0;i<longitudClave;i++)///////////////
   {Serial.print(codigoRecibido[i]);///////////
    Serial.print(" ");//////////////////////////
   }/////////////////////////////////////////////
Serial.println(" ");//////////////////////////
Serial.println(" ");//////////////////////////
Serial.println(" ");//////////////////////////
if(compararClaves(codigoRecibido,codigoLocal))
  { // en cuanto valido que el codigo de autenticación es
    correcto, marco el incide correspondiente al boton
    presionado para que se incremente cuando caduque
    necesitaIndiceCorrecion[numeroControl*cant+buf[5]]
    =true;
    cantidadNecesitaIndiceCorrecion++;
    inicioVuelta[numeroControl*cant+buf[5]]=millis();
    //realizo las acciones correspondientes según el boton
    presionado
    if(buf[5]==1)
      { Serial.println("----------ABRIR PORTON----------");
      }
    else
      { if(buf[5]==0)
          { Serial.println("-------CERRAR PORTON---------");
          }
      }
   indiceLocal=indiceSegmento1;
   if((indiceLocal ==indiceMaximo))
     { indiceLocal=0;
     }
   else
     { indiceLocal++;
     }
   escribirIndice(indiceLocal,numeroControl,buf[5]);
```

```
                        badAuth=0;
                       }
                     else
                      { // si se reciben más de 10 solicitudes con claves
                        incorrectas se espera 1 segundo por petición con clave
                        invalida
                        badAuth++;
                        if(badAuth>=10)
                          { delay(1000);

                          }
                      }
                   }
                }
             }
          }
       Serial.println();///////////////
       digitalWrite(led_pin, LOW);
    }
}

String printHash(uint8_t* hash)
{ int i;
  String resu;
  for (i=0; i<32; i++)
       { resu+=(String)("0123456789abcdef"[hash[i]>>4]);
         resu+=(String)("0123456789abcdef"[hash[i]&0xf]);
       }
  return resu;
}

boolean compararClaves(char c1[], char c2[])
{ for(int i=0;i<longitudClave;i++)
       {if(c1[i]!= c2[i])
          {return false;}
       }
  return true;
}

unsigned long leerIndice(byte nroControl,byte nroBoton)
{ byte descompuesto[4];
  //la librería eeprom lee de a un byte, mi indice tiene 4 byte por lo que tengo
  que leer los bytes separándolos primero
  descompuesto[0]=EEPROM.read(nroControl*cant*4+nroBoton*4+1); // el 4 es por la
  cantidad de bytes del indice a guardar que es unsigned long (4 bytes) y yo
  guardo de a un byte
  descompuesto[1]=EEPROM.read(nroControl*cant*4+nroBoton*4+2); // el 4 es por la
  cantidad de bytes del indice a guardar que es unsigned long (4 bytes) y yo
  guardo de a un byte
  descompuesto[2]=EEPROM.read(nroControl*cant*4+nroBoton*4+3); // el 4 es por la
  cantidad de bytes del indice a guardar que es unsigned long (4 bytes) y yo
  guardo de a un byte
  descompuesto[3]=EEPROM.read(nroControl*cant*4+nroBoton*4+4); // el 4 es por la
  cantidad de bytes del indice a guardar que es unsigned long (4 bytes) y yo
  guardo de a un byte
  //combino los bytes leídos desde la EEPROM en una unsigned long
```

```
  unsigned long index;
  index = ((unsigned long )descompuesto[3]) << 24;
  index |= ((unsigned long )descompuesto[2]) << 16;
  index |= ((unsigned long )descompuesto[1]) << 8;
  index |= descompuesto[0];
  return index;
}

void escribirIndice(unsigned long index,byte nroControl,byte nroBoton)
{ byte descompuesto[4];
  //la librería eeprom guarda de a un byte, mi indice tiene 4 byte por lo que
  tengo que separa los bytes  primero
  descompuesto[0] = (byte) index;
  descompuesto[1] = (byte) (index >> 8);
  descompuesto[2] = (byte) (index >> 16);
  descompuesto[3] = (byte) (index >> 24);
  EEPROM.write((nroControl*cant*4+nroBoton*4+1), descompuesto[0]); // el 4 es por
  la cantidad de bytes del indice a guardar que es unsigned long (4 bytes) y yo
  guardo de a un byte
  EEPROM.write((nroControl*cant*4+nroBoton*4+2), descompuesto[1]); // el 4 es por
  la cantidad de bytes del indice a guardar que es unsigned long (4 bytes) y yo
  guardo de a un byte
  EEPROM.write((nroControl*cant*4+nroBoton*4+3), descompuesto[2]); // el 4 es por
  la cantidad de bytes del indice a guardar que es unsigned long (4 bytes) y yo
  guardo de a un byte
  EEPROM.write((nroControl*cant*4+nroBoton*4+4), descompuesto[3]); // el 4 es por
  la cantidad de bytes del indice a guardar que es unsigned long (4 bytes) y yo
  guardo de a un byte.
}

byte hexToByte(char c)
{// Consigue el valor  0...15 de la variable c
 // Procesa el digito, si c<'A', sino letra
  return c<'A'? c & 0xF : 9 + (c & 0xF);
}
```

G. Demostración

Con fines de realizar debug se ha insertado en el código líneas que muestran por pantalla los mensajes pertinentes a la sección del código correspondiente. Para poder visualizar estas salidas utilicé un conversor USB-TTL [11] para conectar, tanto el emisor como el receptor, a una PC. El conversor es accesible desde la PC en alguno de los puertos de comunicaciones seriales disponibles. Para poder accederlos, utilizaré la aplicación putty.exe.

En las siguientes capturas podemos ver las salidas de ambos dispositivos para distintos eventos.

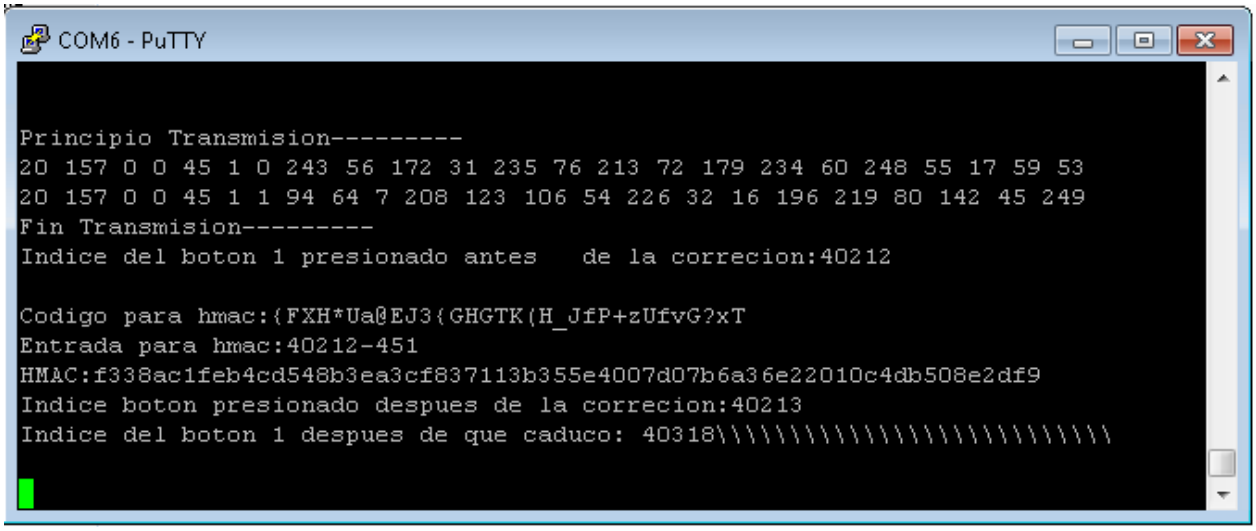

Fig. 8 Salida del Emisor para comando 1

En la figura 8 se puede observar la salida del emisor al presionar el botón para abrir el portón. Vemos uno a uno los 23 bytes (su equivalente en decimal) de cada trama que fueron enviados. En cada trama podemos observar: los cuatro primeros bytes que corresponden al índice del comando enviado, el quinto byte para identificar el control, el sexto para identificar el comando y el séptimo para indicar el número de trama. Los últimos 16 bytes corresponden a una mitad de la clave de 32 bytes. También podemos observar el índice correspondiente al botón presionado antes y después de modificarlos (en una unidad para el índice correspondiente al botón presionado y en 105 después de 5 segundos). Para poder corroborar las funciones criptográficas se incluye la información relacionada. Esto incluye la clave utilizada para calcular el código de autenticación (HMAC) . También se muestra al mensaje a ser procesado. El mismo está compuesto por el número de índice actual y los números de control y de comando correspondientes. Finalmente, el código de autenticación HMAC-SHA2 en hexadecimal, que es como lo presenta la librería criptográfica que se utilizó.

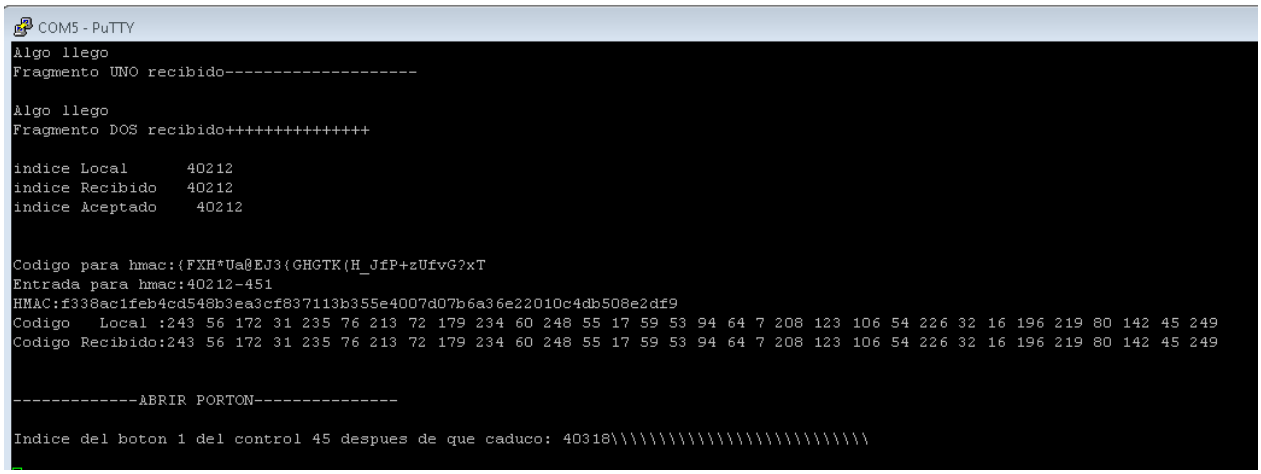

Fig. 9 Salida del Receptor para comando 1

En la Figura 9 se puede observar las salidas producidas por el Receptor, que notifica que se recibió una comunicación, una por cada trama. También se informa que se validó, primera trama y luego de la segunda. A continuación, se muestran las el índice local para el comando recibido, el indicie recibido y el que acepto (para los casos en donde ambos índices no están sincronizados). Acá también podemos ver la clave secreta utilizada, el mensaje a procesar y la salida en hexadecimal del código de autenticación. Seguido podemos observar el código de autenticación calculado localmente byte a byte usando la clave secreta sobre el índice aceptado. Si ambos códigos son iguales se mostrará una leyenda que corresponde a función invocada por el emisor, en este caso la de abrir el portón. Por último, se muestra el índice luego de ser incrementado tras 5 segundos desde que se aceptó el código recibido.

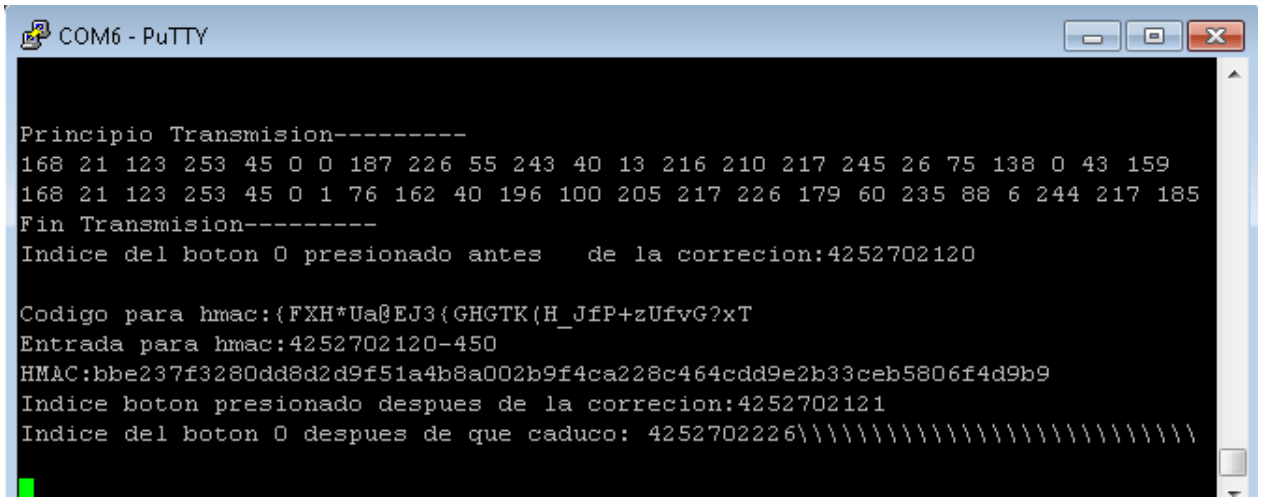

Fig. 10 Salida del Emisor para comando 0

En las figuras 10 y 11 podemos observar lo mencionado anteriormente pero con el otro comando disponible en nuestro control, cerrar el portón.

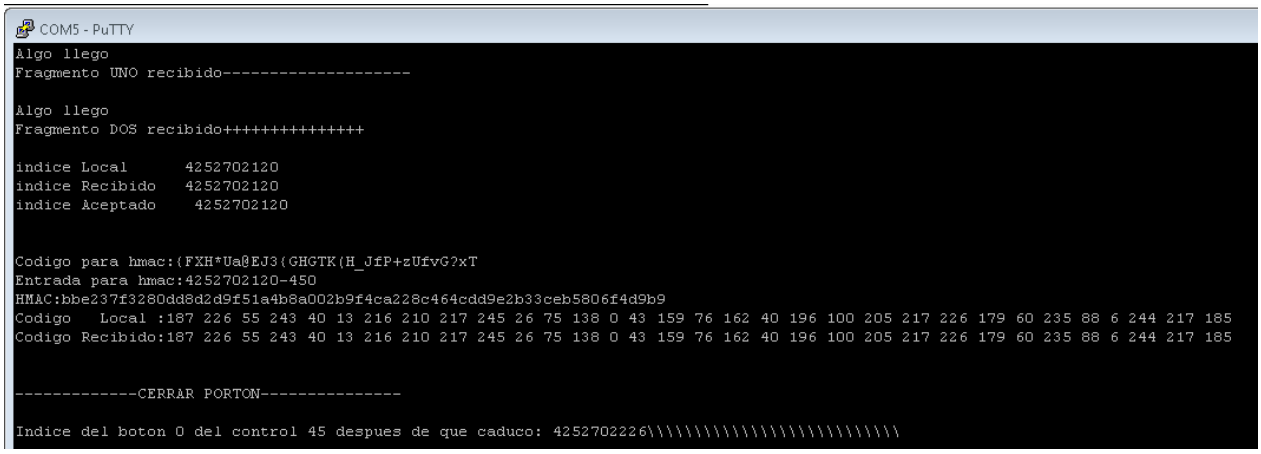

Fig. 11 Salida del Receptor para comando 0

En la figura 12 se observa que luego de presionar 3 veces el mismo botón, solo se incrementa el correspondiente indice en 105 unidades una sola vez, 5 segundos después de la última vez que se presionó el botón. Se intenta disminuir al mínimo la cantidad de veces que se incrementa un indice. Esto se debe a que al hacerlo se escribe en la EEPROM y minimizar esta operación alarga la vida útil del dispositivo.

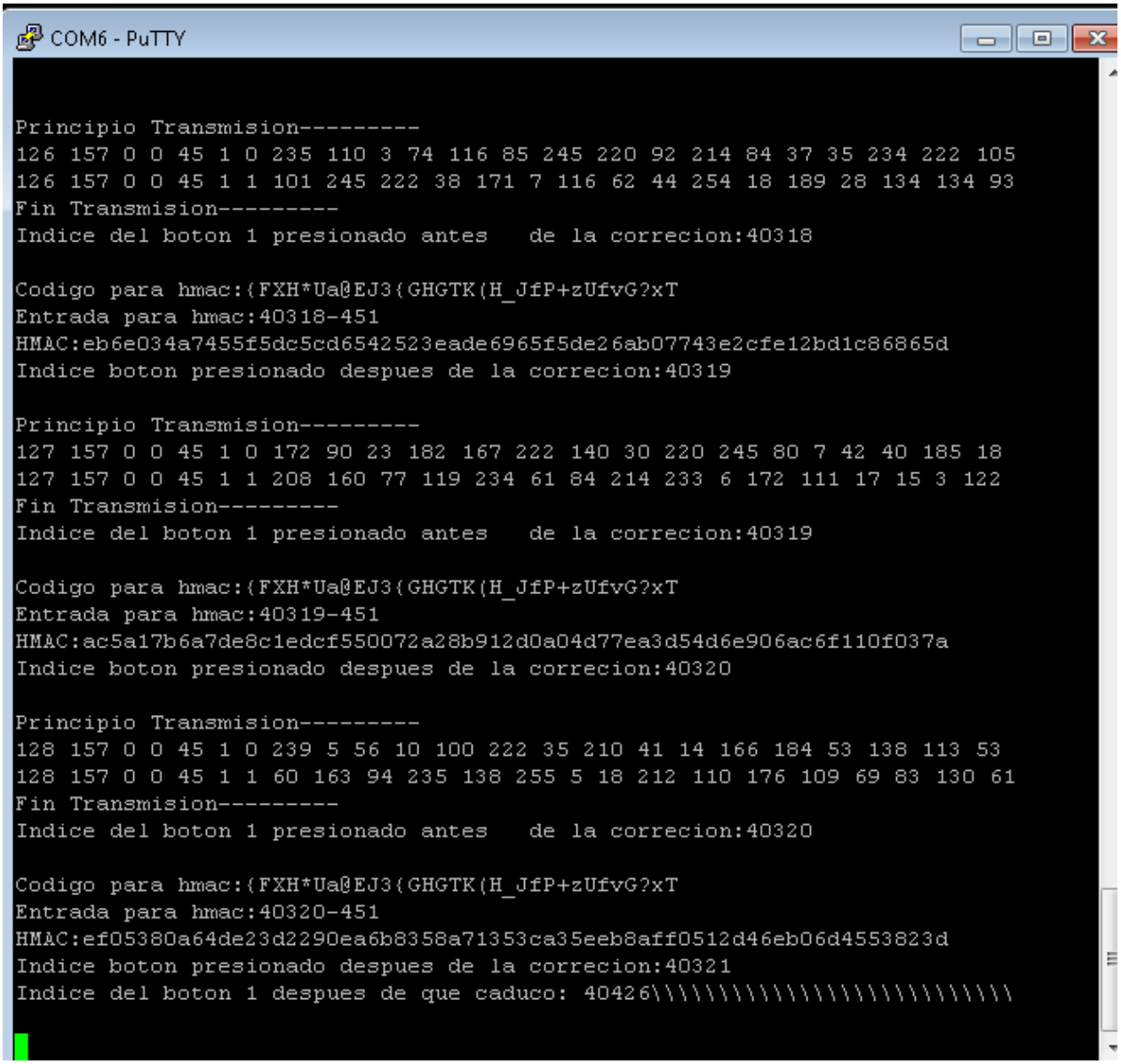

Fig .12 Salida de presionar varias veces un botón.

En la figura 13 que se ve a continuación, se muestra la salida del receptor al presionar un botón y el otro inmediatamente después. Se puede observar que se incrementa el indice correspondiente de cada botón 5 segundo después. También se puede notar para la función “Cerrar Portón ”, que el índice del receptor estaba desincronizado con el emisor pero como estaba dentro del rango permitido es aceptado y se igualan.

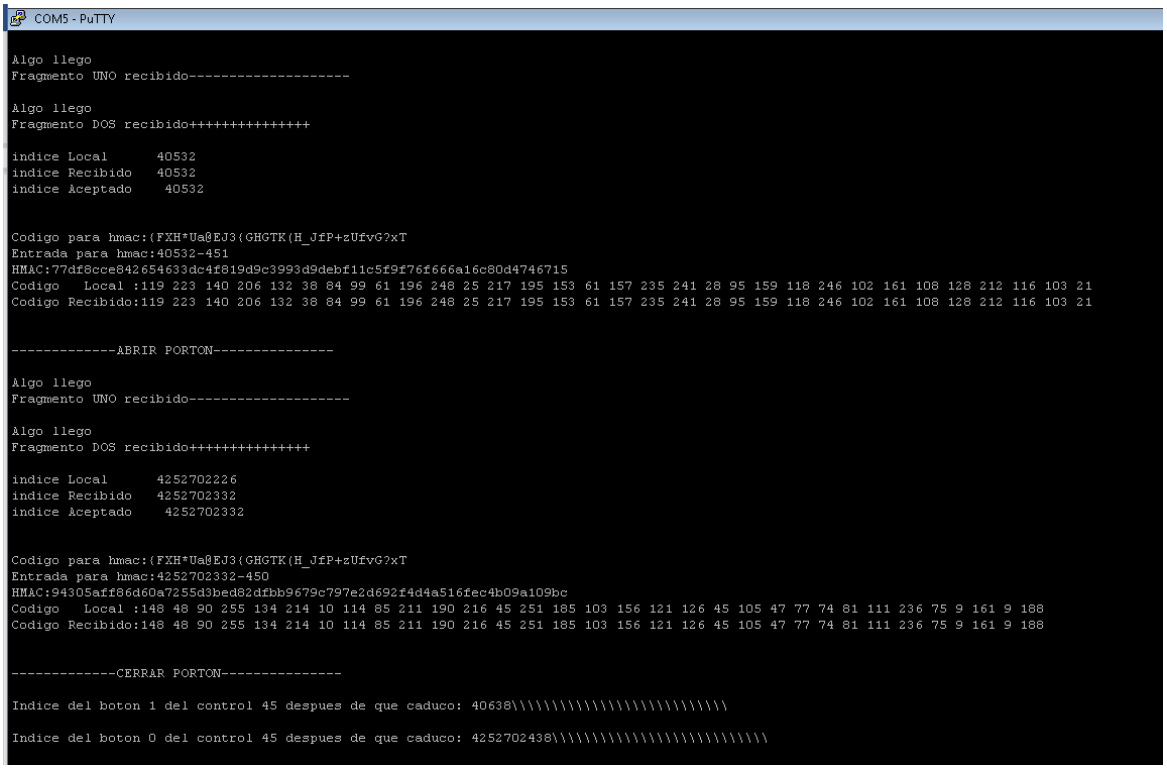

Fig. 13 Salida al presionar botones intercalados

8. Conclusiones

No se debe utilizar para autenticación de dispositivos claves fijas, independientemente de la longitud de las mismas. Se debe utilizar algún tipo de preámbulo para evitar la utilización de técnicas para la reducción del espacio de claves como la secuencia de Brujin. Se debe utilizar HMAC junto con una función de hash segura una clave suficientemente segura (longitud y aleatoriedad) para minimizar la posibilidad de ataques exitosos.

En la implementación de la solución propuesta se pudieron alcanzar todos los objetivos planteados, logrando así alcanzar una solución económica y robusta para mitigar las amenazas más comunes para este tipo de dispositivos. Al ser la comunicación unidireccional, no es posible la sincronización de tiempo entre ambos dispositivos. Si bien no se puede asegurar la caducidad de las claves transmitidas de forma absoluta, se encontró una alternativa para mitigar este problema.

Se alcanzó un dispositivo con un nivel se seguridad muy superior a la mayoría de los equivalente que se consiguen en el mercado. Los mismos tiene clave fija, generalmente de 24 bits de longitud. Estos dispositivos pueden ser muy fácil y rápidamente vulnerados por una computadora con un transmisor RF. El dispositivo construido resiste ataques de fuerza bruta, con una clave de 256 bits, de reproducción y de hombre en el medio. Se puede mejorar el sistema con módulos emisores/receptores tanto en el emisor como en el receptor, para poder asegurar la caducidad de las claves. Si bien el mencionado módulo está disponible, el costo del mismo todavía el elevado.

9. Referencias / Bibliografía

https://es.wikipedia.org/wiki/Ataque_de_REPLAY [1]
http://www.businessinsider.com/samy-kamkar-keyless-entry-car-hack-2015-8 [2]
https://en.wikipedia.org/wiki/Brute-force_attack [3]
https://en.wikipedia.org/wiki/De_Bruijn_sequence [4]
http://samy.pl/opensesame/ [5]
https://en.wikipedia.org/wiki/Rolling_code [6]
http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/KasperTimo/diss.pdf [7]
https://en.wikipedia.org/wiki/Man-in-the-middle_attack [8]
https://store.arduino.cc/usa/arduino-pro-mini [9]
http://saber.patagoniatec.com/modulos-emisor-y-receptor-rf-433mhz/ [10]
https://naylampmechatronics.com/conversores-ttl/79-modulo-cp2102-conversor-usb-a-ttl.html [11]