Universidad
Rey Juan Carlos

CERROJO INTELIGENTE

SISTEMAS EMPOTRADOS Y DE TIEMPO REAL

GRUPO 15
ALEXANDRO SEOANE DOMÍNGUEZ
MARCOS FERRER ZALVE

Universidad
Rey Juan Carlos

ÍNDICE

1. OBJETIVO

Nuestra principal idea era diseñar un proyecto que tuviese aplicaciones reales y útiles; permitiéndonos que usando el mismo esquema de diseño que hemos planteado sea posible utilizar nuestro proyecto de forma cotidiana.
Además, estuvimos pensado y el ámbito que nos parecía más importante en nuestro día a día era la seguridad de nuestras casas.
Llegando así a la idea de diseñar un cerrojo inteligente que nos facilite y nos acomode el acceso a nuestras casas.

2. IMPLEMENTACIÓN

La implementación del proyecto comenzó con dos ideas fundamentales: la primera que fuese totalmente remoto para poder gestionarlo desde cualquier lugar, y la segunda es evitar el uso de llaves ya que esto nos complicaría la idea de conseguir centralizar la gestión del cerrojo.

Por ello el primer paso fue instalar en el Arduino un módulo bluetooth (concretamente el módulo HC-05), con esto nos permitía recibir y enviar información al Arduino. Para poder transmitir dicha información diseñamos una pequeña aplicación móvil la cual nos permite abrir el cerrojo siempre que estemos conectados al HC-05, y siempre que introdujésemos la contraseña correcta.
Esta aplicación la creamos con la ayuda de la aplicación web MIT App Inventor, la cual sigue una interfaz de programación por bloques que facilita bastante la implementación de la app.

El siguiente paso para conseguir nuestro objetivo fue instalar un módulo RTC que nos permitía comprobar la hora actual y poder añadir diferentes horarios de apertura para el cerrojo, añadiendo asi más funciones como añadir horarios y eliminar horarios.

Finalmente, para concluir con nuestro diseño de cerrojo, decidimos instalar un Shield de Ethernet (módulo W5100 R3) a la parte superior del Arduino. Esta nos permitía conectar un cable de red a la placa y así conseguir comunicarnos con ella desde cualquier lugar que tuviésemos una conexión internet.

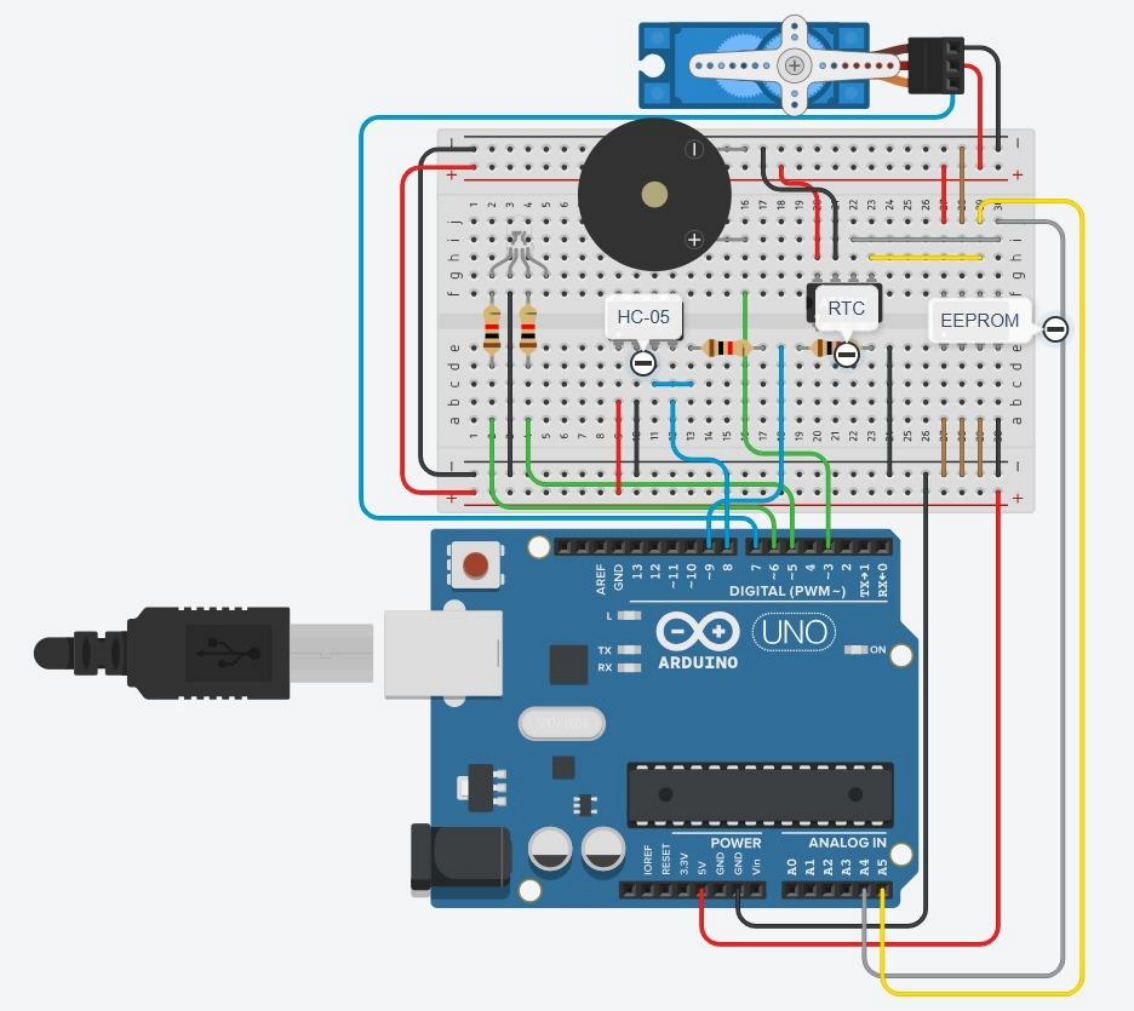
Todas estas opciones que tenemos para comunicarnos con la placa lo que no permiten hacer enviar una señal, a través del Arduino, a un servomotor que actúa como el propio cerrojo, permitiendo o no la apertura de la puerta donde este instalado el cerrojo inteligente. Luego para dar cierto feedback al usuario hemos instalado un LED RGB y un zumbador los cuales indican a través de luces y sonido si se permite o no abrir el cerrojo.

También, destacar que la contraseña que se usa dentro del código Arduino se encuentra almacenada en la EEPROM y de forma cifrada, para que ningun usuario pueda conseguir la contraseña mirando solo el código base.

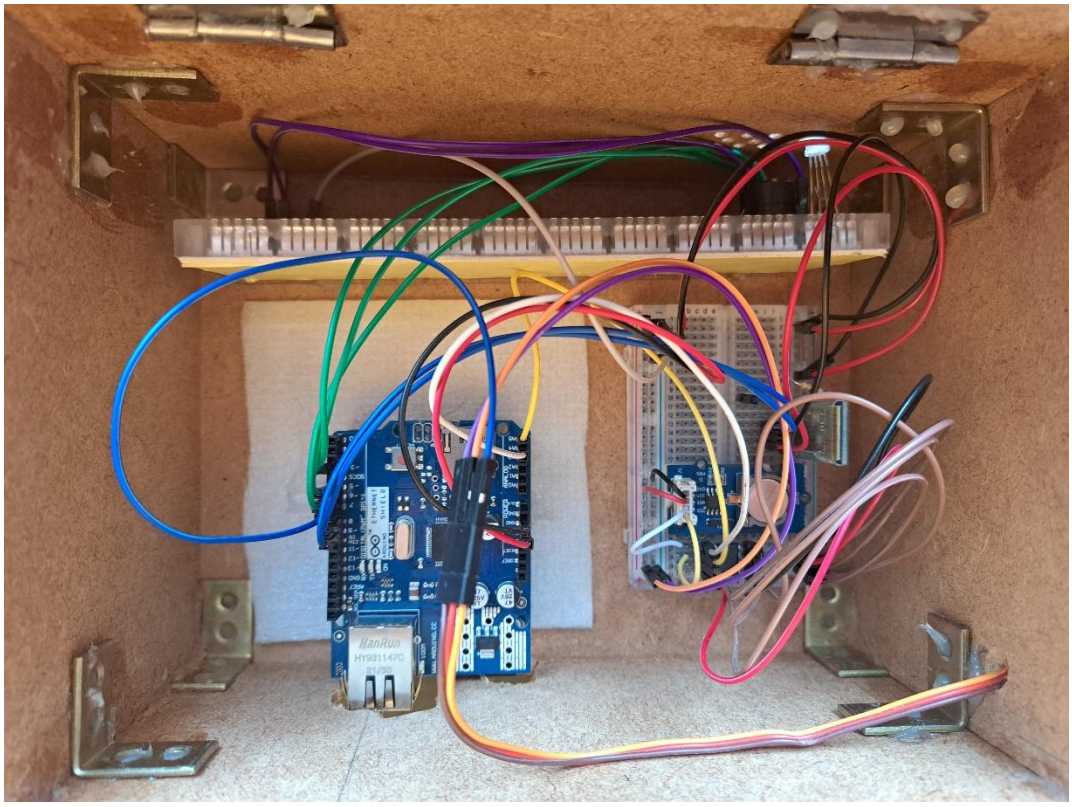
3. HARDWARE

Para la parte del diseño de hardware nos hemos estado ayudando de durante todo el diseño de algunas aplicaciones web como tinkercad, ya que su uso nos facilitaba bastante tanto el diseño , como el testeo del mismo.

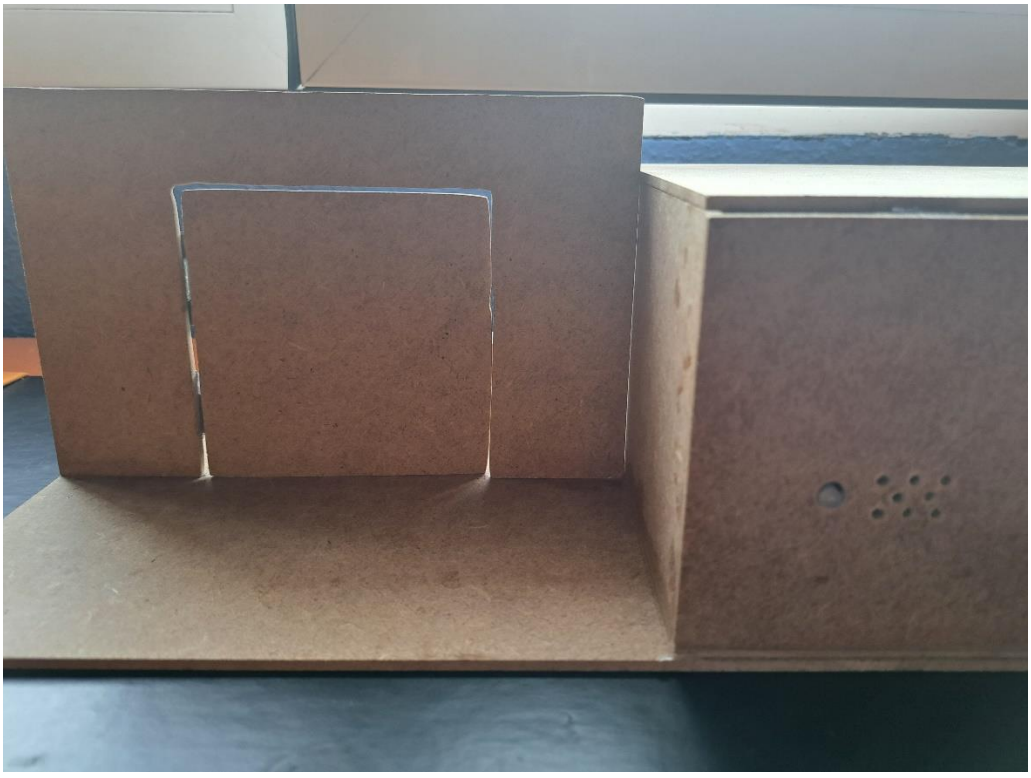
Aquí podemos ver el esquema en tinkercad final del circuito.



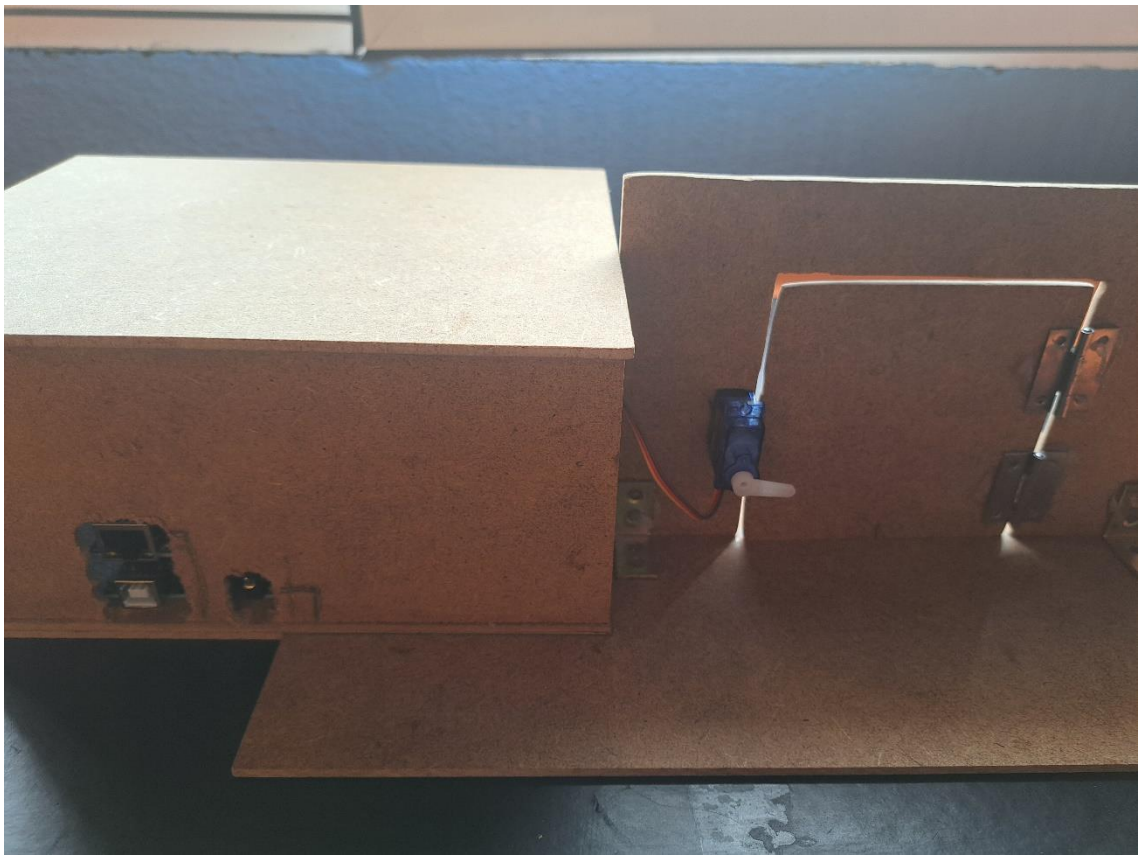
Aquí tenemos el diseño final del proyecto



La parte frontal del proyecto, donde podemos ver el LED RGB y los orificios de salida del audio del zumbador



La parte trasera del proyecto, donde podemos ver el servomotor y las diferentes conexiones de la placa (conector red, conector de pila y conector al ordenador)



Y está formado por los siguientes componentes

COMPONENTES	PRECIO
Protoboard – 400 contactos	3,64€
Protoboard	Incluido en el kit
Modulo bluetooth HC-05	15€
Modulo RTC	8,70€
Shield Ethernet W5100 R3	25,73€
Servomotor	Incluido en el kit
LED RGB	Incluido en el kit
Zumbador	Incluido en el kit
Resistencias	Incluido en el kit
Cables	Incluido en el kit
Madera	12€
Bisagras	3€
Total	68,07 (sin incluir el precio del kit)

4. CÓDIGO

En esta sección indicaremos los pines que hemos utilizado y a que componentes estan conectados

Pin	Componente	Función
D3	Zumbador	Generar tonos cuando abre, cierra y cuando la contraseña introducida es errónea
D5	LED RGB (verde)	Controla el color verde del LED RGB
D6	LED RGB (rojo)	Controla el color rojo del LED RGB
D7	Servomotor	Mueve el servo a 0º o a 90º según si queremos abrir o cerrar
D8	HC-05 (RXD)	Recibe datos del Arduino
D9	HC-05 (TXD)	Envía datos al Arduino
D10 – D13	Shield Ethernet	Usados para el manejador del módulo ethernet
A4	Modulo RTC y Memoria EEPROM*	Línea de datos, señal SDA
A5	Modulo RTC y Memoria EEPROM*	Reloj del bus, señal SCL

*Tanto el módulo RTC como la memoria EEPROM usan para comunicarse con el Arduino el protocolo I2C, el cual solo necesita dos señales en cada componente

También podemos destacar el programa que cifra la contraseña válida y la escribe en la memoria EEPROM, para hacer esto es necesario cambiar manualmente la conexión del pin WP de la EEPROM de GND a VCC.
Este programa va aparte del código fuente del Arduino y solo debería poder acceder el encargado de proporcionar el servicio. A continuación, se muestra dicho programa:

```
#include <Wire.h>

const byte xorKeys[16] = {
  0x8C, 0xB5, 0xB6, 0x45, 0x6F, 0x92, 0xE1, 0x48,
  0x2D, 0xC5, 0x73, 0x7C, 0x25, 0xC8, 0x5D, 0x6F
};
const byte indexMap[16] = {
  5, 0, 2, 7, 14, 3, 8, 11, 1, 4, 12, 9, 15, 6, 10, 13
};

void setup() {
  Serial.begin(9600);
  Wire.begin();

  String psw = "1234"; //aquí se introduciría la contraseña que se desea guardar
  byte hashed[16];

  for (int i = 0; i < 16; i++) {
    char c = (i < psw.length()) ? psw.charAt(i) : 0;
    hashed[i] = c ^ xorKeys[i];
  }

  byte finalHash[16];
  for (int i = 0; i < 16; i++) {
    finalHash[i] = hashed[indexMap[i]];
  }

  writeEEPROM(0x50, 0x00, finalHash, 16);

  Serial.println("Contraseña escrita en EEPROM.");
}

void loop() {}

void writeEEPROM(byte deviceAddr, byte memAddr, byte* data, byte len) {
  Wire.beginTransmission(deviceAddr);
  Wire.write(memAddr);
  for (byte i = 0; i < len; i++) {
    Wire.write(data[i]);
  }
  Wire.endTransmission();
}
```

El código completo se encuentra en el Anexo I.

5. PROBLEMAS

Los principales problemas que nos hemos ido encontrado a lo largo del desarrollo han sido:

- Falta de pines: El Shield Ethernet ocupa muchos pines del Arduino para su propia gestión, lo que nos impidió conectar otros módulos que habíamos planeado, como sensores de temperatura o de movimiento (útiles para detectar manipulaciones indebidas del cerrojo). Tampoco pudimos integrar una pantalla LCD para mostrar información al usuario. Como alternativa, optamos por un LED RGB y un zumbador para transmitir el estado del sistema.

- Falta de memoria interna: Al tener un sketch tan extenso ocupábamos casi por completo la memoria de la que dispone el Arduino, esto nos llevó a incorporar una memoria EEPROM externa para almacenar la contraseña y la rutina de cifrado/descifrado.

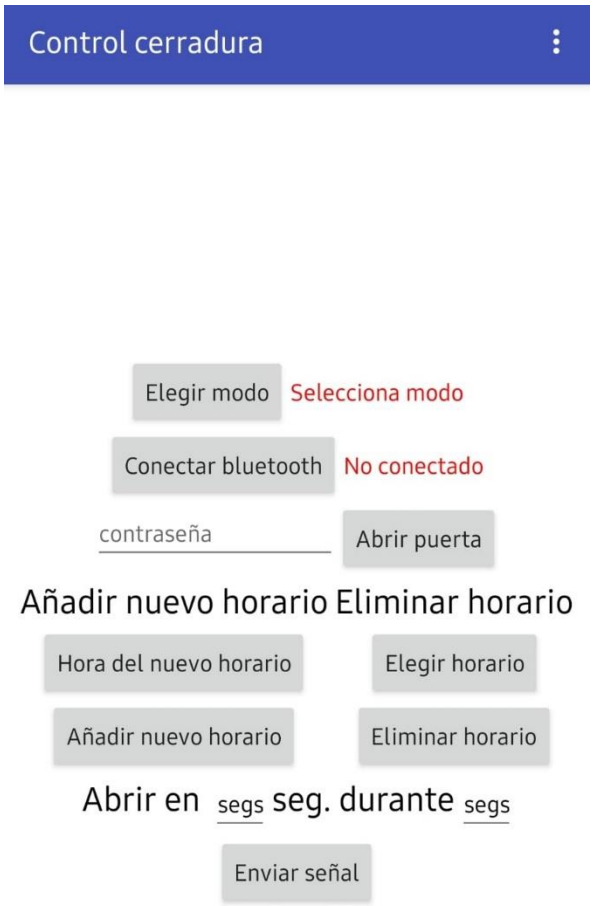
- Falta de componentes físicos compatibles: Inicialmente queríamos construir una puerta corredera automática (tipo garaje), pero no encontramos engranajes ni cremallera compatibles con nuestros motores. Aunque consideramos imprimir las piezas en 3D, resultaba demasiado caro, así que simplificamos el diseño al cerrojo accionado por servomotor.

- Ciertos problemas con el módulo de reloj: El módulo de reloj generaba lecturas erráticas, a veces mostraba horas imposibles, como “126:30”. Para corregirlo, implementamos en el código una rutina que filtraba y descartaba valores fuera de rango.

- Problemas con la IP: la aplicación móvil guarda la IP pública a la que se encuentra conectada cuando se inicia por primera vez, para poder llevar a cabo las diferentes funciones a través de internet. Pero como después con el tiempo esta IP cambia, provoca que la aplicación pierda la conexión impidiendo su correcto funcionamiento. Este problema no lo hemos conseguido solucionar.

6. FUNCIONAMIENTO

Como ya hemos mencionado antes el cerrojo se controla totalmente a través del teléfono móvil, por medio de una aplicación móvil sencilla donde se puede realizar toda la gestión.



Aquí podemos ver el menú de la aplicación. Lo primero que vemos es un botón donde nos deja elegir el tipo de conexión que queremos usar: bluetooth o internet (tener en cuenta que para conectarse por internet el Arduino debe estar conectado a través de un cable de red).
Luego si la opción ha sido por bluetooth necesitamos conectarnos con el cerrojo por medio del botón “Conectar bluetooth”.

Una vez realizado todo esto ya podemos comenzar a usar el cerrojo.
La primera opción es abrir la puerta de forma normal simplemente introduciendo la contraseña y pulsando el botón de “Abrir puerta”, si es correcta se abrirá el cerrojo durante 20 segundos, indicándolo con una luz verde y un sonido de aceptación (existe un sonido intermedio que nos indica que quedan 5 segundos para que se cierre) y después se cerrará; si no era correcta, lo indicara por medio del LED y el zumbador dejando la puerta cerrada.
En esta función la aplicación envía al Arduino un mensaje del siguiente tipo “1/contraseña”

La siguiente sección es la que se encarga de gestionar los horarios (añadirlos o eliminarlos), esto se hará pulsado en los diversos botones que tenemos en esta sección. En esta sección manda dos mensajes diferentes, el primero es “%hh:mm/contraseña” para añadir un horario; y el segundo es “$hh:mm/contraseña” para eliminar el horario elegido.
Finalmente, tenemos la tercera función que sirve para elegir en cuento tiempo se abre la puerta y durante cuantos segundos se mantiene abierta. En esta parte la aplicación manda el siguiente mensaje “@valor1$valor2/contraseña”, donde el valor1 son los segundos que tardará en abrir y valor2 los segundos que se mantendrá abierta.
Tanto en esta función como en la de los horarios es necesario siempre introducir la contraseña correcta para poder realizar cualquier uso o cambio de los mismos.

Universidad Rey Juan Carlos

ANEXO I

CÓDIGO ARDUINO

```
#include <Wire.h>
#include <RTClib.h>
#include <SoftwareSerial.h>
#include <SPI.h>
#include <Ethernet.h>
#include <Servo.h>

SoftwareSerial bluetooth(8, 9);
RTC_DS3231 rtc;

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 177);
EthernetServer server(8081);

Servo puertaServo;

const int TIME_OPEN = 20000;
const int ledR = 6;
const int ledG = 5;
const int buzzerPin = 3;
const int servoPin = 7;

const byte xorKeys[16] = {
  0x8C, 0xB5, 0xB6, 0x45, 0x6F, 0x92, 0xE1, 0x48,
  0x2D, 0xC5, 0x73, 0x7C, 0x25, 0xC8, 0x5D, 0x6F
};
const byte indexMap[16] = {
  5, 0, 2, 7, 14, 3, 8, 11, 1, 4, 12, 9, 15, 6, 10, 13
};

byte storedHash[16];
String schedules[10];
int schedulesIdx = 0;

void setup() {
  pinMode(ledR, OUTPUT);
  pinMode(ledG, OUTPUT);
  pinMode(buzzerPin, OUTPUT);

  puertaServo.attach(servoPin);
  puertaServo.write(0);

  setLEDColor(255, 0, 0);
```

```
  bluetooth.begin(9600);
  Serial.begin(9600);
  Wire.begin();

  Ethernet.begin(mac, ip);
  server.begin();

  if (!rtc.begin()) {
    Serial.println("No se encuentra el RTC");
    while (1);
  }
  if (rtc.lostPower()) {
    Serial.println("RTC sin hora. Estableciendo hora de compilación.");
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }

  readEEPROM(0x50, storedHash, 0x00, 16);
  Serial.println("Listo para recibir comandos por Bluetooth y red.");
}

void loop() {
  if (bluetooth.available()) {
    String command = bluetooth.readStringUntil('\n');
    handleCommand(command);
  }

  EthernetClient client = server.available();
  if (client) {
    String request = "";
    while (client.connected() && client.available()) {
      char c = client.read();
      request += c;
    }

    if (request.startsWith("GET /")) {
      int startIdx = 5;
      int endIdx = request.indexOf(' ', startIdx);
      if (endIdx != -1) {
        String command = request.substring(startIdx, endIdx);
        if (command.startsWith("/")) {
          command = command.substring(1);
        }
        handleCommand(command);
      }
    }
```

```
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println("Connection: close");
  client.println();
  client.println("<html><body><h1>Comando recibido</h1></body></html>");
  client.stop();
 }

 static int lastCheckedMinute = -1;
 DateTime now = rtc.now();
 int currentHour = now.hour();
 int currentMinute = now.minute();

 if (currentHour < 0 || currentHour > 23 || currentMinute < 0 || currentMinute > 59) {
  Serial.println("Hora inválida detectada. Saltando...");
  return;
 }


 if (currentHour >= 0 && currentHour <= 23 && currentMinute >= 0 && currentMinute <= 59) {
  if (currentMinute != lastCheckedMinute) {
   lastCheckedMinute = currentMinute;
   String nowStr = String(currentHour) + ":" + (currentMinute < 10 ? "0" : "") +
String(currentMinute);
   Serial.print("Hora actual: ");
   Serial.println(nowStr);

   for (int i = 0; i < schedulesIdx; i++) {
    if (schedules[i] == nowStr) {
     Serial.println("Horario coincidente. Abriendo...");
     buzzOpen();
     setLEDColor(0, 255, 0);
     delay(TIME_OPEN * 0.8);
     buzzWarning();
     delay(TIME_OPEN * 0.2);
     setLEDColor(255, 0, 0);
     buzzClose();
     Serial.println("Cierre completado.");
     break;
    }
   }
  }
 }
}
```

```
void handleCommand(String command) {
  Serial.print("Recibido: ");
  Serial.println(command);

  if (command.startsWith("1/")) {
    simpleOpen(command);
  } else if (command.startsWith("@")) {
    timedOpen(command);
  } else if (command.startsWith("%")) {
    saveSchedule(command);
  } else if (command.startsWith("$")) {
    deleteSchedule(command);
  }
}

bool isCodeValid(const String& input) {
  if (input.length() == 0) return false;
  byte hashedInput[16];
  for (int i = 0; i < 16; i++) {
    char c = (i < input.length()) ? input.charAt(i) : 0;
    hashedInput[i] = c ^ xorKeys[i];
  }
  byte obfuscated[16];
  for (int i = 0; i < 16; i++) {
    obfuscated[i] = hashedInput[indexMap[i]];
  }
  for (int i = 0; i < 16; i++) {
    if (obfuscated[i] != storedHash[i]) return false;
  }
  return true;
}

void readEEPROM(byte deviceAddr, byte* buffer, byte startAddr, byte length) {
  Wire.beginTransmission(deviceAddr);
  Wire.write(startAddr);
  Wire.endTransmission();
  Wire.requestFrom(deviceAddr, length);
  for (byte i = 0; i < length && Wire.available(); i++) {
    buffer[i] = Wire.read();
  }
}

void simpleOpen(String command) {
  int passIndex = command.indexOf('/');
  String psw = command.substring(passIndex + 1);
  if (isCodeValid(psw)) {
    buzzOpen();
```

```
    setLEDColor(0, 255, 0);
    Serial.println("OPEN");
    delay(TIME_OPEN * 0.8);
    buzzWarning();
    delay(TIME_OPEN * 0.2);
    setLEDColor(255, 0, 0);
    buzzClose();
    Serial.println("CLOSE");
  } else {
    Serial.println("UNAUTHORIZED");
    buzzUnathorized();
  }
}

void timedOpen(String command) {
  int hashIndex = command.indexOf('$');
  int passIndex = command.indexOf('/');

  int inTime = command.substring(1, hashIndex).toInt();
  int forTime = command.substring(hashIndex + 1, passIndex).toInt();
  String psw = command.substring(passIndex + 1);

  if (isCodeValid(psw)) {
    delay(inTime * 1000);
    buzzOpen();
    setLEDColor(0, 255, 0);
    Serial.println("OPEN");
    delay(forTime * 1000 * 0.8);
    buzzWarning();
    delay(forTime * 1000 * 0.2);
    setLEDColor(255, 0, 0);
    buzzClose();
    Serial.println("CLOSE");
  } else {
    Serial.println("UNAUTHORIZED");
    buzzUnathorized();
  }
}

void saveSchedule(String command) {
  int passIndex = command.indexOf('/');
  String schedule = command.substring(1, passIndex);
  String psw = command.substring(passIndex + 1);

  if (isCodeValid(psw)) {
    addSchedule(schedule);
  } else {
```

```
    Serial.println("UNAUTHORIZED");
    buzzUnathorized();
  }
}

void addSchedule(String sch) {
  if (schedulesIdx < 10) {
    schedules[schedulesIdx] = sch;
    schedulesIdx++;
    Serial.print("Añadido nuevo horario: ");
    Serial.println(sch);
  } else {
    Serial.println("Máximo de horarios alcanzado.");
    buzzUnathorized();

  }
}

void deleteSchedule(String command) {
  int passIndex = command.indexOf('/');
  String schedule = command.substring(1, passIndex);
  String psw = command.substring(passIndex + 1);

  if (isCodeValid(psw)) {
    if (removeElement(schedule)) {
      Serial.println("Elemento eliminado.");
    } else {
      Serial.println("Elemento no encontrado.");
    }
  } else {
    Serial.println("UNAUTHORIZED");
    buzzUnathorized();
  }
}

bool removeElement(String elem) {
  for (int i = 0; i < 10; i++) {
    if (schedules[i] == elem) {
      for (int j = i; j < 9; j++) {
        schedules[j] = schedules[j + 1];
      }
      schedules[10] = "";
      schedulesIdx--;
      return true;
    }
  }
  return false;
}
```

```
void setLEDColor(byte r, byte g, byte b) {
  analogWrite(ledR, 255 - r);
  analogWrite(ledG, 255 - g);
}

void buzzOpen() {
  puertaServo.write(90);
  tone(buzzerPin, 1000, 300);
}

void buzzClose() {
  puertaServo.write(0);
  tone(buzzerPin, 1000, 150);
  delay(10);
  tone(buzzerPin, 1000, 150);
}

void buzzWarning() {
  setLEDColor(0, 0, 0);
  delay(100);
  setLEDColor(0, 255, 0);
  tone(buzzerPin, 5000, 300);
}

void buzzUnathorized() {
  setLEDColor(0, 0, 0);
  tone(buzzerPin, 500, 100);
  delay(1000);
  setLEDColor(255, 0, 0);
  tone(buzzerPin, 500, 100);
}
```