UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

GESTIÓN DE SISTEMAS DE ACCESO INTELIGENTES CON LATCH

Autor: Ferrer Gallén, Iván

Tutor: López Patiño, José Enrique

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2017-18

Valencia, 20 de noviembre de 2018

Resumen

Se diseña un sistema para gestionar una cerradura inteligente en la cuál la apertura se solicita mediante la pulsación de un botón, y se comprueba la posibilidad de apertura a través de un sistema intermediario llamado Latch. Este sistema permite al usuario bloquear desde el móvil el acceso y recibir una alerta cuando hay solicitudes no permitidas. Para sumar más seguridad se conecta con una cámara que proporciona un streaming de video con la finalidad de visualizar a la persona que pulsa el botón de apertura.

Resum

Es disenya un sistema per a gestionar un pany intel·ligent en el qual l'obriment es sol·licita mitjançant la pulsació d'un botó, i es comprova la posibilitat d'obrir a través d'un sistema intermediari que es diu Latch. Aquest sistema permet al usuari bloquejar des del mòbil l'accés i rebre una alerta quan hi ha solicituts no permeses. Per a sumar més seguretat es conecta amb una càmera que dona un streaming de video amb la finalitat de vore a la persona que ha pulsat el botó d'obertura.

Abstract

It's designed a system to manage a smart lock which the opening of it is requested by pushing a button, and is checked the possibility of opening by a intermediary system called Latch. This system allow user to block from mobile the access and receiving an alert when there are forbidden requests. Adding more security is connected with a camera to provide a video streaming to visualize who is pushing the button to open.

Índice

Capítulo 1. Introducción

1.1 Situación actual del sector IoT

La introducción de dispositivos IoT (Internet of Things), en la vida cotidiana de las personas para favorecer el control de ciertos ámbitos de su casa, han impulsado un crecimiento de las empresas y un abaratamiento de los costes de fabricación. Es muy común ver en los hogares diferentes tipos de sensores, cámaras y otros dispositivos que puedan medir y proporcionar datos en tiempo real a sus usuarios.

El crecimiento de estos dispositivos se prevé que crezca un 17,9% hasta 2020. España es el quinto país de Europa en inversión en IoT. Con estos dispositivos las empresas logran mejorar productividad, ahorro en costes. Se espera en un futuro que haya más beneficios para consumidores y empresas, por ejemplo:

- Nuevos productos y servicios.
- Las empresas pueden ajustar y personalizar los servicios a cada cliente.
- Empresas proporcionan información útil y personalizada para que los consumidores tomen decisiones y puedan ahorrar tiempo y dinero, en energía y seguridad de sus hogares.

No sólo proporciona beneficios en los hogares conectados, el IoT también se está introduciendo en otros sectores: industrial, agrícola, salud, atención social, infraestructura urbana, servicios comunitarios, etc. Estos son solo algunos ejemplos de la implantación a gran escala que está llegando.

1.2 IoT: Principales riesgos

Pero muchos de estos dispositivos carecen de conexiones cifradas y herramientas de seguridad a la hora de mandar los datos de nuestras casas a través de la red. Han venido acompañados de dos riesgos:

1. La seguridad del consumidor y su privacidad están viéndose afectadas por la vulnerabilidad de estos pequeños dispositivos.
2. La economía se enfrenta diariamente a un creciente número de ataques informáticos que en ocasiones son lanzados desde este tipo de dispositivos inseguros. Ataques de Denegación distribuida de servicios (DDoS) para repercutir en empresas o incluso países.

Al aprovechar los fallos de seguridad en los dispositivos, los servicios comprometidos pueden causar muchos problemas. Por ejemplo, un micrófono o una cámara IP mal asegurada podría

exponer conversaciones o imágenes de los habitantes de la casa registrando una rutina que se pueden emplear para espiar o incluso robar en una vivienda.

Para la industria sanitaria es un problema añadido ya que los datos del historial médico de sus clientes son más lucrativos para los piratas informáticos incluso que las tarjetas de crédito.

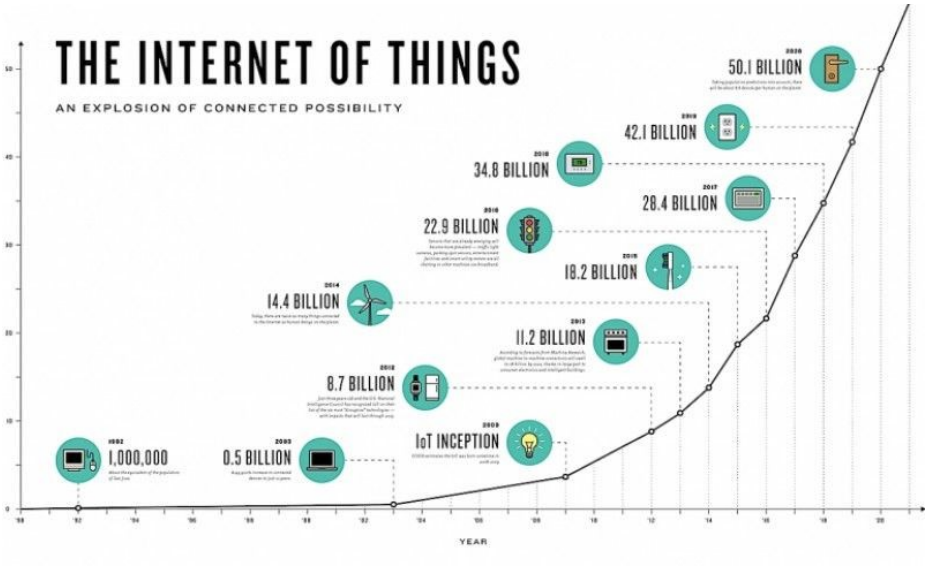

Figura 1. Implantación de dispositivos IoT

1.3 Recomendaciones de seguridad para dispositivos IoT

- Interfaz web
  - Analizar posibles vulnerabilidades web
  - Implementar uso de SSL para encriptar el flujo de información
  - Uso contraseñas robustas y cambiarlas con regularidad
  - Deshabilitar acceso remoto si no se va a utilizar
- Autenticación
  - Evaluar la posible implantación de segundo factor de autenticación
  - Evaluar mecanismo de recuperación de contraseñas
- Interfaz móvil
  - Evaluar posibilidad de autenticación en dos factores
  - Evaluar el uso de cifrado en el flujo de información
  - Instalar actualizaciones de las aplicaciones
- Firmware
  - Mantener actualizados los dispositivos a la última versión
  - Comprobar en la web de la marca que hay soporte para el dispositivo
- Vulnerabilidades físicas
  - Minimizar puertos USB innecesarios, sólo entradas alimentación

- Deshabilitar características que no vayamos a utilizar

Capítulo 2. Objetivos

2.1 Objetivo del trabajo de fin de grado (TFG)

El objetivo de este TFG es implementar un sistema doméstico de apertura de una puerta mediante una cerradura inteligente. Para dotar al sistema de la mayor seguridad posible se va a complementar con un servicio llamado Latch, que intermediará proporcionando al usuario la posibilidad a distancia mediante una app móvil de controlar el sistema y de recibir una notificación de alarma cuando se produzcan intentos de acceso no permitidos. Se hace uso de dispositivos como Arduino y Raspberry Pi que facilitan la interconexión de tecnologías analógicas y digitales con la programación de software.

El sistema se va a aproximar para ser lo más realista a una instalación real usando dispositivos lo más pequeños posibles, simulando así una posible instalación y no grandes montajes haciendo uso de mucho cableado. Se usará tecnología inalámbrica, un cable USB entre las placas y un cable de alimentación. El cable USB alimentará a un dispositivo para evitar otro alimentador más, además de proporcionar un canal de comunicación serie entre ellos.

2.2 Objetivos personales

Realizando este proyecto pretendo ser capaz de poner en práctica algunas de las diferentes áreas que he aprendido en este grado. Una de mis inquietudes personales, y por la que he querido obtener la titulación de ingeniero, es poder desarrollar ideas haciendo uso de las tecnologías a mi alcance, al igual que ser capaz por mi mismo de afrontar los diferentes problemas que me han ido surgiendo en el desarrollo.

Por otra parte, me parece muy interesante la idea de poder mezclar tecnologías diferentes para llevar a cabo un proyecto, investigar en que puede beneficiarme o que complicaciones puedo tener a la hora de elegir un camino u otro. Además, me despierta mucha curiosidad el mundo del IoT, ya que creo que en un futuro vamos a tener todo tipo de objetos de uso cotidiano conectados para facilitarnos muchas de las tareas que realizamos diariamente. Y, al mismo tiempo, me preocupa mucho la seguridad, ya que si abrimos nuestras casas al mundo y no están bien securizadas podemos tener problemas de privacidad.

Para finalizar, pienso que es un reto para mi poder aprender de forma autónoma herramientas diferentes a las adquiridas en la carrera. Enfrentarme a hacer uso de una API de un servicio y adquirir soltura para en un futuro hacer investigación e implementaciones por mi cuenta.

Capítulo 3. Latch, el interruptor para tu vida digital

3.1 ¿Qué es Latch?

Latch es un servicio que proporciona la capacidad de bloquear el acceso a nuestras cuentas en servicios digitales. Los usuarios en su vida digital se dan de alta en numerosos servicios: banca, correo electrónico, redes sociales, video, música…

Figura 2. Contraseñas de los servicios

Latch proporciona la posibilidad, a los desarrolladores, de añadir a sus servicios y aplicaciones una capa más de seguridad para que hagan uso sus usuarios. Brindando la posibilidad, a los usuarios, de añadir pestillos virtuales a cada uno de estos servicios, aunque alguien tuviera la contraseña para acceder sería imposible, y el usuario recibiría una notificación con el intento de acceso.

Figura 3. Servicios pareados con Latch

Además, Latch nos ofrece un servicio de generación de códigos temporales (TOTP) que permiten al usuario introducir un código temporal para hacer login en el servicio requerido. Y evitando otro tipo de sistemas como:

Figura 4. Otras opciones 2º Factor autenticación

Este tipo de servicios encarecen los costes teniendo que comprar bonos de paquetes de SMS, montar plataformas de mailing en los servicios o por ejemplo distribuir memorias USB con claves con el peligro de que empleados puedan perderlas.

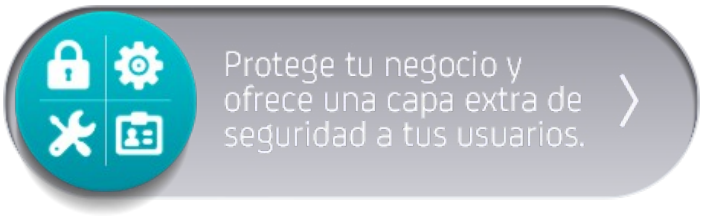

Figura 5. Ejemplo pestillo Latch

Los usuarios podrán bloquear de forma individual cada uno de los servicios que tengan pareados en la aplicación móvil con un pestillo o dispondrán de un pestillo global para bloquear todos los servicios a la vez. Podemos observar un ejemplo en la siguiente figura:

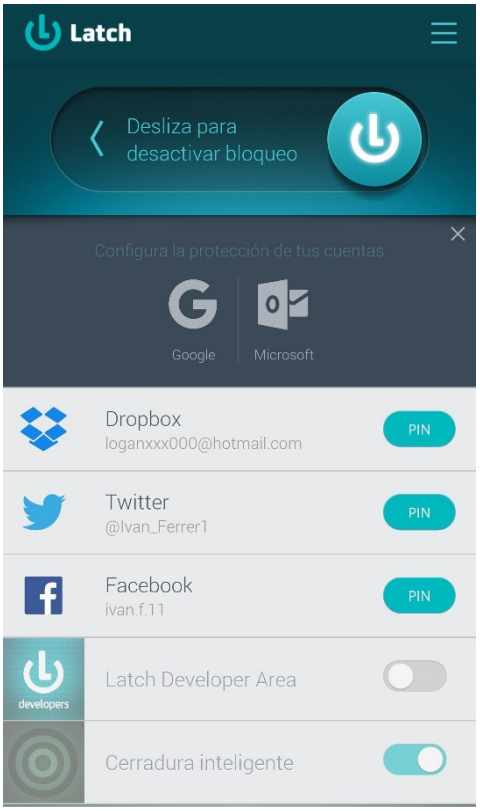

Figura 6. Ejemplo de servicios añadidos

3.2 ¿Cómo funciona?

- **Para usuarios**

La app móvil muestra una serie de guías para añadir algunos servicios muy conocidos como redes sociales, correo electrónicos, nubes de almacenamiento, entre otros. Teniendo posibilidad de ampliar los servicios controlados que necesiten factor TOTP.

- **Empresas y Desarrolladores**

Se dispone de un API para que las empresas puedan integrar sus servicios y para que los desarrolladores puedan añadirlo en sus aplicaciones.

En primer lugar hemos de crear una cuenta en su web [2]. Y debemos asociar las cuentas de los servicios que queramos incluir o las aplicaciones que hayamos creado. Para ello, se requiere introducir un código de pareado generado en el móvil. A partir de ese momento nos aparecerá en la app móvil un pestillo digital asociado al nombre del servicio o aplicación pareada.

3.3 Porqué usar Latch

Hoy en día es muy habitual, debido al incremento de los ataques informáticos, el hackeo de servicios que no han sido dotados de la seguridad adecuada. Hay una mayoría de usuarios que utilizan las mismas contraseñas en todos servicios que utilizan. Hemos podido observar que algunos de estos servicios tardan años o incluso no informan a sus usuarios de que han sido atacados sufriendo robos de información sensible. Incluso se han creado páginas web como https://haveibeenpwned.com/ o https://hacked-emails.com/check_email/ que permiten a los usuarios comprobar si su email ha sido comprometido en alguno de los servicios hackeados.

Latch nos permite reducir el riesgo de comprometer estos servicios aún incluso en el caso de que dispongan de nuestra contraseña correcta. Los usuarios pueden bloquear de manera individual o de forma general todos los servicios compatibles con Latch cuando no los están utilizando.

Desde la propia aplicación de móvil es sencillo configurar su uso con las aplicaciones más comunes como: Google, Facebook, Twitter, Dropbox, Microsoft.

Hay una gran comunidad de desarrolladores detrás ampliando los servicios compatibles mediante plugins, algunos ejemplos: OpenLDAP, Wordpress, Joomla, Drupal, OpenVPN, Windows, Linux, Ubuntu, Moodle, Owncloud, Jenkins….

A los usuarios les proporciona un control muy intuitivo y sencillo tanto para añadir los servicios como para bloquear/activar su uso. Añade una capa extra de seguridad para evitar intrusos.

Por otro lado, Latch permite acceder a los servicios usando Time-based One-Time Password Algorithm (TOTP). El uso del segundo factor de autenticación se ha extendido muy rápidamente y Latch permite gestionarlo escaneando el código QR que proporcionan estos servicios. De esta manera nos aparecerá en la aplicación el servicio seguido de la palabra PIN, que al pulsarla generará un código temporal junto a un pequeño reloj visual contabilizando el tiempo de validez de ese número.

3.4 API de Latch

La documentación sobre el proceso de pareado y consulta de las aplicaciones Latch se va a tratar en el **Anexo II**.

Capítulo 4. Estructura de la aplicación

Para realizar el proyecto se ha tenido que emplear diferentes dispositivos hardware y consta de las siguientes partes:

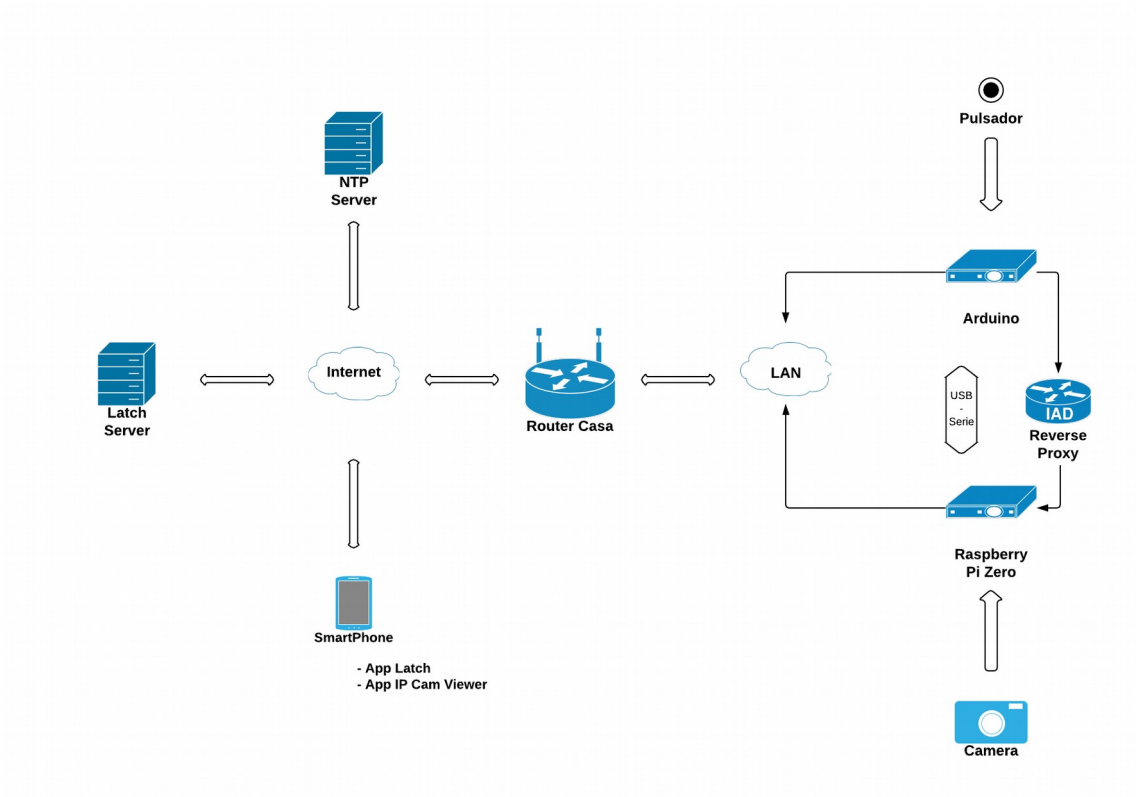

Figura 7. Diagrama de bloques del sistema

Como podemos observar en la figura 7, se van a emplear 2 dispositivos hardware para implementar este sistema. Estos dispositivos van a tener una serie de elementos conectados a ellos:

- En el caso de Arduino, una placa protoboard para hacer las conexiones oportunas con las resistencias y leds oportunos.
- En el caso de Raspberry Pi Zero, una cámara conectada por un bus directa a la placa.

Los dispositivos van a interactuar entre ellos de dos formas: por cable USB y por conexión wifi dentro de la red local (LAN). Establecerán conexión a Internet de forma inalámbrica para comunicarse con los servidores de Latch y NTP, así como con el móvil del usuario.

Los dispositivos serán programados para realizar las funciones correspondientes para el desarrollo correcto del sistema. El microcontrolador Arduino usará una aplicación que, una vez cargada en memoria, correrá indefinidamente. En el miniPC Raspberry Pi, empleará un script escrito en Python que comunicará en serie con Arduino para conectar y emitir la webcam.

Por otro lado, el miniPC hará funciones de proxy para reenviar las peticiones del microcontrolador para negociar con el servidor Latch, ya que Arduino es incapaz de hacerla por si mismo por falta de potencia.

El usuario, que estará fuera de casa, recibirá una alerta cuando tenga el sistema configurado mediante la aplicación de Latch (Bloqueado) con algún intento de apertura solicitado al presionar el botón. A partir de este momento podrá visualizar la app del móvil para visualizar la cámara web y ver quien ha pulsado el botón de apertura.

4.1 Aplicación Arduino

Todas las aplicaciones de Arduino constan de tres partes diferentes: librerías, función setup y bucle loop.

4.1.1 Librerías

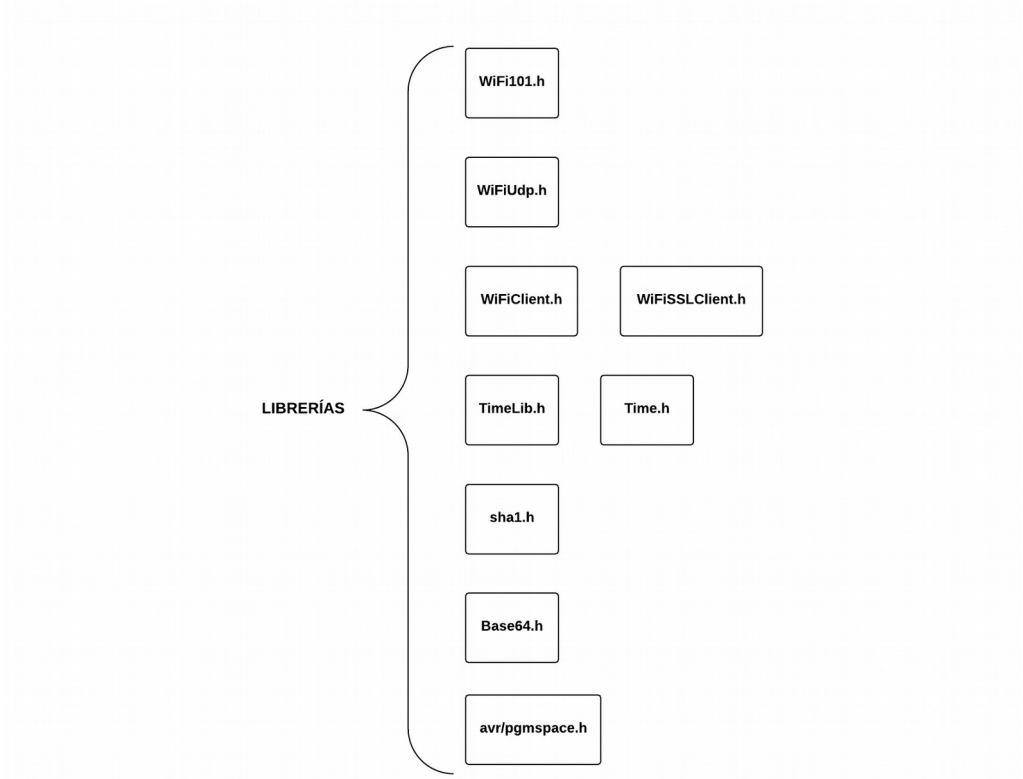

Figura 8. Estructura de librerías utilizadas

Como podemos observar en la figura 8, se hacen uso de librerías ya programadas que proporcionan herramientas ya programadas que facilitan la tarea al programador que hace llamadas a métodos o funciones implementados en la librería correspondiente. La librerías utilizadas son las siguientes:

- WiFi101.h [3]

  Esta librería permite usar funciones para conexión wifi en las placas Arduino MKR1000. También nos permite el envío de paquetes UDP y TCP para el control del flujo de datos. Es necesaria para actualizar el firmware de la placa.

- WiFiUdp.h [4]

  Se necesita para crear instancias WiFiUpd que nos da la posibilidad de envío y recepción de paquetes UDP que usaremos para conectar con el servidor NTP.

- WiFiClient.h [5] – WiFiSSLClient.h [6]

  Estas librerías sirven para poder conectar a un cliente, especificando dirección IP y puerto de conexión. Una conecta con certificados SSL, como su nombre indica, y hay que instalar el certificado correspondiente en la placa Arduino usando WIFI101 Firmware Updater.

- TimeLib.h [7] - Time.h [8]

  Con esta librería podemos manejar muy fácilmente las variables temporales del sistema para extraerlas por separado y darle el formato requerido por el servidor de Latch, que requiere un formato muy concreto: **aaaa-MM-dd HH:mm:ss**

- sha1.h [9]

  Librería criptográfica para poder manejar hashes SHA-1, SHA-256, HMAC-SHA-1 y HMAC-SHA-256. En nuestro caso necesitaremos para generar una firma de una cadena mediante el algoritmo HMAC-SHA-1.

- Base64.h [10]

  Esta librería nos proporciona fácilmente poder codificar strings en formato binario a base64 y viceversa. Ya que el servidor Latch requiere que los datos después de aplicar la firma HMAC-SHA-1 los procesemos en base64.

- avr/pgmspace.h [11]

  Esta librería se utiliza para almacenar datos en la memoria flash del microcontrolador que usaremos para almacenar los strings.

4.1.2 Función Setup( )

Esta función se encarga de hacer las configuraciones en la placa para establecer los pines que van a ser usados de entrada (INPUT) o de salida (OUTPUT). Así como, realizar algunas tareas que sólo es necesario hacerlas una vez como es el caso de establecer una salida **Serial** o conectar con la red Wifi.

4.1.3 Diagrama de flujo de la aplicación Arduino

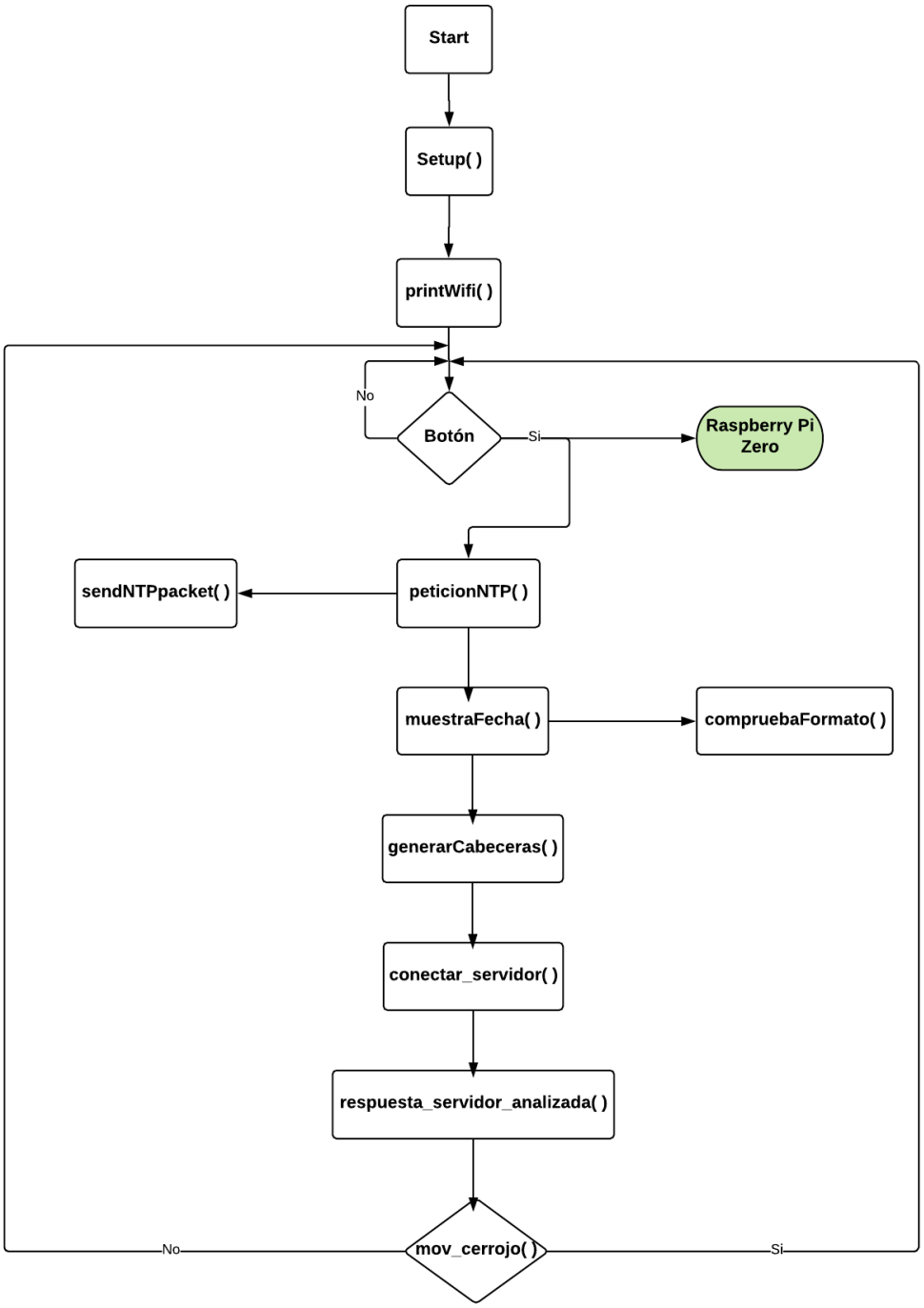

Figura 9. Diagrama de flujo aplicación

Según se puede observar en la figura 9, el diagrama de flujo de la aplicación nos da una visión general para poder ver la estructura del programa.

4.1.4 Funciones del programa

- printWiFiStatus()

La función imprime por pantalla en la salida Serial los siguientes datos de la red WIFI: SSID, IP recibida por el microcontrolador y RSSI la potencia recibida de señal.

- sendNTPpacket()

Esta función recibe como parámetro una dirección IP y envía un paquete UPD en un formato predefinido al puerto 123 del servidor NTP, y recibe un entero sin signo (unsigned long) que devuelve a la línea que la llamó.

- peticionNTP()

La respuesta recibida por la función *sendNTPpacket()* devolvió un valor unsigned long de 32 bytes pero los los datos de los segundos desde 1900 se almacenan en la parte alta de la variable, por lo que recuperamos sólo los 16 bytes de mayor peso. Debemos calcular los segundos desde el 1 Enero de 1970 que son 2208988800 segundos que restaremos al valor recibido. A partir de ese momento tendremos los segundos los segundos actuales hasta el 1 Enero de 1970 que empezaron los sistemas UNIX. Para añadir este dato al sistema usamos función *setTime()*.

- muestraFecha()

Esta función se encarga de extraer del sistema las variables de la hora y fecha para darles el formato requerido por el servidor de Latch.

- compruebaFormato()

El formato requerido es de dos dígitos para cada cada uno de los valores salvo para el año que son cuatro dígitos. Se encarga de poner un “0” delante si el valor está entre 0-9.

- generarCabeceras()

En esta función se preparan las cabeceras que serán enviadas en el paquete TCP al servidor de Latch. Se utiliza un búffer *(MainBuffer)* para ir añadiendo los datos en el orden correspondiente y poder añadirlo directamente a un string.

Se inicia HMAC-SHA-1 con el Secreto obtenido mediante el Script en bash. Se prepara el string que se va a encriptar con todos los parámetros: GET + Date + API + AccountID , y se firma con el Secreto.

Ahora el resultado de HMAC-SHA-1 estará en formato binario, por lo que se codifica en base64, teniendo como resultado la firma.

Luego se preparan las cabeceras que se enviarán: Authorization y 11Paths-Date, en los correspondientes strings: Cabecera1 y Cabecera2.

- conectar_servidor()

Se ha configurado un Reverse Proxy para intermediar en el Handshake SSL de Arduino con el servidor Latch, ya que la capacidad de cálculo de este Arduino MKR1000 es insuficiente para la negociación. El servidor de Latch utiliza un certificado SSL con una longitud de clave de 2048 bits y SHA-256. El proxy se ha configurado en una placa Raspberry Pi Zero HW que dispone de más procesamiento y más recursos para la comprobación de los certificados.

- respuesta_servidor_analizada()

Esta función se encarga de analizar la respuesta del servidor Latch que devuelve un archivo JSON con el estado on/off de la posición del cerrojo virtual de la app móvil de Latch, y que guarda el valor asociado en la cuenta de la aplicación en el servidor de Latch. Una vez rescatado el valor recibido lo devuelve a su llamada.

- mov_cerrojo()

Esta función recibe el valor de la función anterior y según sea on/off actúa sobre un relé y sobre dos leds. La actuación sobre el relé proporcionará tensión a la cerradura para su apertura, mientras que no se le aplique tensión esta permanecerá cerrado. Si la función recibe ON procederá a abrir el cerrojo iluminando el led amarillo y después de 10 segundos procederá a cerrarlo apagando a su vez el led. Por otro lado, si la función recibe OFF significa que no se tiene permiso para la apertura del cerrojo, se iluminará el led rojo durante 3 segundos y no se actuará sobre el relé.

- setup()

En está función se comprueba que el microcontrolador conecte a la conexión wifi que hemos configurado con los parámetros correspondientes en el archivo ***arduino_secrets.h*** adjunto. En cuanto conecte imprime por salida serial la salida de la función antes detallada ***printWiFiStatus()***. Y por último, configura los pines de la placa en modo entrada (INPUT) o salida (OUTPUT) para controlar los led, el pulsador y el relé.

4.1.5 NTP Server [12]

Se decide preguntar a un servidor nacional para realizar una sincronización más exacta ya que la proximidad de los servidores evitará una demora por la distancia. Se emplea la Red Iris ya que es una red académica y de investigación española que proporciona este servicio. La red funciona por nodos en cada una de las comunidades autónomas del país.

Al realizar pruebas mediante la herramienta PING se ha comprobado que el servidor más rápido ha sido el situado en Madrid (~11ms) que comprobamos que según su web que obtiene la hora por GPS. Mientras que la comprobación realizada al servidor de Valencia (~13.5ms) por lo que se deduce que hace preguntas recursivas dentro de su topología, por ello se decide emplear el servidor **hora.rediris.es** (Madrid) con dirección IP: **130.206.3.166**

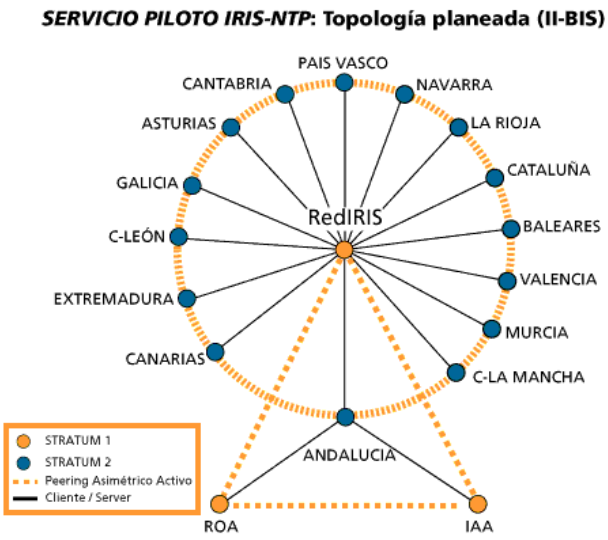

Figura 10. Topología red IRIS-NTP

Como podemos observar en la figura 10, la topología de la red IRIS es radial para proporcionar a cada comunidad autónoma un nodo de forma que no se vea saturado el servidor central.

4.1.6 Conexiones eléctricas circuito con Arduino – Protoboard

El microcontrolador Arduino necesita de una placa protoboard para establecer las conexiones de los componentes que se va a utilizar en el circuito. Esta placa facilita las conexiones ya que introduce un sistema mallado con conexiones internas que permite tener varios componentes interconectados sin tener que tocarse los pines.

También permitirá a Arduino controlar el relé conectado a la cerradura eléctrica para proporcionar control sobre ella. Las conexiones se pueden ver en la siguiente figura:

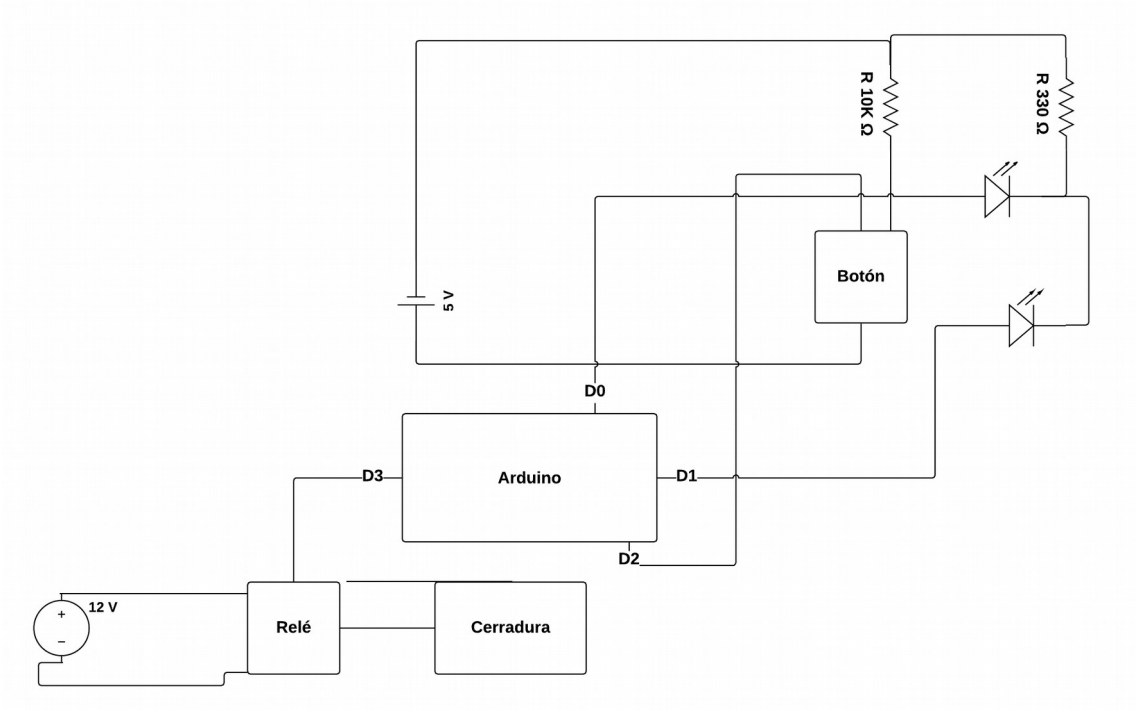

Figura 11. Conexión eléctrica Arduino – Placa Protoboard

4.1.7 Funcionamiento global aplicación Arduino

El funcionamiento global de la aplicación se basa en el bucle infinito del bucle ***loop( )*** en el cuál se hace un llamamiento al resto de las funciones del código.

El principal factor de la aplicación es que se encuentra a la espera de la pulsación del botón (a modo de timbre). En el momento de la pulsación se generará una petición de la hora del servidor NTP para actualizar la hora de la placa, y generar un ***timestamp***. Este sello temporal es importante para la generación de las cabeceras en la petición del estado de la aplicación de Latch. Así no se puede replicar la petición más tarde porque va asociada al momento temporal de la petición.

Luego, se generarán las cabeceras de la petición. Estas se componen de dos cabeceras con unos valores concretos en un orden concreto. La función correspondiente ***generarCabeceras( )*** se encargará de ello.

Con los datos de las cabeceras se procederá a enviar la ***petición http*** al proxy de nuestra red local compuesto por una placa Raspberry Pi Zero que se encargará de redirigir las peticiones al servidor de Latch. Es necesaria para poder realizar la función de intermediación en la negociación del Handshake SSL. Al recibir la respuesta, el proxy la redirigirá a nuestro Arduino MKR1000.

La función recogerá la respuesta, se analizará y procederá a actuar sobre los leds y sobre el relé, que a su vez, alimentará de tensión o no al cerrojo para la apertura de éste.

4.2 Creación aplicación como desarrollador en Latch

Accedemos a https://latch.elevenpaths.com y pinchamos en la pestaña "***Área de Desarrolladores"***, nos llevará a la página de la figura 12 donde pulsaremos sobre el recuadro que contiene ***"Registrarse como Desarrollador"*** para darnos de alta:

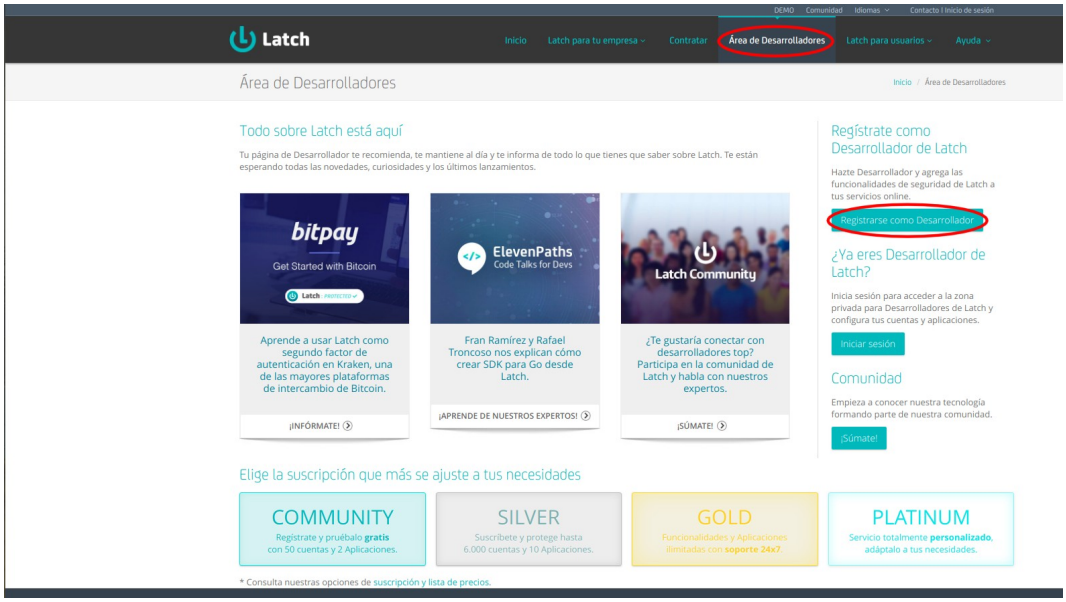

Figura 12. Registrar cuenta desarrollador en Latch

Nos aparecerá el siguiente formulario para darnos de alta y debemos aceptar el contrato. En el caso de este TFG se va a proceder a crear cuenta gratuita que limita a 2 el número de aplicaciones y un máximo de 50 cuentas pareadas (entre las 2 aplicaciones). El formulario se puede observar en la figura 13:

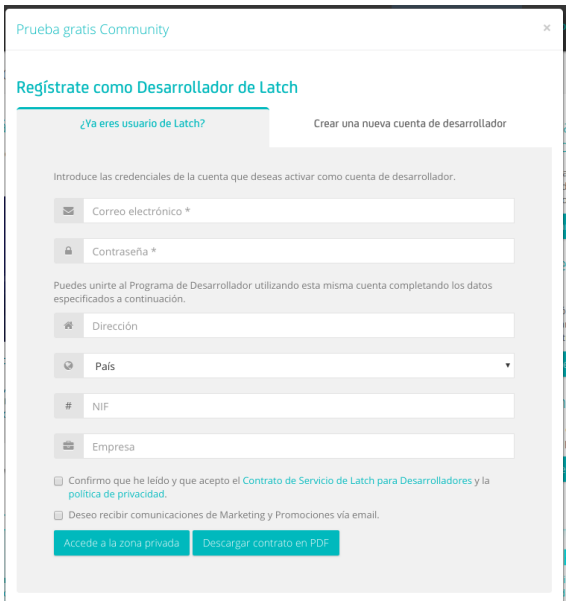

Figura 13. Formulario desarrollador

Una vez tengamos estos pasos podremos acceder a nuestra área de desarrollador y crear una aplicación dentro de nuestro espacio de desarrollador. Accedemos con nuestra cuenta al dashboard, en la pestaña “Mis aplicaciones” pulsamos “Añadir una nueva aplicación” como podemos ver en la figura 14:

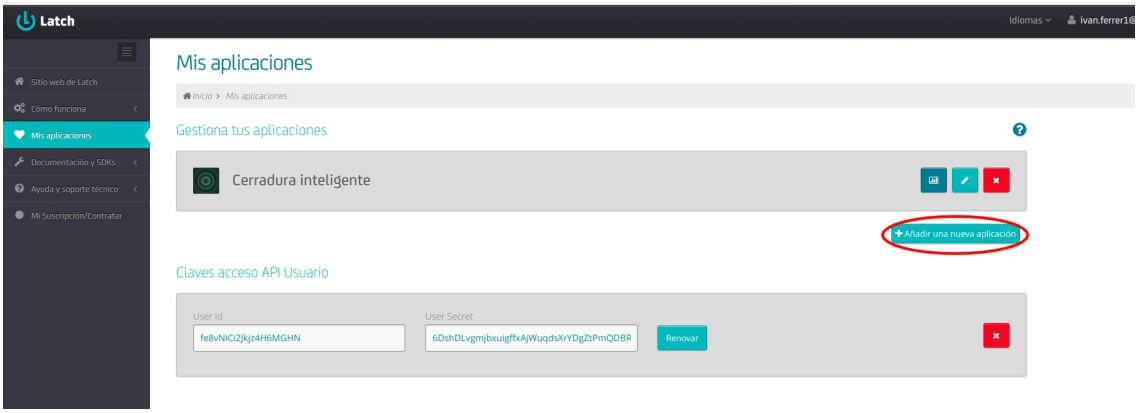

Figura 14. Creación de nueva aplicación

Accederemos a la siguiente pantalla para poner el nombre de nuestra aplicación, y así la crearemos definitivamente:

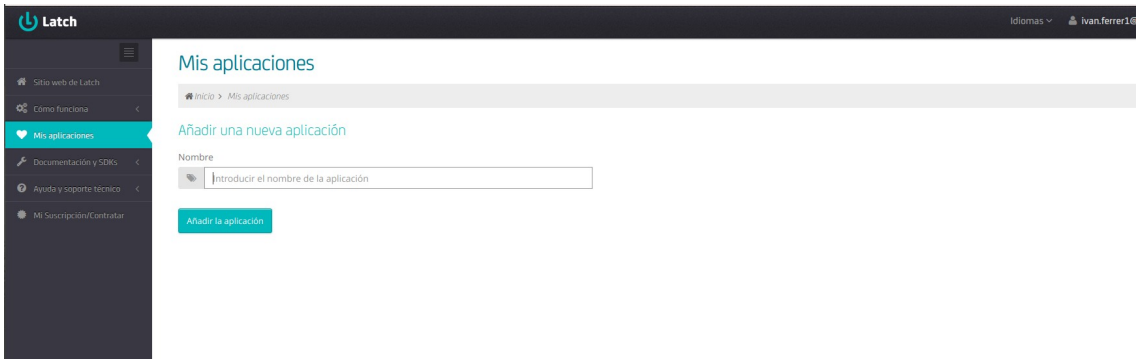

Figura 15. Nombre de la aplicación

A partir de ahí, tendremos la aplicación con los datos requeridos para conectarla con nuestra aplicación de Arduino y la app móvil: ***ID de aplicación*** y ***Secreto***

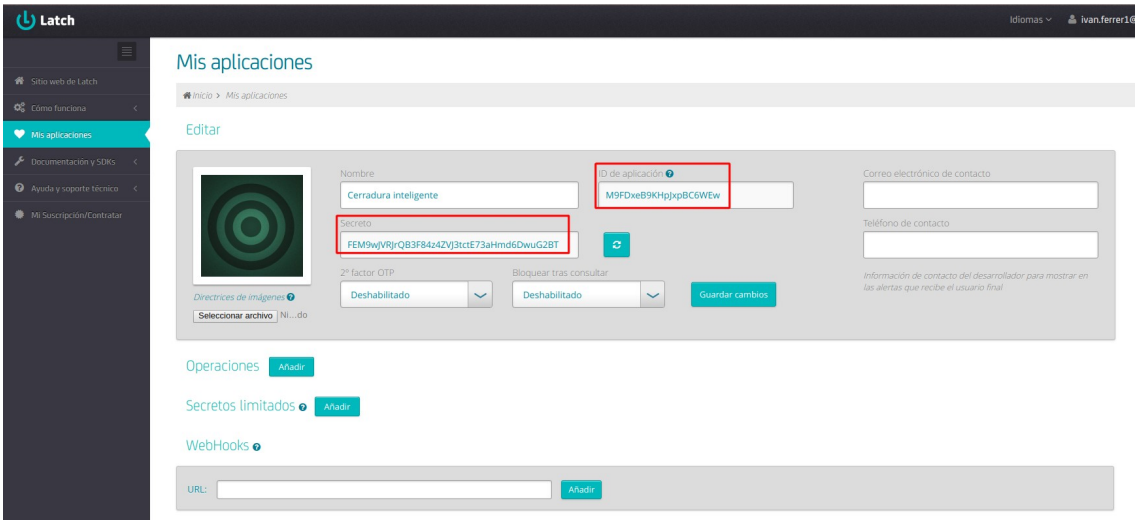

Figura 16. Panel de control de la aplicación de Latch

4.3 App móvil Latch

Una vez registrado en la app de Latch y obtenido una cuenta, añadiremos un nuevo servicio como podemos ver en la figura 17. En la figura 18, pulsamos el botón “Parear con Latch” para asociar con la aplicación que creamos en la web:

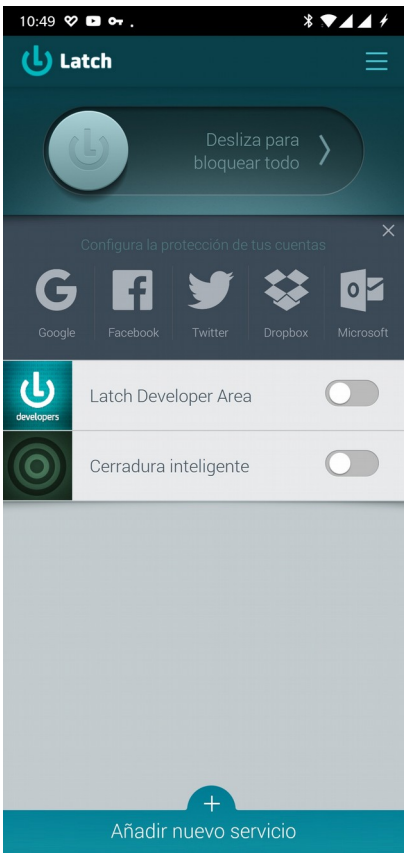

Figura 17. Añadir nuevo servicio

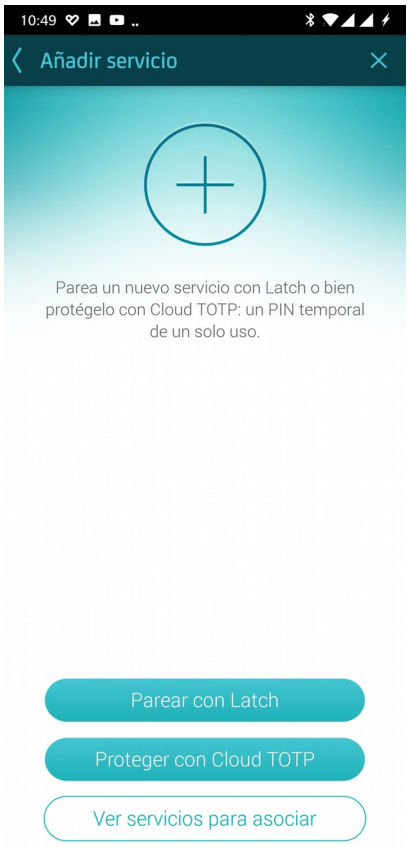

Figura 18. Parear con Latch

En la figura 19, pulsamos “Generar nuevo código” para la creación de un código temporal de asociación (pareado). En la figura 20, obtenemos un código de 6 caracteres que tiene un período de validez de 120 segundos, el cuál debemos usarlo en este tiempo junto con un script para obtener el “accountID” en formato JSON.

Figura 19. Generación código de pareado

Figura 20. Código temporal de pareado

* Nota: el código generado lo necesitamos junto con el script.

4.4 Script pareado servicio web – App móvil

En primer lugar debemos instalar la app móvil **Latch** disponible en Google Play Store. Deberemos crearnos una cuenta con nuestros datos y, una vez la tengamos vamos a necesitar varios datos de nuestra aplicación creada en la cuenta de desarrollador:

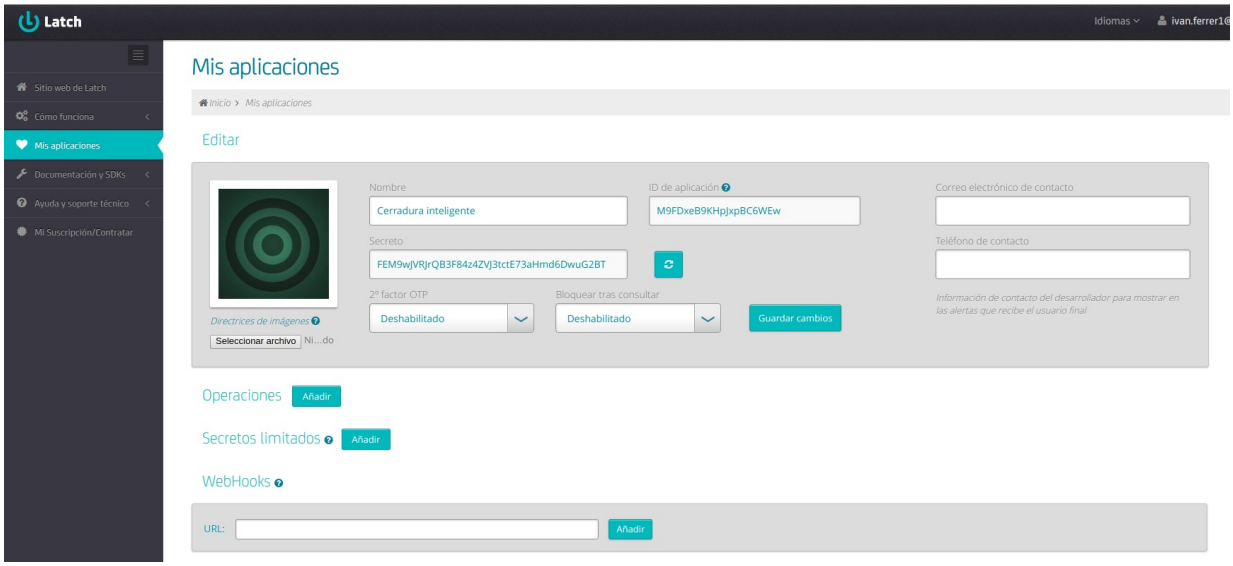

Figura 21. Dashboard de la aplicación “Cerradura Inteligente”

En la figura 22, se puede observar como se ha de editar el script de pareado y los valores que hay que completar del archivo ***pareado.sh*** y rellenaremos con estos datos las variables: ***ApplicationId*** y ***SecretKey***.

```
pareado.sh
1  #!/bin/bash
2
3  if [ -z "$1" ]; then
4   echo -e "\nUsage: $0 \n"
5   exit 0
6  fi
7
8  ApplicationId="M9FDxeB9KHpJxpBC6WEw"
9  SecretKey="FEM9wJVRJrQB3F84z4ZVJ3tctE73aHmd6DwuG2BT"
10
11 Server="https://latch.elevenpaths.com"
12 URL="/api/1.3/pair/$1"
13
14 requestSignature+="GET\n"
15 date=`date -u '+%Y-%m-%d %H:%M:%S'`
16 requestSignature+="$date\n\n$URL"
17 signed=`echo -en "$requestSignature" | openssl dgst -sha1 -hmac "$SecretKey" -binary`
18 b64signed=`echo -n "$signed"|base64`
19
20 auth_header="Authorization: 11PATHS $ApplicationId $b64signed"
21 date_header="X-11Paths-Date: $date"
22
23 response=`curl -q -s -N --header "$auth_header" --header "$date_header" "$Server$URL"`
24
25 echo $response
26
```

Figura 22. Script pareado.sh

Para ejecutar el script debemos escribir en una terminal el siguiente comando junto con el código temporal obtenido con la app móvil antes de que caduquen los 120 segundos:

$ ./pareado.sh Pg9Qid

*El resultado obtenido será un JSON con el valor de "**accountId**", el cuál necesitaremos para nuestra aplicación en Arduino*:

{"data":
{"accountId":"U3BC26u6vT3paRjcBu8zvTMEF7JQyR4QFx296uFj8gt2ButsFRjaQNGC9Zi7v4Ta"}}

4.5 Reverse proxy con Raspberry Pi Zero HW

4.5.1 Instalación y configuración de Raspbian

- Descarga del S.O Raspbian [13]

Vamos a descargar el Sistema Operativo (S.O) Raspbian que está basado en una versión adaptada de Debian para placas Raspberry. En nuestro caso vamos a utilizar una placa Raspberry Pi Zero W [15]. Elegimos esta placa dado que tiene wifi, un sistema operativo completo y sus recursos son muy superiores a los del microcontrolador Arduino MKR1000. Este dispositivo hará de intermediario en nuestra red local por lo que sólo se encargará de retransmitir peticiones http y sus respuestas entre la placa Arduino y el servidor de Latch, debido a esto se elige la versión **Raspbian Stretch Lite** que no dispone de interfaz de escritorio, únicamente acceso por terminal.

- Escritura de imagen (.img) en tarjeta MicroSD [14]

Al descargar el archivo del S.O, descomprimimos el archivo .zip obteniendo una imagen .img. Necesitaremos grabar la imagen en una tarjeta MicroSD, a poder ser de clase10.

Para la grabación se puede realizar mediante un software con interfaz gráfica (Etcher) aunque en nuestro caso se procederá por terminal con el siguiente comando desde la carpeta donde se encuentre el archivo .img:

$ sudo dd bs=1M if=2018-06-27-raspbian-stretch-lite.img of =/dev/mmcblk0

Tardará un rato ya que se va a copiar por sectores de 1MB para evitar fallos. Ya estará preparada la tarjeta MicroSD para introducir en la placa.

- Primer arranque

Para conectar y configurar la placa necesitaremos de disponer de monitor y teclado conectados a ella. Una vez cargue todo el sistema nos mostrará al final del proceso el prompt de línea de comandos quedando a la espera de órdenes:

pi@raspbian:~$

- Configuración de la placa [16]

Para acceder a la configuración de la placa introduciremos el siguiente comando:

$ sudo raspi-config

Ésta orden nos llevará a un menú con una serie de opciones:

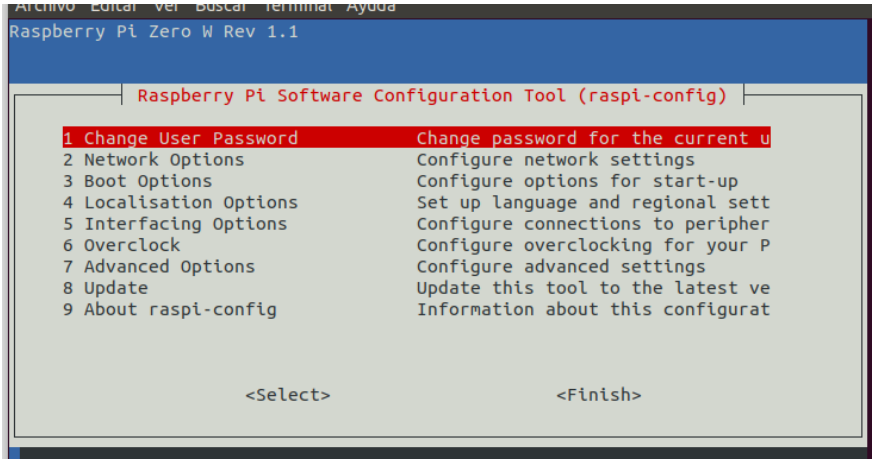

Figura 23. Menú configuración Raspbian

- Expand Filesystem → es importante expandir las particiones del sistema operativo para que ocupe toda la capacidad de la tarjeta MicroSD.
- Change User Password → la placa va a estar conectada a Internet y es importante cambiar la contraseña por defecto del usuario pi.
- Network Options — Hostname → introduciremos el nombre de red del dispositivos
  Wifi → introduciremos el SSID y Password de la red Wifi
- Boot Options — Desktop/CLI → elegimos CLI (command Line Interface)
  Wait Network at Boot → No
- Localisation — Change Locale → es_ES.UTF8 UTF8
  Change Timezone → Europe/Madrid
  Keyboard → 105 Key Generic/Spanish-Catalan/Right Alt (AltGr)/Left Logo Key
  Wifi Country → ES Spain
- Interfacing Options — Camera → Yes
  SSH → Yes
  SPI → Yes
  I2C → Yes
  Serial → Yes

A partir de este momento ya podemos salir de la configuración seleccionando ***Finish***. Hemos activado SSH por lo que podremos dejar la placa sin monitor ni teclado después del próximo reinicio. Ella sola se conectará al punto wifi que hemos configurado y se quedará a la espera de una conexión por SSH. Para reiniciar escribiremos:

$ sudo reboot -f

Después de unos segundos de reinicio y carga del sistema únicamente se necesitará saber la dirección IP que le ha asignado el DHCP del router para acceder por ssh, el usuario será ***'pi'.*** Desde cualquier PC/portátil podremos usar un software como PuTTY o escribir desde línea de comandos:

$ ssh pi@192.168.1.10

Nos pedirá la contraseña que cambiamos anteriormente en el menú de configuración, y a partir de ahí tomaremos el control de la terminal desde nuestro PC para seguir más cómodamente con el proceso de configuración del sistema.

- Actualización firmware y sistemas

Es necesario actualizar el firmware de la placa Raspberry para corregir errores y nuevas funciones. Siempre se ha de mantener actualizado, por lo que se podría hacer un script programado cada cierto tiempo con cron, pero vamos a proceder a hacerlo manualmente con el siguiente comando:

$ sudo rpi-update

El siguiente paso será actualizar el kernel y repositorios del S.O con el siguiente comando:

$ sudo apt-get update | sudo apt-get upgrade -y | sudo apt-get dist-upgrade -y

Tardará un tiempo dependiendo de la velocidad de conexión y del tiempo de instalación de los paquetes en la placa. Reiniciaremos para que carguen los nuevos paquetes y nuevo kernel del sistema:

$ sudo reboot -y

Ahora procedemos a eliminar paquetes viejos y dependencias que no necesitemos porque se hayan actualizado, así como kernels antiguos que pudieran haber:

$ sudo apt-get autoremove -y | sudo apt-get autoclean -y

- Instalación Nginx

Se procede ahora a instalar un servidor web que consuma pocos recursos del sistema que garantizará rápidas y múltiples conexiones por si en un futuro ampliamos el sistema con más sensores de puertas u otros dispositivos. Se encuentra en los repositorios de Raspbian, por lo que únicamente debemos introducir lo siguiente:

$ sudo apt-get install nginx

- Configuración Nginx [17]

Crearemos un nuevo servicio dentro la carpeta "***/etc/nginx/sites-available***" llamado "***latch***". El servicio de redirección lo vamos a colocar en el puerto específico ***8080*** de modo que únicamente retransmitirá las peticiones al servidor de Latch que le sean enviadas a dicho puerto. Quedando el servicio como la figura 24:

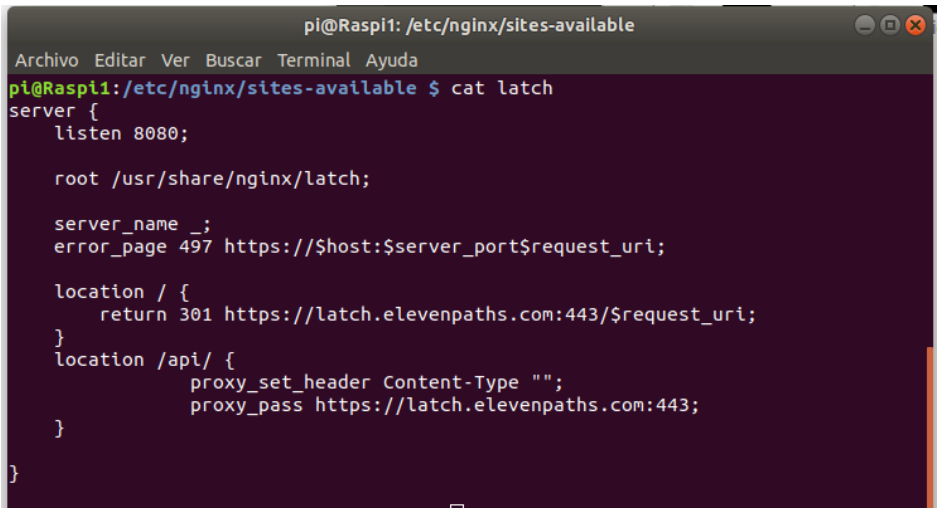

Figura 24. Configurar servicio proxy en Nginx

Este servicio retransmite la petición completa únicamente cambiando el servidor y el puerto: ***https://latch.elevenpaths.com:443*** el resto de cabeceras no se tocarán.

Para que el servicio quede activo de todo, ha de configurarse en la carpeta "***/etc/nginx/sites-enabled***" de forma que para no tener 2 archivos que puedan tener versiones diferentes, se va a proceder a realizar un enlace simbólico desde la carpeta "***/...enabled***" que apunte al archivo de la carpeta "***/...available***" con el siguiente comando:

$ sudo ln -s /etc/nginx/sites-available/latch /etc/nginx/sites-enabled/latch

Si hacemos un listado extendido de la carpeta "/etc/nginx/sites-enabled" comprobaremos que el servicio Latch apunta a la carpeta anterior:

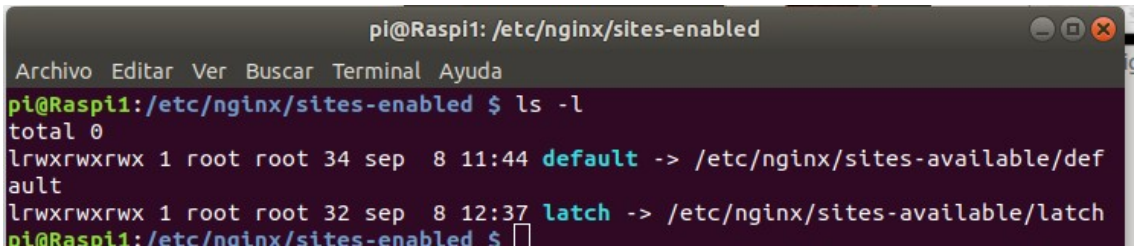

Figura 25. Comprobación sites-enabled

Ahora debemos reiniciar el servidor Nginx para que los cambios tengan efecto:

$ sudo service nginx restart

Abrimos un navegador web desde el PC y comprobamos que el proxy hace su función correspondiente, introducimos la siguiente URL: " ***ip_raspberry : 8080*** " y nos tiene que redirigir a la página web de https://latch.elevenpaths.com

4.6 Conexión con Picamera [18]

4.6.1 Conexión serie Arduino – Raspberri Pi Zero

Necesitaremos que haya comunicación entre la placa Arduino y la placa Raspberry, para ello usaremos la conexión serie que nos proporcionará el cable USB. De esta forma podremos ahorrarnos la fuente de alimentación de Arduino ya que la alimentará el puerto USB de la Raspberry.

La comunicación será unidireccional iniciada por la salida por consola serie de Arduino, y mediante un script en Pichona recogeremos y analizaremos para detectar el momento de pulsación del botón implementado en el código de nuestro programa Arduino. La detección de la pulsación del botón es con anterioridad al envío de la petición de estado a los servidores de Latch, para que de tiempo a iniciar la emisión de la cámara en streaming.

Necesitaremos instalar en la placa Raspberry los siguientes paquetes:

$ sudo apt-get install python python-pip python-dev build-essential

Y también deberemos instalar la librería siguiente en Python: [19]

$ pip install pyserial

4.6.2 Script Python 'lecturaSerial.py' [20]

El código de este script se encuentra en el **Anexo III**.

Con este script se busca la apertura de un puerto serie comunicando las dos placas con el mismo reloj establecido en 9600 baudios. Mediante este puerto, se crea un buble infinito de lectura para recibir la salida de datos desde la placa Arduino. El script lee cada una de las líneas recibidas hasta encontrar una en concreto que activa el streaming de video. Lo mantiene activo 120 segundos, tiempo suficiente para apertura de la aplicación móvil de visualización del streaming, y pasado este tiempo matará el proceso asociado a la emisión del video. Volviendo a quedar a la espera de todo lo recibido por la placa Arduino.

4.6.3 Instalación y configuración librería 'Motion' [21]

Esta librería nos va a proporcionar la opción de automatizar la captura de imágenes y video de la cámara y publicarlas en un servidor web para que podamos visualizar quien ha pulsado el botón y está intentando acceder a la apertura de la cerradura.

Para instalar el paquete *Motion* escribiremos:

$ sudo apt-get install motion

Ahora necesitaremos configurar una serie de opciones en su archivo de configuración que se encuentra en la siguiente ruta: /***etc/motion/motion.conf***. Lo editaremos con el siguiente comando:

$ sudo nano /etc/motion/motion.conf

Y modificaremos las siguientes líneas para que quede de la forma siguiente:

width 640	***locate_motion_mode on***
height 480	***locate_motion_style redbox***
stream_port 8081	***webcontrol_port 8082***

stream_quality 75

webcontrol_localhost off

stream_localhost off

v4l2_palette 8

stream_maxrate 100

framerate 100

ffmpeg_output_movies off

Básicamente en las opciones anteriores configuraremos el tamaño del video, pintaremos un recuadro rojo en la parte de la pantalla donde haya movimiento e indicamos los puertos en los que el servidor web va a emitir el video, dando acceso a visualizarlo desde la IP de nuestro móvil.

Una vez configurado, se inicia la emisión con la orden del script **lecturaSerial.py**.

La emisión del streaming de video de la cámara se realizará desde la ip de la placa Raspberry en el puerto 8081, mientras que también tendremos la posibilidad de controlar algunos parámetros durante la emisión desde el puerto 8082 que nos brindará un panel de control.

4.6.4 App Android visualización streaming video

Necesitaremos una aplicación desde la que visualizar el streaming de video con nuestro dispositivo móvil, para ello he seleccionado una app llamada **IP Cam Viewer Lite** que se puede descargar desde el siguiente enlace del Google Play Store:

https://play.google.com/store/apps/details?id=com.rcreations.ipcamviewer

En la aplicación entraremos en la opción ‘Gestionar cámaras’ y pulsaremos el botón + para añadir una nueva cámara. Rellenaremos las siguientes opciones como muestra la siguiente figura 26:

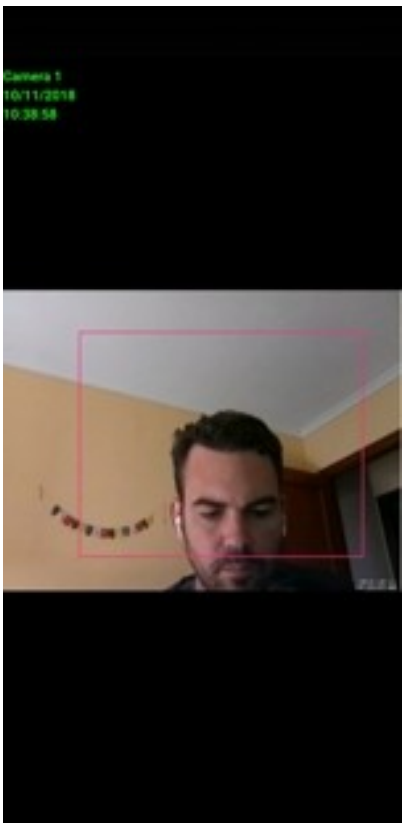

Figura 26. Añadir cámara

Figura 27. Visualización streaming

En la figura 27, podemos ver como se visualiza el video, con un rectángulo rojo en la zona de la imagen que detecta movimiento, y como aparece una marca temporal con el nombre de la cámara.

Capítulo 5. Funcionamiento de la aplicación

En funcionamiento de sistema de la cerradura inteligente parte de un estado en reposo a la espera de la pulsación de un botón (timbre) implementado con la placa Arduino. En el instante de pulsación:

- El microcontrolador **Arduino** hará una petición al servidor NTP para actualizar el reloj interno y usar como sello temporal para el envío de la petición de estado del pestillo asociado a la aplicación de la cerradura en la app móvil de Latch. El programa Arduino generará las cabeceras y el formato adecuado para realizar la petición de estado. Una vez generado enviará la petición a la placa Raspberry para que establezca comunicación con el servidor Latch a través del proxy. Una vez recibida la respuesta de estado del pestillo Latch asociado, el microcontrolador actuará sobre el relé al que está conectada la cerradura.

- La placa **Raspberry Pi Zero** se encargará de dos tareas principalmente:

  - **Reverse Proxy**: realiza intermediación en la comunicación https con el servidor Latch ya que por si mismo Arduino no es capaz.
  - **Streaming video**: pequeño script escrito en python recoge la información recibida del puerto serie que le envía Arduino, y analiza la respuesta. En caso de que no se permita la apertura de la cerradura, se iniciará la emisión del streaming de video para visualizar a la persona que ha presionado el timbre de la puerta. La emisión de video durará 2 minutos.

- **App móvil Latch**, con la aplicación móvil el usuario será capaz de permitir o denegar la apertura de la cerradura, siendo alertado mediante una notificación del intento de apertura cuando se haya configurado para el bloqueo.

- **App móvil IP Cam Viewer**, con esta aplicación móvil el usuario tiene la capacidad de visualizar en tiempo real un streaming de video de la persona que ha pulsado el botón.

Capítulo 6. Conclusiones

Realizando este proyecto me he enfrentado diferentes dificultades por falta de información, principalmente en la conexión de Arduino con el servidor Latch, siendo imposible de establecer la conexión SSL. Agradecer a uno de los desarrolladores de Latch que me proporcionó un servidor de pruebas sin HTTPS, que no viene en la documentación, y poder seguir desarrollando el sistema hasta poder investigar y añadir un proxy.

Gracias a la realización de esta aplicación he ampliado conocimientos sobre Arduino para seguir desarrollando e introducirme en el mundo "Maker" y poder implementar nuevas ideas. También he despertado mucha curiosidad por el desarrollo con Python, descubriendo un lenguaje de scripting muy potente que me ha permitido interconectar las dos placas para ampliar las posibilidades.

El sistema desarrollado permite una utilidad dentro de una LAN por parte de un usuario, aunque tengo pensadas diferentes posibilidades de ampliación que no se han incorporado en este proyecto, como:

- **Uso Firewall**

Los S.O Linux llevan un firewall incorporado, UFW, con el que podríamos controlar tráfico indebido a la placa Raspberry. Mediante el uso de reglas de paquetes entrantes y paquetes salientes, podemos aceptar/rechazar paquetes dependiendo del tipo de protocolo y desde el rango de IP que queramos filtrar.

- **Uso VPN**

El uso de VPNs hoy en día está muy recomendado, sirven para crear una red LAN, a través de diferentes redes conectadas a Intenet, entre los dispositivos que tengan instalado el cliente VPN y el certificado correspondiente para encriptar el tráfico entre ellos gracias a un servidor VPN.

- **Uso DynDNS**

En las redes LAN domésticas lo normal es disponer de una IP pública dinámica, de forma que no siempre se dispone de la misma IP, complicando el acceso a la red LAN desde Internet. Servicios como por ejemplo: www.dyndns.com o www.noip.com permiten comprobar la IP

pública que está utilizando el router de la red LAN y asociarla a un dominio previamente configurado para ello.

Capítulo 7. Bibliografía

[1] https://ciberseguridad.blog/recomendaciones-de-ciberseguridad-en-iot/
http://www.blog-idcspain.com/mercadoiotenespana/
https://www.ithinkupc.com/blog-es/recomendaciones-de-seguridad-en-dispositivos-iot
https://aunclicdelastic.blogthinkbig.com/el-reto-de-la-seguridad-en-iot/

[2] https://latch.elevenpaths.com

[3] https://www.arduino.cc/en/Reference/WiFi101

[4] https://www.arduino.cc/en/Reference/WiFi101UDPConstructor

[5] https://www.arduino.cc/en/Reference/WiFiClient

[6] https://www.arduino.cc/en/Reference/WiFi101SSLClient

[7] https://github.com/PaulStoffregen/Time/blob/master/TimeLib.h

[8] https://playground.arduino.cc/Code/Time

[9] https://github.com/Cathedrow/Cryptosuite

[10] https://github.com/adamvr/arduino-base64

[11] http://www.nongnu.org/avr-libc/user-manual/group__avr__pgmspace.html

[12] https://www.rediris.es/gt/iris-ntp/drafts/

[13] https://www.raspberrypi.org/downloads/raspbian/

[14] https://www.raspberrypi.org/documentation/installation/installing-images/linux.md

[15] https://www.raspberrypi.org/products/raspberry-pi-zero-w/

[16] https://www.raspberrypi.org/documentation/configuration/raspi-config.md

[17] https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_pass

[18] https://www.raspberrypi.org/documentation/raspbian/applications/camera.md

[19] https://pythonhosted.org/pyserial/shortintro.html#opening-serial-ports

[20] https://www.python.org/doc/
http://www.mclibre.org/consultar/python/index.html

[21] https://motion-project.github.io/motion_config.html#The_Config_Files

[22] https://blog.elevenpaths.com/2015/10/latch-y-el-internet-de-las-cosas_26.html

[23] https://latch.elevenpaths.com/www/developers/doc_api

Capítulo 8. Anexos

8.1 Anexo I. Código script "pareado.sh" [22]

```
#!/bin/bash
if [ -z "$1" ]; then
 echo -e "\nUsage: $0 \n"
 exit 0
fi
ApplicationId="M9FDxeB9KHpJxpBC6WEw"
SecretKey="FEM9wJVRJrQB3F84z4ZVJ3tctE73aHmd6DwuG2BT"
Server="https://latch.elevenpaths.com"
URL="/api/1.3/pair/$1"
requestSignature+="GET\n"
date=`date -u '+%Y-%m-%d %H:%M:%S'`
requestSignature+="$date\n\n$URL"
signed=`echo -en "$requestSignature" | openssl dgst -sha1 -hmac "$SecretKey" -binary`
b64signed=`echo -n "$signed"|base64`
auth_header="Authorization: 11PATHS $ApplicationId $b64signed"
date_header="X-11Paths-Date: $date"
response=`curl -q -s -N --header "$auth_header" --header "$date_header" "$Server$URL"`
echo $response
```

8.2 Anexo II. Documentación API Latch [23]

Todas las solicitudes a la API de Latch deben estar firmadas. El proceso de firma es una versión simplificada del protocolo Oauth de dos vías.

Cada solicitud HTTP a la API debe ir acompañada de dos encabezados de autenticación: **Authorization** y **Date**.

El encabezado Authorization

El encabezado Authorization debe tener el formato siguiente:

11PATHS requestId requestSignature

requestId puede ser o bien un 'applicationId' o un 'userId', dependiendo de la API a la que acceda la petición.

11PATHS es una constante que determina el método de autenticación.

ApplicationId es un identificador alfanumérico que se obtiene al registrar una aplicación nueva.

RequestSignature es una firma derivada de la url, parámetros, encabezados personalizados y fecha de la solicitud actual, todos ellos con hash utilizando un Secreto que también se obtiene mediante el applicationId al registrar la aplicación.

La firma de solicitud es una cadena codificada en Base64 y con firma HMAC-SHA1. La cadena se debe crear siguiendo este proceso.

1.Empezar por una cadena vacía.

2.Anexar el nombre de método en mayúsculas. Actualmente, solo se admiten los valores **GET**, **POST**, **PUT** y **DELETE**.

3.Anexar un separador de línea, " " (Punto Unicode U+000A).

4.Anexar una cadena con la fecha actual en este formato exacto **aaaa-MM-dd HH:mm:ss**. Todo en este formato debe explicarse por sí solo, ten en cuenta que todo es numérico y se debe completar con ceros en caso necesario para que todos los números tengan dos dígitos, excepto el año, que tiene cuatro. Este valor se debe mantener para utilizarlo en el encabezado **Date**. El proceso de comprobación de firma fallará si no coinciden ambos.

5.Anexar un separador de línea, " " (Punto Unicode U+000A).

6.Serializar todos los encabezados específicos de esta aplicación (no cada encabezado HTTP de la solicitud). Todos los nombres de estos encabezados empiezan por **X-11paths-**.

1.Convertir todos los nombres de encabezados a minúsculas.
2.Ordenar los encabezados por nombre de encabezado en orden alfabético ascendente.
3.Para cada valor de encabezado, convierte los encabezados de varias líneas en una única línea sustituyendo los caracteres de línea nueva " " por espacios sencillos " ".
4.Crear una cadena vacía. A continuación, para cada encabezado después de la ordenación y transformaciones anteriores, añadir a la cadena recién creada el nombre de encabezado seguido de dos puntos ":" y del valor de encabezado. Cada nombre:valor debe estar separado del siguiente por un espacio sencillo " ".

5.Recortar la cadena para asegurarse de que no contenga ningún carácter de espacio al inicio o al final.

7.Anexar un separador de línea, " " (Punto Unicode U+000A).

8.Anexa la cadena de consulta con codificación URL que consta de la ruta (empezando por la primera barra inclinada) y los parámetros de consulta. La cadena de consulta no debe contener el nombre de host o el puerto y no debe contener ningún carácter de espacio como prefijo o sufijo.

9.Solamente para peticiones POST o PUT, anexar un separador de línea, " " (Punto Unicode U+000A).

10.Solamente para peticiones POST o PUT, serializar los parámetros de la petición de la siguiente forma, siendo el nombre del parámetro y su valor, la representación en UTF-8 de su codificación URL.

1.Ordenar los parámetros por nombre de parámetro en orden alfabético ascendente y a continuación por valor de parámetro.
2.Crear una cadena vacía. A continuación, para cada parámetro después de la ordenación, añadir a la cadena recién creada el nombre de parámetro seguido de un signo igual "=" y del valor del parámetro. Cada nombre=valor debe estar separado del siguiente por un ampersand "&".
3.Recortar la cadena para asegurarse de que no contenga ningún carácter de espacio al inicio o al final.

Una vez que se haya creado la cadena siguiendo el proceso descrito más arriba, se debe firmar mediante el algoritmo **HMAC-SHA1** y el **secreto** obtenido al registrar la aplicación. Después de la firma, los datos binarios sin procesar se deben codificar en **base64**. La cadena resultante es la cadena **requestSignature** que añadir al encabezado Authorization.

El encabezado X-11Paths-Date

El encabezado **X-11Paths-Date** contiene el valor de la fecha UTC actual y debe tener el siguiente formato:

yyyy-MM-dd HH:mm:ss

donde yyyy es el año, MM es el número del mes, dd es el número del día, HH es la hora en formato de 24 horas, mm son los minutos y ss los segundos. Todos los valores se deben completar con ceros para que tengan valores de dos dígitos, excepto el año, que tiene cuatro.

Es muy importante que el valor y el formato de este encabezado sean exactamente los mismos que los utilizados en el proceso de creación de **requestSignature** para el encabezado de autorización como se explica más arriba.

Ten en cuenta que puedes seguir utilizando el encabezado HTTP **Date** estándar en el formato que desees, por ejemplo RFC 1123. Solo asegúrate de no confundir ambos y de utilizar siempre el valor que utilices en **X-11Paths-Date** en el proceso de firma. La API ignorará el encabezado **Date** estándar.

Errores

La siguiente es una lista de condiciones de error posibles que pueden ocurrir durante la autenticación:

Error 101: Invalid Authorization header format

Error 102: Invalid application signature

Error 103: Authorization header missing

Error 104: Date header missing

Error 108: Invalid date format

Error 109: Request expired, date is too old

Error 111: User not authorized

Error 112: Invalid user signature

Error 113: Secret signing this request is not authorized to perform this operation

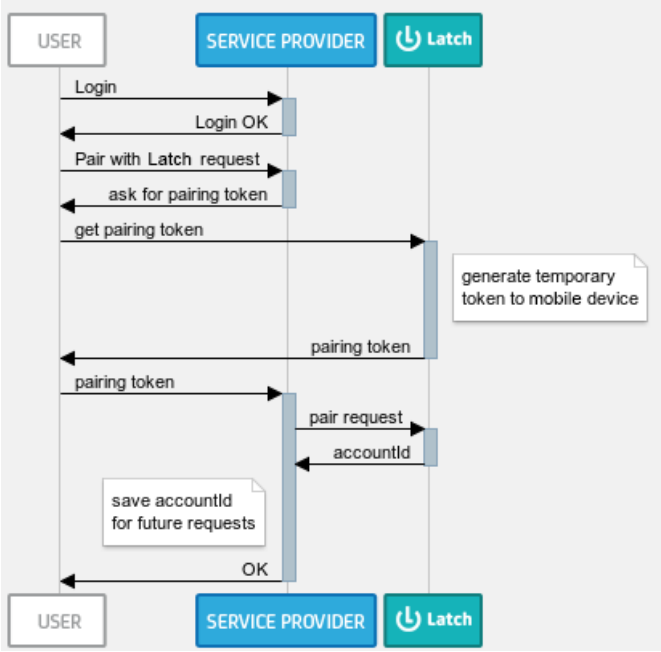

Figura 28 . Proceso 'Parear cuenta'

Solicitud:

Para parear con la API de token, lleva a cabo una solicitud HTTP GET en el siguiente extremo:

https://latch.elevenpaths.com/api/1.3/pair/{token}

donde **token** es un token alfanumérico de seis caracteres de longitud que se debe obtener del usuario que, a su vez, lo ha obtenido mediante la app móvil.

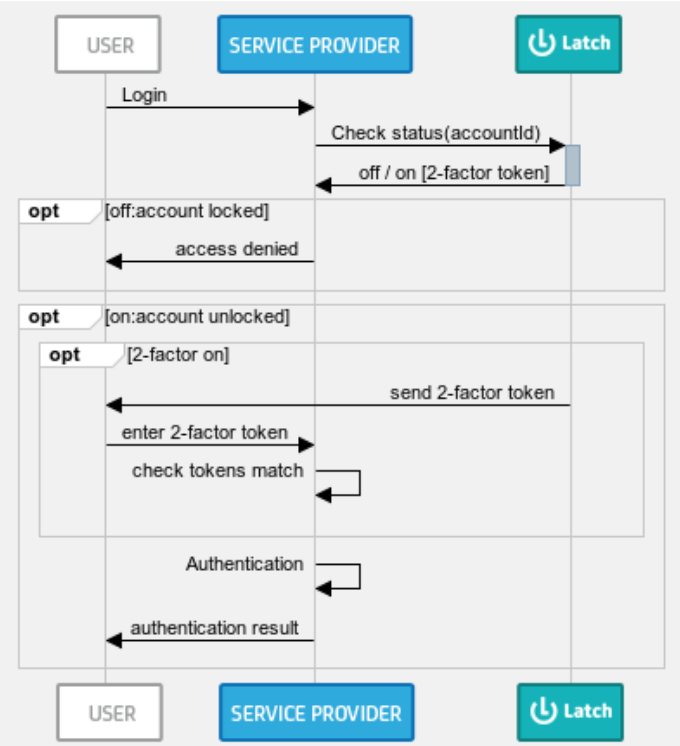

Figura 29. Proceso 'Estado de la cuenta'

Autenticación:

Para utilizar esta solicitud tienes que autenticar tu app utilizando tu applicationId y firmarlo como se explica en la sección de autenticación.

Solicitud:

Para utilizar la API de estatus, lleva a cabo una solicitud HTTP GET en el siguiente extremo:

https://latch.elevenpaths.com/api/1.3/status/{accountId}

donde **accountId** es un token alfanumérico de 64 caracteres de longitud que se puede utilizar para identificar de forma única la cuenta cuyo estado se está consultando, que se ha obtenido mediante el método de pareado.

8.3 Anexo III. Código script ‘lecturaSerial.py’

```
#!/usr/bin/python
# _*_ coding: utf-8 _*_

# Importamos librerias
import serial
import shlex, subprocess
import time

# Abrimos el puerto del arduino a 9600
PuertoSerie = serial.Serial('/dev/ttyACM0', 9600)
# Creamos un buble sin fin
while True:
    # leemos hasta que encontarmos el final de linea
    line = PuertoSerie.readline()
    # Mostramos el valor leido y eliminamos el salto de linea del final
    print line.rstrip('\n')
    # Detección de "BOTON PULSADO" para arrancar Picamera
    if 'No tiene permitido abrir el cerrojo...' in line:
        print '¡DETECTADA PULSACIÓN!'
        subprocess.check_call('sudo motion', shell=True)
        time.sleep(120)
        subprocess.check_call('sudo pkill -9 motion', shell=True);
```

8.4 Anexo IV. Código programa Arduino

```
/*
 Control Cerradura Inteligente con Latch


 creado Abril 2018
 por Iván Ferrer
 */


//  ********************************** LIBRERIAS ARDUINO **********************************************************

#include <SPI.h>
#include <WiFi101.h>
#include <WiFiUdp.h>
#include <WiFiClient.h>
//#include <WiFiSSLClient.h>
#include  <TimeLib.h>                                                                  // github.com/PaulStoffregen/Time/blob/master/TimeLib.h
//#include <Time.h>                        // playground.arduino.cc/Code/Time  Arduino Time library
#include <sha1.h>                      // github.com/Cathedrow/Cryptosuite  Cryptographic suite for Arduino (SHA, HMAC-SHA)
#include <Base64.h>                      // github.com/adamvr/arduino-base64  Copyright (C) 2013 Adam Rudd
#include <avr/pgmspace.h>                  // AVR libreria libc para almacenamiento datos en memoria flash en lugar de SRAM #######


//  ********************************** CONFIGURACIÓN WIFI **********************************************************

#include "arduino_secrets.h"
///////introduce los datos sensiblesen la pestaña/arduino_secrets.h
char ssid[] = SECRET_SSID;        // your network SSID (name)
char pass[] = SECRET_PASS;        // your network password (use for WPA, or use as key for WEP)
```

```
int keyIndex = 0;            // your network key Index number (needed only for WEP)


// ********************************** DATOS SERVIDOR LATCH Y DIRECCION IP ****************************************

int status = WL_IDLE_STATUS;

// if you don't want to use DNS (and reduce your sketch size)

// use the numeric IP instead of the name for the server:

//IPAddress server(52,30,249,113);          // dirección IP para "https://latch.elevenpaths.com" (sin DNS)

//char server[] = "latch.elevenpaths.com";          // dirección por nombre "https://latch.elevenpaths,com" (con DNS)

//char server1[] = "iot-latch.e-paths.com";   // dirección servidor pruebas

IPAddress server2(192,168,1,99);          // IP Reverse Proxy (Intermediación SSL)


// ********************************** LIBRERIA SSL - PÁGINAS HTTPS ****************************************

// Initialize the Ethernet client library

// with the IP address and port of the server

// that you want to connect to (port 80 is default for HTTP):

//WiFiSSLClient client;              // servidor https

WiFiClient client;                  // servidor http (Reverse Proxy)


// ********************************** SERVIDOR NTP - DATOS ************************************************

unsigned int localPort = 2390;          // puerto local de escucha de paquetes UDP

IPAddress timeServer(130, 206, 3, 166);     // RedIRIS NTP Server

const int NTP_PACKET_SIZE = 48;          // NTP time stamp (sello temporal NTP) esta en los primeros 48 bytes del mensaje

byte packetBuffer[ NTP_PACKET_SIZE];          //buffer para mantener llegada y salida de paquetes

time_t horaAnterior = 0;


// Necesario para enviar paquetes UDP al servidor NTP

// Instancia UDP para enviar y recibir paquetes UDP
```

```
WiFiUDP Udp;

// ********************************************** VARIABLES API LATCH **********************************************

char  Date[] = "X-11Paths-Date:";

char Auth[] = "Authorization:11PATHS";

const char applicationId[] = "M9FDxeB9KHpJxpBC6WEw";

const uint8_t Secret[] = "FEM9wJVRJrQB3F84z4ZVJ3tctE73aHmd6DwuG2BT";


// Variables de apoyo

String X11PathsDate;

String Cabecera1, Cabecera2, URL;

boolean latchStatus;


#define STR_DATE_SIZE           20

#define MAIN_BUFF_SIZE          128

#define LATCH_HMAC_SIZE         20

#define LATCH_HMAC_STR_SIZE     32

#define LATCH_SECRET_SIZE       40


// Long strings to Flash Memory

const char    LatchAppID[]      PROGMEM {"M9FDxeB9KHpJxpBC6WEw"};

const uint8_t LatchSecret[] PROGMEM {"FEM9wJVRJrQB3F84z4ZVJ3tctE73aHmd6DwuG2BT"}; // uint8_t array mandatory, without \0 at end.

const char LatchAccountID[] PROGMEM {"U3BC26u6vT3paRjcBu8zvTMEF7JQyR4QFx296uFj8gt2ButsFRjaQNGC9Zi7v4Ta"};


// Aux strings

const char    LatchAPI[]        PROGMEM {"/api/1.3/status/"};    // servidor https

//const char    LatchAPI[]        PROGMEM {"/api/1.0/status/"};  // servidor pruebas

const char    LatchAuth[]       PROGMEM {"Authorization:11PATHS "};

const char    LatchDate[]       PROGMEM {"X-11Paths-Date"};
```

```
const char     LatchON[]         PROGMEM {"{\"status\":\"on\"}"};

// Global buffers
char MainBuffer[MAIN_BUFF_SIZE];            // Main buffer
char DateTime[STR_DATE_SIZE];               // Date Time Buffer 11Paths-Date format
uint8_t hmac[LATCH_HMAC_SIZE];              // HMAC buffer format uint8_t (20 bytes)
char hmac_str[LATCH_HMAC_STR_SIZE];         // HMAC buffer format string (32 bytes)
uint8_t* hmac_p = hmac;                     // Pointer to HMAC buffer format uint8_t

//          ****************************************          BOTÓN          Y          LEDS
**************************************************

const int rojo = 0;
const int amarillo = 1;
const int pulsador = 2;
int boton = 0;
const int rele = 3;

//          ****************************************          IMPRIME    DATOS    WIFI
****************************************

void printWiFiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
```

```
  Serial.println(" dBm");
}

// ************************************* ENVIO PETICIÓN NTP AL SERVIDOR DE TIEMPO  ****************************
unsigned long sendNTPpacket(IPAddress& address)
{
  // set all bytes in the buffer to 0
  memset(packetBuffer, 0, NTP_PACKET_SIZE);
  // Initialize values needed to form NTP request
  // (see URL above for details on the packets)
  packetBuffer[0] = 0b11100011;   // LI, Version, Mode
  packetBuffer[1] = 0;     // Stratum, or type of clock
  packetBuffer[2] = 6;     // Polling Interval
  packetBuffer[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  packetBuffer[12]  = 49;
  packetBuffer[13]  = 0x4E;
  packetBuffer[14]  = 49;
  packetBuffer[15]  = 52;

  // all NTP fields have been given values, now
  // you can send a packet requesting a timestamp:
  Udp.beginPacket(address, 123); //NTP requests are to port 123
  Udp.write(packetBuffer, NTP_PACKET_SIZE);
  Udp.endPacket();
}

//       ************************************************************       PETICION SERVIDOR NTP ************************************************************
void peticionNTP(){
  Serial.println("\nIniciando conexion al servidor NTP...");
  Udp.begin(localPort);
```

```
  sendNTPpacket(timeServer); // send an NTP packet to a time server
  // wait to see if a reply is available
  delay(1000);
  if ( Udp.parsePacket() ) {
    Serial.println("packet received");
    // We've received a packet, read the data from it
    Udp.read(packetBuffer, NTP_PACKET_SIZE);                // lectura del paquete en el buffer

    //the timestamp starts at byte 40 of the received packet and is four bytes,
    // or two words, long. First, extract the two words:

    unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // combine the four bytes (two words) into a long integer
    // this is NTP time (seconds since Jan 1 1900):
    unsigned long secsSince1900 = highWord << 16 | lowWord;

    // Unix time starts on Jan 1 1970. In seconds, that's 2208988800:
    const unsigned long seventyYears = 2208988800UL;
    // subtract seventy years:
    unsigned long epoch = secsSince1900 - seventyYears;

     setTime(epoch);                // AÑADIMOS SEGUNDOS CALCULADOS AL SISTEMA ARDUINO
                        // YA TENEMOS HORA UTC PARA PETICIONES
  }
} //peticionNTP()

//     ********************************************** MOSTRAR FECHA SEGÚN FORMATO  ************************************
void muestraFecha(){
  String mes, dia, hora, minutos, segundos;
```

```
  mes = compruebaFormato(month());
  dia = compruebaFormato(day());
  hora = compruebaFormato(hour());
  minutos = compruebaFormato(minute());
  segundos = compruebaFormato(second());
   X11PathsDate = String(year()) + "-" + mes + "-" + dia + " " + hora + ":" + minutos + ":" + segundos;
  Serial.println (X11PathsDate);
}

//      ********************************************** COMPROBACIÓN FORMATO DIGITOS HORA **********************************
String compruebaFormato (int i) {
  if (i<10){
    return ("0"+String(i));
  }
  else
    return (String(i));
}

//          ********************************************** GENERACIÓN DE LAS CABECERAS **************************************
void generarCabeceras(){

  // Conversión String 'X11PathsDate' to Char 'DateTime'
  strncpy(DateTime, X11PathsDate.c_str(),20);

  // Inicio HMAC-SHA1 con el Secreto Latch
  memset(MainBuffer, 0, sizeof(char)*MAIN_BUFF_SIZE);          // Clean main buffer
  memcpy_P((uint8_t*)MainBuffer,LatchSecret,LATCH_SECRET_SIZE); // Copy Latch Secret (40 bytes uint8_t) like "I8QsZC1U----------yZeb7rMmzHml"
  Sha1.initHmac((uint8_t*)MainBuffer,LATCH_SECRET_SIZE);       // Init SHA-1 HMAC
```

```
// string a encriptar
memset(MainBuffer, 0, sizeof(char)*MAIN_BUFF_SIZE);          // Vacio Main buffer
sprintf(MainBuffer,"GET\n%s\n\n",DateTime);                  // Añado DateTime          "GET\
n2015-08-27 21:55:49\n\n"
strcat_P(MainBuffer,LatchAPI);                               // Añado PROGMEM LatchAPI      "/api/
1.3/status/"
strcat_P(MainBuffer,LatchAccountID);                         // Añado PROGMEM LatchAccountID
"2294544baf39----------15f7b66467b49d43297"

// Firma HMAC-SHA
Sha1.print(MainBuffer);                                      // Calculo HMAC
memset(hmac,0,(LATCH_HMAC_SIZE*sizeof(uint8_t)));            // Vacio HMAC unit8_8 dst
buffer
hmac_p=Sha1.resultHmac();                                    // Calculo HMAC (20 bytes uint8_t )

// Codificación Base64
 memset(hmac_str,0,(LATCH_HMAC_STR_SIZE*sizeof(char)));      // Vacio dst string
HMAC buffer
 Base64.encode(hmac_str,(char*)hmac_p, LATCH_HMAC_SIZE);     // Cofifica base64
HMAC a string HMAC buffer

Serial.print(F("Base64: Latch Request Signature -->  "));
Serial.println(hmac_str);

// Cabeceras "Authorization" and "11Paths-Date" (se necesita AppID and HMAC string)
memset(MainBuffer, 0, sizeof(char)*MAIN_BUFF_SIZE);          // Vacio Main buffer
 strcpy_P(MainBuffer,LatchAuth);                             // Añado PROGMEM LatchAuth
{"Authorization: 11PATHS "}
strcat_P(MainBuffer,LatchAppID);                             // Añado PROGMEM LatchAppID
{"..."}
strcat  (MainBuffer," ");
 strcat  (MainBuffer,hmac_str);                              // HMAC str 32 bytes like
"bVh7NwAOdlo32ZPQY6ixviNbX50="
strcat  (MainBuffer,"\r\n");
```

```
  Cabecera1 = MainBuffer;                          // *** Copio a cabecera1 ***


  memset(MainBuffer, 0, sizeof(char)*MAIN_BUFF_SIZE);          // Vacio Main buffer
  strcat_P(MainBuffer,LatchDate);                          // Añado PROGMEM LatchDate {"X-
11Paths-Date"}
  strcat  (MainBuffer,": ");
  strcat  (MainBuffer,DateTime);                          // Añado 11Paths-Date str 20 bytes like
"2015-08-27 21:55:49"
  strcat  (MainBuffer,"\r\n");
  Cabecera2 = MainBuffer;                          // *** Copio a cabecera2 ***


  // Genero el String URL
  memset(MainBuffer, 0, sizeof(char)*MAIN_BUFF_SIZE);          // Vacio Main buffer
  strcat_P(MainBuffer,LatchAPI);                          // Añado PROGMEM LatchAPI       "/api/
1.3/status/"
  strcat_P(MainBuffer,LatchAccountID);                          // Añado PROGMEM LatchAccountID
"2294544baf39----------15f7b66467b49d43297"
  URL = MainBuffer;


}//generarCabeceras()


// ********************************************************** CONEXIÓN SERVIDOR
LATCH **********************************************************
void conectar_servidor(){
 // Test comprobación de que se envía
 Serial.println ("Impresión de cabeceras --> ");
 Serial.println ("Cabecera1 --> "+Cabecera1);
 Serial.println ("Cabecera2 --> "+Cabecera2);
 Serial.println ("URL --> "+URL);



  Serial.println("\n Estableciendo conexión al servidor...");
  // if you get a connection, report back via serial:
  //if (client.connectSSL(server, 443)) {                          // Servidor LATCH(https)
```

```
  //if (client.connect(server1, 80)) {                    // Servidor test (NO https)
  if (client.connect(server2, 8080)) {                    // Conexión a traves SSL Intermediación
    Serial.println("conectado al servidor");
    // Make a HTTP request:
    client.print("GET ");
    client.print(URL);
    client.println(" HTTP/1.1");
    client.print("Host: ");
    client.println("latch.elevenpaths.com");
    client.println("User-Agent: Arduino/1.0");
    client.print(Cabecera1);
    client.print(Cabecera2);
    client.println("Connection: close");
    //client.println("Connection: keep-alive");
    client.println();

  }
  else
    Serial.println("No conectado al servidor");

  unsigned long timeout = millis();
  while (client.available() == 0) {
    if (millis() - timeout > 5000) {
      Serial.println(">>> Client Timeout !");
      client.stop();
    }
  }

}// conectar_servidor()
```

```
//      ************************************************************      RECEPCIÓN RESPUESTA ************************************************************
String respuesta_servidor_analizada(){

 String latch;

 // Si hay llegada de bytes disponibles desde el servidor, los lee e imprime por pantalla
 while (client.available()) {

  // Opción 2     CAPTURA POR STRING
  String line = client.readStringUntil('\n');
  Serial.println(line);
  char charBuf[100];
  line.toCharArray(charBuf,100);

  if (strstr(charBuf,"status")){
   if (strstr(charBuf,"off")){
    latch = "off";
   }
   else
    latch = "on";
  }
 }

 // if the server's disconnected, stop the client:
 if (!client.connected()) {
  Serial.println();
  Serial.println("disconnecting from server.");
  client.stop();
 }

 return (latch);
```

```
}// respuesta_servidor_analizada()



// ********************************************** MOVIMIENTO CERROJO Y LEDs **********************************************
void mov_cerrojo(String accion){
  int pos = 0;

  if (accion == "on"){
    Serial.println("Abriendo cerrojo...");
    digitalWrite (amarillo, HIGH);
    digitalWrite (rele, LOW);
    delay(8000);              // Actuación sobre el relé durante 8seg
    Serial.println("Cerrando cerrojo...");
    digitalWrite (amarillo, LOW);
    digitalWrite (rele, HIGH);
  }
  else {
    Serial.println("No tiene permitido abrir el cerrojo...");
    digitalWrite (rojo, HIGH);
    delay(3000);
    digitalWrite (rojo, HIGH);
    digitalWrite (rojo, LOW);
  }
}// mov_cerrojo()

//          ************************************************************          SETUP **********************************************
void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
```

```
  }

  // Configuración de los pines IN/OUT
  pinMode(amarillo, OUTPUT);              // LED amarillo
  pinMode(rojo, OUTPUT);                  // LED rojo
  pinMode(pulsador, INPUT);               // Entrada del pulsador
  pinMode(rele, OUTPUT);                  // Salida hacia Relé
  digitalWrite (rele, HIGH);              // Deja cerradura cerrada

  // check for the presence of the shield:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue:
    while (true);
  }

  // attempt to connect to WiFi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }
  Serial.println("Connected to wifi");
  printWiFiStatus();

}

void loop() {
```

```
  boton = digitalRead(pulsador);              // Lectura del pulsador para efectuar petición de estado
a Latch


  if (boton == HIGH){
    Serial.println("BOTON PULSADO");
    Serial.println("Enviando consulta...");

    // Peticion Servidor NTP
    peticionNTP();

    muestraFecha();
    delay(500);

    Serial.println("\nGenerando cabeceras petición");
    generarCabeceras();

    // Conexión al servidor
    Serial.println("\nConectando al servidor...");
    conectar_servidor();

    // Recepción Respuesta
    Serial.println("\nRecibiendo respuesta del servidor...");
    String estado = respuesta_servidor_analizada();                  // Análisos del JSON recibido
    Serial.println ("Estado de latch: " + estado);
    mov_cerrojo(estado);                                    // Análisis de actuación a realizar
  }

  Serial.println("...");
  delay(300);
}
```