UNIVERSIDAD DE COSTA RICA

Facultad de Ingeniería

Escuela de Ingeniería Eléctrica

IE0527 Ingeniería de Comunicaciones

Proyecto Final

Comunicación BLE-WiFi en ESP32 para recuperación de credenciales

Estudiantes:
Paulette Pérez Monge B95916
Junior Ruiz Sánchez B97026
Justin Jiménez Jiménez B94037
Mario Rocha Morales B96561

Profesor: Mario Vega Arguello

3 de julio de 2025

Índice

1. Análisis del problema y desafíos

El escenario planteado requiere la implementación de un sistema de autenticación y distribución segura de credenciales para permitir que un microcontrolador ESP32 se conecte automáticamente a una red WiFi, como se observa en la figura 1.

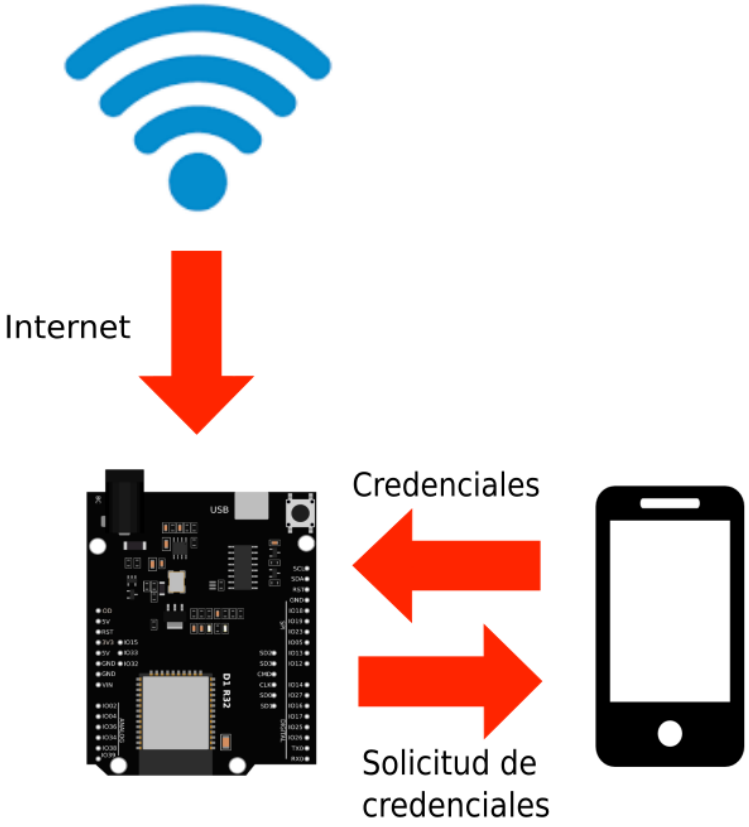

Figura 1: Diagrama de la conexión del ESP32 con el dispositivo y el WiFi [Elaboración propia]

Por el momento no se pretende enviar datos por Internet, sólo confirmar la conexión. Se identifican los siguientes pasos:

1. Inicializar el módulo Bluetooth en modo BLE.
2. Escanear dispositivos BLE cercanos.
3. Establecer conexión con el dispositivo seguro que contiene las credenciales.
4. Autenticarse con el dispositivo seguro.
5. Una vez autenticado, recibir el SSID y password por BLE.
6. Guardar las credenciales en la memoria.
7. Conectarse al wifi.
8. Confirmar la conexión

Este proceso involucra múltiples desafíos técnicos que deben ser abordados para garantizar seguridad, confiabilidad y eficiencia en la comunicación entre dispositivos. A continuación, se presentan algunos:

1.1. Uso del ESP 32

El equipo de trabajo cuenta con un ESP32 del tipo D1 R32. Para utilizarlo, será necesario descargar la IDE de Arduino y programar en ésta y asegurarse de que los drivers del ESP sean instalados.

Asimismo, es preciso tener claro los recursos con los que cuenta éste dispositivo, tal como:

- Frecuencia de reloj: rango de 240 MHz
- RAM: 512 kB
- Memoria flash externa: 4 MB

[1], de forma que el código pueda funcionar con éstas limitaciones.

1.2. Establecimiento del servidor de la Conexión BLE

Como se indicó en la nota teórica, el uso de Bluetooth Low Energy (BLE) requiere de un cliente y un servidor [2]. Esto indica que es necesario conseguir algún dispositivo que funja de servidor de Bluetooth para que contenga las credenciales y pueda transmitirlas.

Aplicaciones como *nRF Connect for Mobile* permiten que un teléfono celular interactúe con dispositivos BLE [3].

1.3. Administración de Energía y Consumo

El ESP32 tiene la ventaja de presentar varias opciones de ahorro de energía como las siguientes:

- Configurar el ESP32 para usar modos de suspensión como *Light Sleep* o *Deep Sleep* cuando no se esté comunicando [4].
- Usar interrupciones BLE para despertar el dispositivo solo cuando sea necesario [5].

Esto podría ser implementado como optimización adicional.

2. Funcionamiento de WiFi y Bluetooth Low Energy

2.1. Wi-Fi (IEEE 802.11)

2.1.1. Funcionamiento general

IEEE 802.11 es la familia de estándares para redes inalámbricas de área local (WLAN) más utilizada en todo el mundo. Define cómo los dispositivos acceden a un canal compartido mediante CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance), empleando funciones como DCF (Distributed Coordination Function) o EDCA (Enhanced Distributed Channel Access) para priorizar tráfico en entornos de alta demanda [6, 7].

Versiones recientes como Wi-Fi 6 (802.11ax) y Wi-Fi 7 (802.11be) integran tecnologías como OFDMA, MU-MIMO y Multi-Link Operation (MLO), lo que permite reducir latencia, mejorar uso espectral y mantener compatibilidad con dispositivos heredados [8, 9].

2.1.2. Topologías

IEEE 802.11 admite tres configuraciones principales:

Infraestructura (BSS/ESS): Un punto de acceso (AP) centraliza la coordinación entre estaciones clientes (STA). Varios BSS con el mismo SSID pueden formar un ESS (Extended Service Set), permitiendo roaming sin perder conexión.

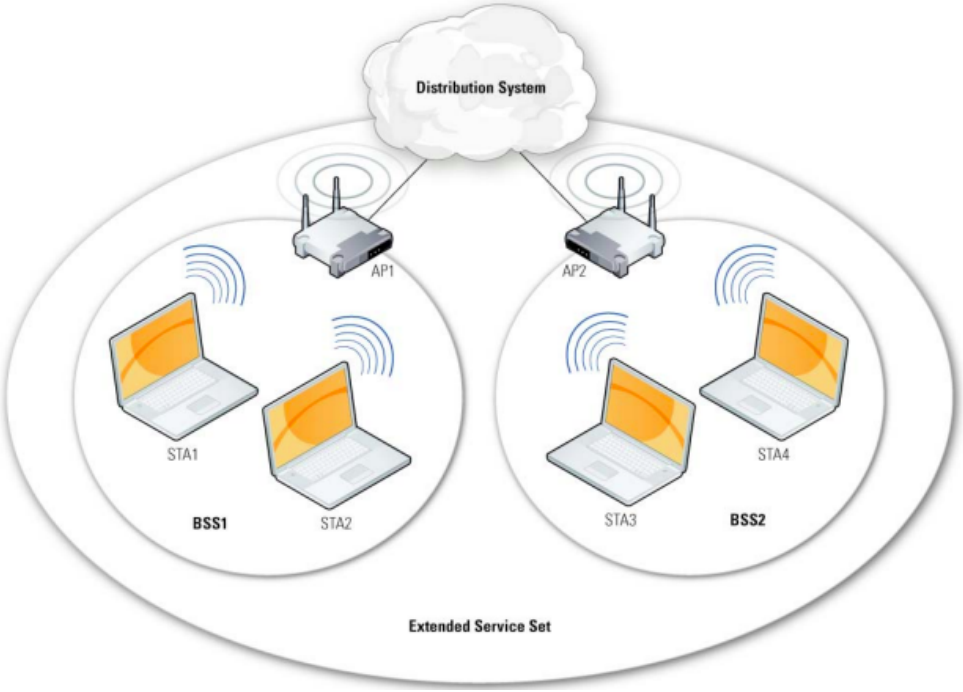

Figura 2: Modo de infraestructura IEEE 802.11 [10]

Ad hoc (IBSS): Las estaciones se comunican directamente sin un AP. Es ideal para redes temporales de corto alcance.

Figura 3: Arquitectura del modo Ad Hoc de IEEE 802.11 [10]

Mallada (802.11s): Cada nodo funciona como repetidor, reenvía tramas y enruta paquetes mediante múltiples saltos, formando una red auto-reparable y flexible.

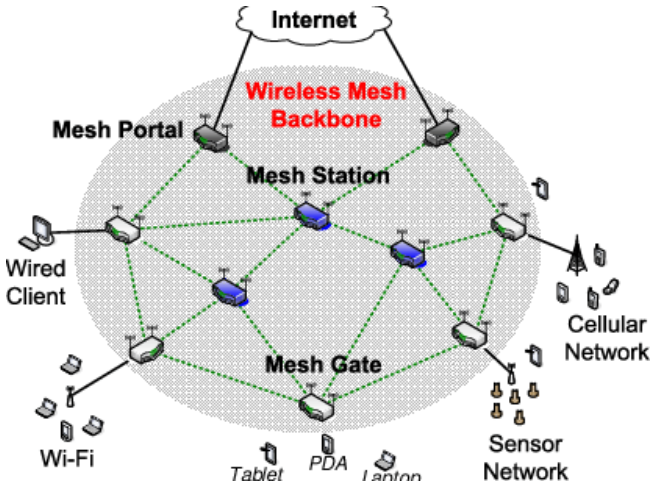

Figura 4: Arquitectura de red en malla IEEE 802.11s [11]

2.1.3. Parámetros de desempeño

Generación	Banda	BW máx.	Modulación	Velocidad pico	Latencia
802.11n	2.4/5 GHz	40 MHz	64-QAM	600 Mb/s	5–30 ms [6]
802.11ac	5 GHz	160 MHz	256-QAM	6.9 Gb/s	2–10 ms [6]
802.11ax	2.4/5/6 GHz	160 MHz	1024-QAM	9.6 Gb/s	$<$1 ms [6]
802.11be	2.4/5/6 GHz	320 MHz	4096-QAM	30–46 Gb/s	sub-ms (MLO) [9]

Tabla 1: Evolución de Wi-Fi según estándar.

Limitaciones y mejoras La congestión en 2.4/5 GHz y el alto consumo energético siguen siendo problemas; sin embargo, mecanismos como Target Wake Time (TWT) y programación OFDMA mejoran la eficiencia espectral y la vida de batería. Wi-Fi 7 introduce *Multi-Link Operation* y coordinación multi-AP para reforzar capacidad y robustez [9].

2.2. Bluetooth Low Energy (BLE)

2.2.1. Topologías

BLE opera en modo *central–peripheral:* el periférico (bajo consumo) anuncia su presencia; el central inicia y gestiona la conexión y el flujo de datos, difusión (*broadcast*): el broadcaster envía paquetes de advertising sin aceptar conexiones; los observers solo escuchan y *Bluetooth Mesh*, esta última escalable hasta 32 767 nodos mediante *managed flooding* [12].

Figura 5: Topología de una red BLE [13]

2.2.2. Parámetros de desempeño

PHY	Tasa (Mb/s)	Sensibilidad	Alcance	Fuente
LE 1M	1	∼–95 dBm	50 m	[14]
LE 2M	2	∼–92 dBm	40 m	[14]
LE Coded	0.125/0.5	∼–105 dBm	>100 m	[14]

Tabla 2: Desempeño de los PHY de BLE 5.x.

Limitaciones y avances El tasa efectiva rara vez supera 700 kb/s y la seguridad depende del método de emparejamiento; estudios recientes hallan implementaciones sin *LE Secure Connections* [15]. Las especificaciones 5.3 añaden canales isócronos para *LE Audio* y mejoras de control de potencia [16].

2.3. Sinergias Wi-Fi / BLE

Combinar BLE para aprovisionamiento de credenciales y Wi-Fi como enlace de datos maximiza autonomía y ancho de banda, práctica recomendada en sistemas IoT heterogéneos [16, 12].

	Bluetooth Low Energy (LE)	Bluetooth Basic Rate/ Enhanced Data Rate (BR/EDR)
Optimized For...	Short burst data transmission	Continuous data streaming
Frequency Band	2.4GHz ISM Band (2.402 – 2.480 GHz Utilized)	2.4GHz ISM Band (2.402 – 2.480 GHz Utilized)
Channels	40 channels with 2 MHz spacing (3 advertising channels/37 data channels)	79 channels with 1 MHz spacing
Channel Usage	Frequency-Hopping Spread Spectrum (FHSS)	Frequency-Hopping Spread Spectrum (FHSS)
Modulation	GFSK	GFSK, π/4 DQPSK, 8DPSK
Power Consumption	~0.01x to 0.5x of reference (depending on use case)	1 (reference value)
Data Rate	LE 2M PHY: 2 Mb/s LE 1M PHY: 1 Mb/s LE Coded PHY (S=2): 500 Kb/s LE Coded PHY (S=8): 125 Kb/s	EDR PHY (8DPSK): 3 Mb/s EDR PHY (π/4 DQPSK): 2 Mb/s BR PHY (GFSK): 1 Mb/s
Max Tx Power*	Class 1: 100 mW (+20 dBm) Class 1.5: 10 mW (+10 dbm) Class 2: 2.5 mW (+4 dBm) Class 3: 1 mW (0 dBm)	Class 1: 100 mW (+20 dBm) Class 2: 2.5 mW (+4 dBm) Class 3: 1 mW (0 dBm)
Network Topologies	Point-to-Point (including piconet) Broadcast Mesh	Point-to-Point (including piconet)

Figura 6: Tabla comparativa entre BLE y Bluetoooth Classic. [2]

3. Análisis del Proyecto

3.1. Solución propuesta.

Un comunicación BLE se conforma por dos dispositivos: Un cliente y un servidor. El servidor anuncia su existencia para que otros dispositivos puedan encontrarlo y contiene los datos que el cliente puede leer. El cliente escanea los dispositivos cercanos y, al encontrar el servidor que busca, establece una conexión y recive los datos entrantes. Esto se denomina **comunicación punto a punto** [2]. Por esta razón el **hardware** a utilizar será el siguiente:

- Un **microcontrolador TTGO ESP32 Board UNO** encargado de conectarse a la red WiFi que actuará como **cliente**.
- Un **dispositivo Android**, que actúa como el *dispositivo seguro* para el almacenamiento de credenciales, implementando un **servidor** Bluetooth Low Energy (BLE).

Este enfoque permite utilizar aplicaciones móviles disponibles.

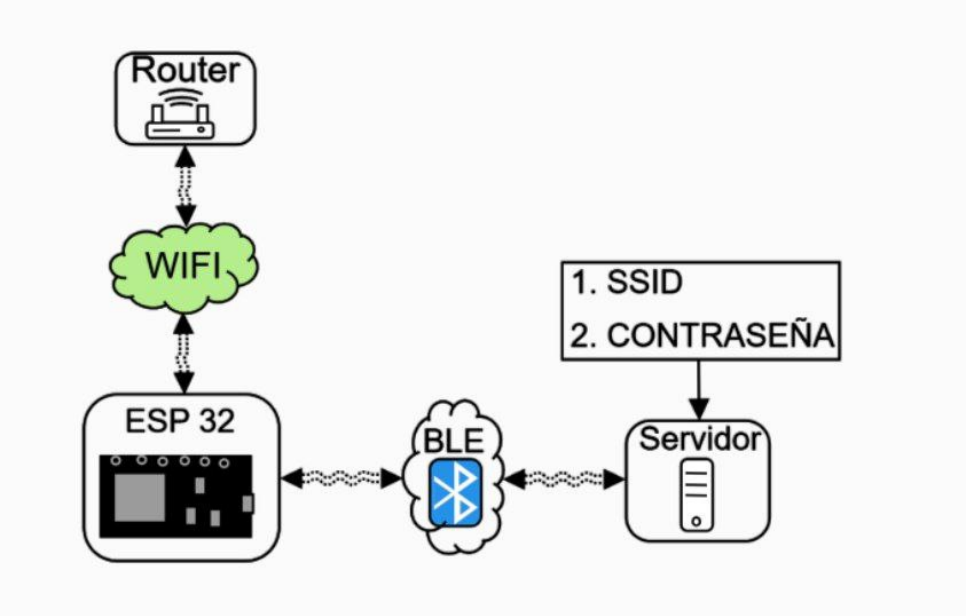

Figura 7: Diagrama de la conexión [Elaboración Propia]

Etapa 1 — Comunicación BLE con el dispositivo Android

Al inicio del sistema, el ESP32 realizará un escaneo buscando el periférico BLE identificado como:

- **Nombre del dispositivo BLE:** `ServidorWiFi-BLE`

El servidor BLE será implementado en el dispositivo Android mediante la aplicación **nRF Connect for Mobile**, la cual permite crear servicios GATT personalizados [3]. La estuctura de los datos GATT usada para la comunicación BLE está organizada de forma jerárquica, como indica la figura 8.

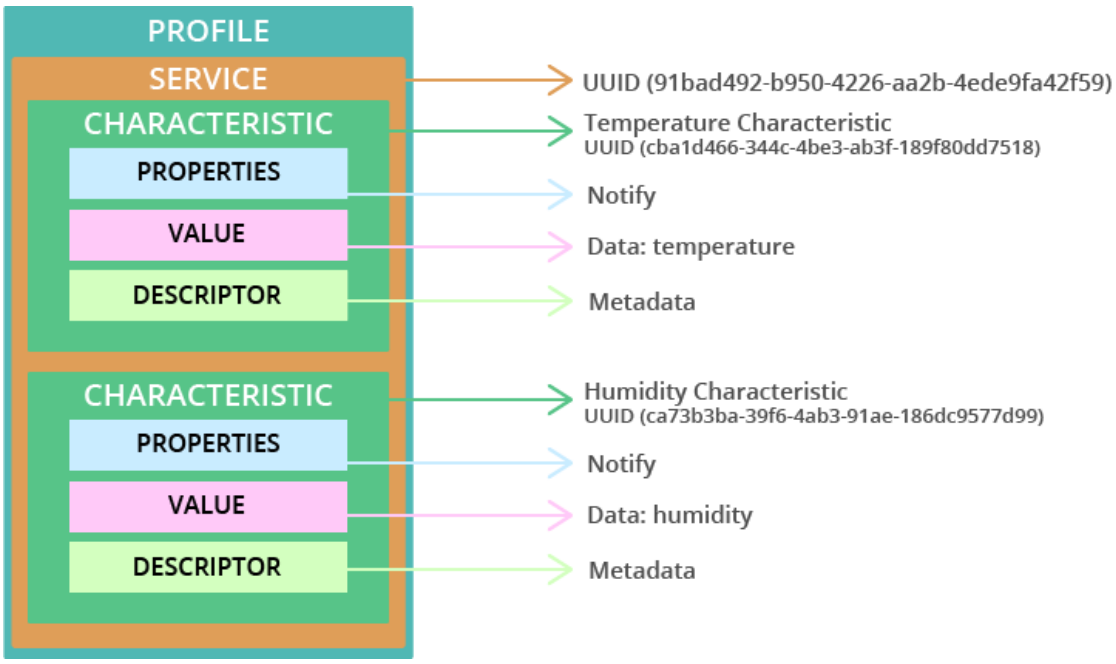

Figura 8: Estructura de datos GATT [17]

Los perfiles (conjuntos de servicios), los servicios (conjuntos de características) y las características, que es donde se almacena la información y efectúan operaciones de Broadcast, Read, Write, entre otras. Tanto los servicios como las características se distingues por UUIDs o identificadores únicos universales de 16 bytes [2].

Con el contexto anterior, el servicio y las características configuradas serán las siguientes:

Tabla 3: UUIDs y operaciones del servicio BLE.

Propósito	**UUID**	**Operación**
Servicio principal	12345678-1234-1234-1234-1234567890ab	—
Autenticación	abcd1234-5678-90ab-cdef-1234567890ab	WRITE
Leer SSID	abcd1234-5678-90ab-cdef-1234567890ac	READ
Leer contraseña	abcd1234-5678-90ab-cdef-1234567890ad	READ

En este escenario, la lógica de autenticación no será controlada desde el servidor BLE, sino que será validada por el propio cliente ESP32. Las características `READ` podrán ser accedidas directamente una vez establecida la conexión.

La configuración del servidor BLE se realizará de la siguiente manera [18]:

1. Descargar e instalar la aplicación `nRF Connect for Mobile`.
2. Ingresar al apartado `GATT Server`.
3. Crear un nuevo servicio con UUID `12345678-1234-1234-1234-1234567890ab`.
4. Agregar las características:
   - UUID: `abcd1234-5678-90ab-cdef-1234567890ab` (WRITE)
   - UUID: `abcd1234-5678-90ab-cdef-1234567890ac` (READ)
   - UUID: `abcd1234-5678-90ab-cdef-1234567890ad` (READ)
5. Asignar los valores de SSID (el nombre de la red [19]) y contraseña correspondientes.

Etapa 2 — Conexión WiFi desde el ESP32

Mecanismo de autenticación

El proceso de autenticación va a funcionar de la siguiente forma:

1. Una vez conectado al servidor BLE, el ESP32 escribe la palabra `''hacking''` en la característica de autenticación con UUID WRITE.
2. Después de escribir la palabra clave, el ESP32 procede a leer las características de SSID y contraseña.
3. Si logra obtener las credenciales correctamente, considera la autenticación exitosa.
4. Si la lectura falla o los valores obtenidos no son válidos, el ESP32 puede notificar un fallo de autenticación.

Una vez obtenidas las credenciales de la red, el ESP32 utilizará su interfaz WiFi para intentar conectarse directamente a la red especificada. Si la conexión es exitosa, el dispositivo obtendrá una dirección IP [20], la cual podrá ser visualizada mediante un monitor serial.

3.2. Solución implementada.

Como se mencionó en la propuesta, el microcontrolador usado fue el TTGO ESP32 Board UNO, sin embargo, para realizar la implementación del servidor, la aplicación que estaba contemplada inicialmente (nRF Connect for Mobile) no contaba con las funcionalidades necesarias para cumplir con las especificaciones del proyecto, ya que no era posible autenticar de manera correcta el dispositivo, permitiendo acceso total a las credenciales sin necesidad de acertar la característica de autenticación, ya que como lo muestra la tabla 3 la autenticación debe hacerse en modo de operación de escritura, y al usar esta aplicación el comportamiento no era el deseado, para poder desarrollar el proyecto usando dicha aplicación y manteniendo la segurar del sistema habría que usar el modo de autenticación en read por lo que no cumpliría con las indicaciones del enunciado.

Considerando lo anterior, a manera de reemplazo se implementó un servidor en un sistema Linux, mediante un script de Python usando los paquetes mostrados a continuación:

- **bluez**: Pila oficial de Bluetooth para Linux, incluye el daemon `bluetoothd` con soporte GATT. [21]
- **python3-dbus**: Soporte de D-Bus en Python para interactuar con `bluetoothd`.[22]
- **python3-gi**: Binding para GObject Introspection, permite usar `GLib.MainLoop` y otras funciones necesarias. [22]
- **pydbus** (instalado vía `pip`): Biblioteca Python de alto nivel para publicar servicios GATT mediante D-Bus. [22]
- **network-manager** (opcional): Herramienta para gestionar conexiones de red WiFi.
- **nmcli** (opcional): Interfaz de línea de comandos para controlar `NetworkManager`.

Finalmente se obtuvo el desempeño esperado, autenticando de manera correcta y transmitiendo la información sin problemas.

La solución está disponible e:

Repositorio GitHub

El proyecto completo se encuentra disponible en: https://github.com/jrpompio/ProyectoTeleco/

- **esp32/esp32.ino**: Código para el ESP32. Escanea dispositivos BLE, se conecta, envía el token de autenticación y recibe SSID y contraseña para conectar a Wi-Fi.
- **server_ble.py**: Servidor BLE escrito en Python usando BlueZ y D-Bus. Publica el servicio GATT, valida el token recibido y permite leer SSID y PASS sólo si la autenticación es correcta.
- **interfaz.py**: Interfaz gráfica con PyQt5. Lee mensajes del ESP32 (`ESTADO:`, `RESULTADO:`, `IP:`) y muestra cada estado como bloque tipo diagrama de flujo, resaltando el activo.
- **run_server.bash**: Script Bash para reiniciar el servicio Bluetooth y ejecutar el servidor BLE con permisos de superusuario.

3.3. Diagrama de flujo de la solución.

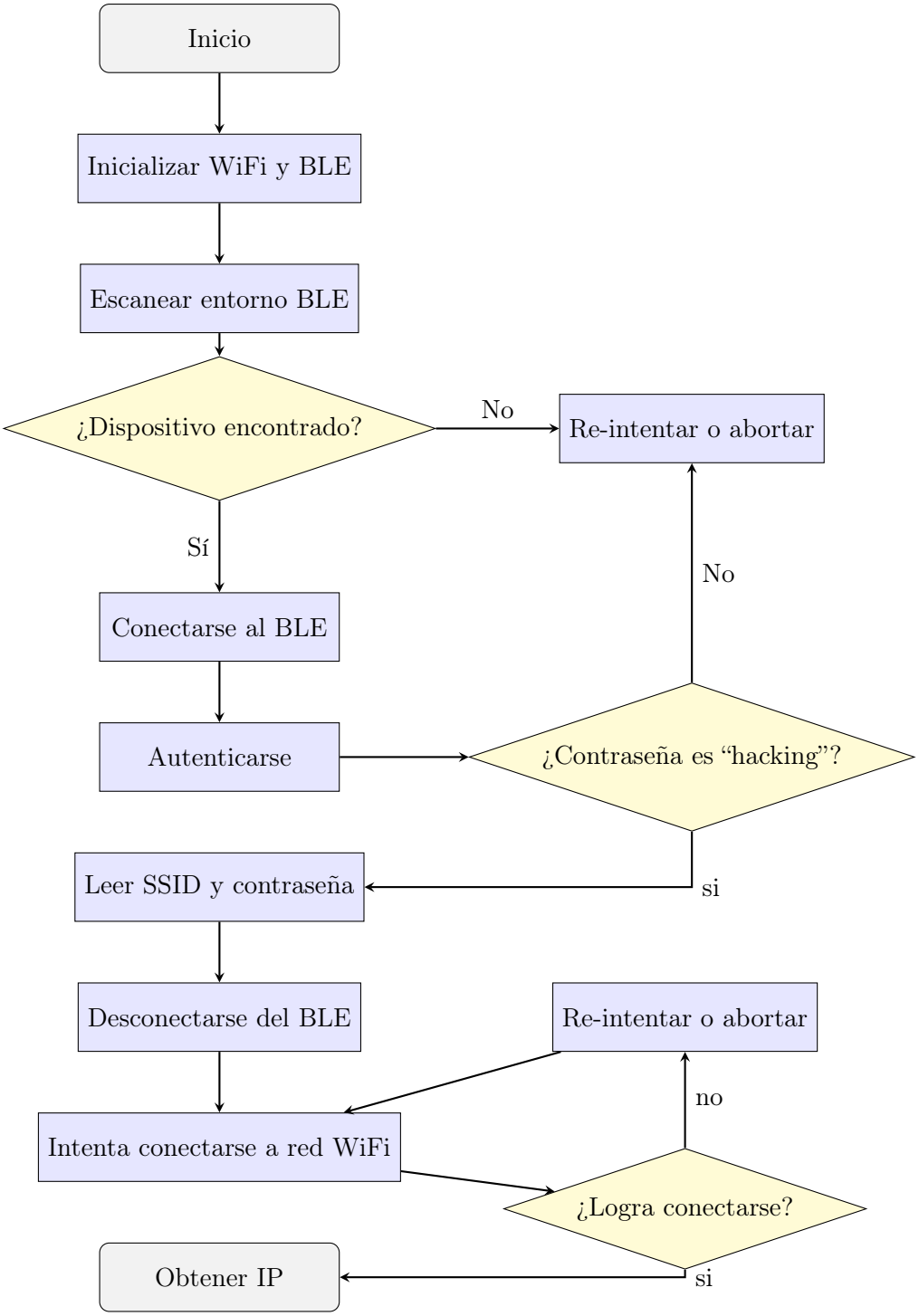

Figura 9: Diagrama de flujo del proceso [Elaboración propia]

Inicio: Punto de partida de la rutina. Se energiza el ESP32.

Iniciar el WiFi y BLE: Se configuran ambos módulos del microcontrolador.

Escanea entorno BLE: Se busca el dispositivo `ServidorWiFi-BLE`.

¿Dispositivo encontrado?: Se decide si continuar o reintentar.

Re-intentar o abordar: Se repite el escaneo o se finaliza con error. Acá debe existir un contador para aumentar el tiempo de retry con cada intento incrementado.

Conectarse al BLE: Establece enlace BLE con el servidor.

Autenticarse: Envía la clave al UUID de autentificación.

¿Contraseña es "hacking"?: Se comprueba si la clave es "hacking", para continuar, re-intentar o abortar.

Leer SSID y contraseña Se recuperan los datos desde BLE.

Desconectarse del BLE: Libera el canal de comunicación BLE.

Intenta conectarse a red WiFi Usa las credenciales obtenidas para conectarse al WiFi.

¿Logra conectarse? Se comprueba si efectivamente se pudo conectar a la red wifi.

Re-intentar o abordar: Se vuelve a intentar conectarse al wifi o se finaliza con error. Esto sucedería, por ejemplo, si la red se apaga.

Obtener IP: El sistema queda conectado, listo para operar fuera de este bucle.

3.4. Especificaciones del Hardware (TTGO ESP32 Board UNO).

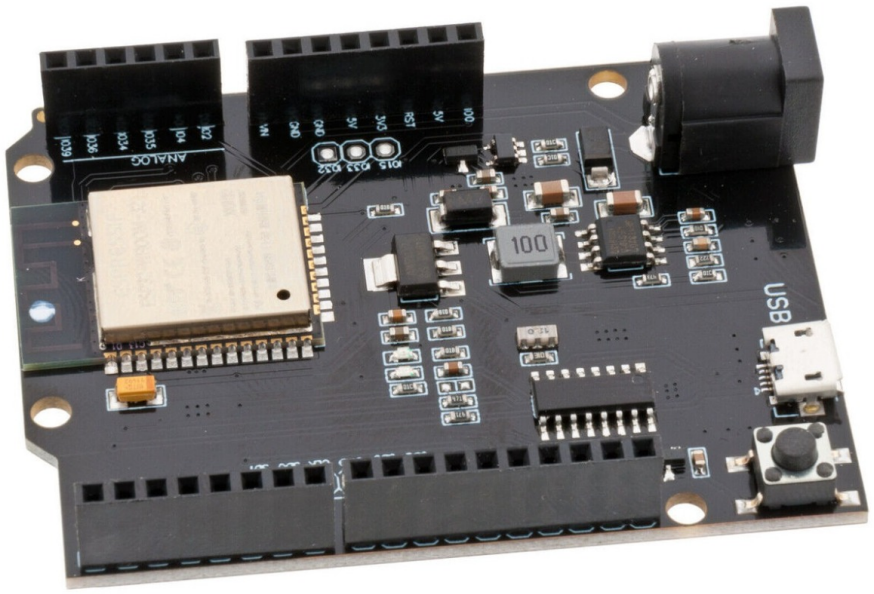

Figura 10: TTGO ESP32 Board UNO (WiFi + Bluetooth)

- **Tensión de alimentación (microUSB):** 5 V DC [23]

- **DC input:** 7–12 V [23]
- **Nivel lógico de E/S:** 3.3 V [23]
- **Corriente de operación mínima:** 250 mA [23][23]
- **SoC:** ESP32-WROOM-32 [23]
- **Frecuencia de reloj:** hasta 240 MHz [23]
- **RAM interna:** 512 kB [23]
- **Flash externa:** 4 MB [23]
- **Pines digitales disponibles:** 20 [23]
- **Entradas analógicas (ADC):** 6 [23]
- **Interfaces:** SPI, I^2C, I^2S, IR, UART, PWM [23]
- **Wi-Fi:** IEEE 802.11 b/g/n/i (hasta 150 Mbps a 2.4 GHz) [23]
- **Antena:** impresa en PCB [23]
- **Chip USB-Serie:** CH340 [23]
- **Dimensiones:** 70 × 55 × 13 mm (2.7 × 2.1 × 0.5 in) [23]
- **Bluetooth:** Bluetooth v4.2 BR/EDR y LE (compatible con BLE) [23]

Referencias

[1] A. Delivery, "D1 r32," disponible en https://www.halloweenfreak.de/arduino/pdfs/D1_R32_ENG.pdf.

[2] R. Santos, "Getting started with esp32 bluetooth low energy (ble) on arduino ide," 2024, disponible en https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/.

[3] N. Semiconductor, "nrf connect for mobile," 2025, disponible en https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-for-mobile.

[4] E.-I. P. Guide, "Introduction to low power mode for systemic power management," 2023, disponible en https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/low-power-mode/low-power-mode-soc.html.

[5] R. Santos, "Esp32 with pir motion sensor using interrupts and timers," disponible en https://randomnerdtutorials.com/esp32-pir-motion-sensor-interrupts-timers/.

[6] M. Z. Ali, J. Misić, and V. B. Misić, "Performance analysis of IEEE 802.11ax heterogeneous network in the presence of hidden terminals," *arXiv preprint arXiv:1908.01834*, 2019.

[7] I. S. Association, "Ieee standard for information technology—part 11: Wireless lan mac and phy specifications," 2020, iEEE Std 802.11-2020.

[8] V. Bhargava and N. Raghava, "An enhancement for IEEE 802.11 sta power saving and access point memory management mechanism," *Electronics*, vol. 11, no. 23, p. 3914, 2022.

[9] X. Liu, Y. Dong, Y. Li, Y. Lin, X. Yang, and M. Gan, "IEEE 802.11be wi-fi 7: Feature summary and performance evaluation," *arXiv preprint arXiv:2309.15951*, 2024, disponible en https://arxiv.org/abs/2309.15951.

[10] K. Scarfone, D. Dicoi, M. Sexton, and C. Tibbs, "Guide to securing legacy IEEE 802.11 wireless networks," National Institute of Standards and Technology (NIST), Tech. Rep. Special Publication 800-48 Revision 1, 2008, disponible en https://www.researchgate.net/publication/329973442_NIST_Special_Publication_800-48_Revision_1_Guide_to_Securing_Legacy_IEEE_80211_Wireless_Networks.

[11] R. J. Neto, R. M. C. Andrade, R. Braga, and F. Theoleyre, "Performance issues with routing in multi-channel multi-interface IEEE 802.11s networks," in *Proceedings of the IFIP Wireless Days (WD)*, Rio de Janeiro, Brazil, 2014, disponible en https://www.researchgate.net/publication/268687031_Performance_Issues_with_Routing_in_Multi-Channel_Multi-Interface_IEEE_80211s_Networks.

[12] A. Aijaz, A. Stanoev, D. London, and V. Marot, "Demystifying the performance of bluetooth mesh: Experimental evaluation and optimization," in *IEEE Wireless Days*, 2021, arXiv:2106.04230.

[13] D. Tudose, "Lab 7. bluetooth low energy (ble)," 2023, disponible en https://ocw.cs.pub.ro/courses/iothings/laboratoare/2022/lab7.

[14] H. Zhao, "Simulate bumblebee and extend it to support LE coded phy in BLE version 5," *arXiv preprint arXiv:2305.10138*, 2023.

[15] Y. Kurt Peker, G. Bello, and A. J. Perez, "On the security of bluetooth low energy in two consumer wearable heart rate monitors/sensing devices," *Sensors*, vol. 22, no. 3, p. 988, 2022.

[16] G. Koulouras, S. Katsoulis, and F. Zantalis, "Evolution of bluetooth technology: BLE in the iot ecosystem," *Sensors*, vol. 25, no. 4, p. 996, 2025.

[17] S. Santos, "Esp32 ble server and client (bluetooth low energy)," 2021, disponible en https://randomnerdtutorials.com/esp32-ble-server-client/.

[18] N. Semiconductors, "Configuring server setup," 2025, disponible en https://docs.nordicsemi.com/bundle/nrf-connect-ble/page/maintaining_server_setup.html.

[19] N. Barney, A. S. Gillis, and L. Phifer, "What is an ssid (service set identifier)?" Online; TechTarget – Search Mobile Computing, Oct. 2024, disponible en https://www.techtarget.com/searchmobilecomputing/definition/service-set-identifier.

[20] S. Santos, "Esp32 useful wi-fi library functions (arduino ide)," Online; Random Nerd Tutorials, 2021, disponible en https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/.

[21] "BlueZ: Bluetooth protocol stack for Linux (Git repository)," https://git.kernel.org/pub/scm/bluetooth/bluez.git, BlueZ Project, 2025, cloned as of commit, see repository tags (e.g., v5.83). [Online]. Available: https://git.kernel.org/pub/scm/bluetooth/bluez.git

[22] "BlueZ Git Examples - example-gatt-server," https://git.kernel.org/pub/scm/bluetooth/bluez.git, 2025, scripts de ejemplo para GATT server en Python. [Online]. Available: https://git.kernel.org/pub/scm/bluetooth/bluez.git

[23] "D1 r32 development board datasheet (english)," AZ-Delivery Vertriebs GmbH, 2019, accessed: 12 Jun 2025. [Online]. Available: https://www.halloweenfreak.de/arduino/pdfs/D1_R32_ENG.pdf