Práctica 3. WiFi. Provisionamiento y ahorro de energía.

Objetivos

La presente práctica se divide en dos partes en las que se estudiarán dos aspectos avanzados de wifi. Los objetivos de cada parte son:

- Provisionamiento de credenciales wifi
    - Entender y experimentar con distintos modos de provisionamiento de credenciales WiFi, vía `BLE` y vía `softAP`.
    - Comprobar el intercambio de claves en claro realizando provisionamientos desde línea de comandos, así como observar la utilidad (y necesidad) del envío cifrado de credenciales.
- Modos de ahorro de energía
    - Entender los tres modos de funcionamiento del ESP32 a nivel de ahorro energético en conexiones WiFi.
    - Observar la desviación en latencia de recepción de paquetes en función del modo aplicado.

Entregable

Para esta práctica los alumnos harán un breve informe documentando las tareas realizadas y los resultados obtenidos.

Parte 1.Provisionamiento de credenciales WiFi

Entendemos por provisionamiento WiFi el mecanismo o mecanismos mediante los cuales es posible proporcionar, de forma externa y segura, el conjunto de credenciales de conexión a una red WiFi a un dispositivo no conectado. Es decir, si nuestro código en el ESP32 está preparado para conectarse a una wifi por WPA2 personal, en lugar de configurar una red fija o una serie de redes válidas, añadimos a nuestro código un componente de provisionamiento, que permitirá al usuario conectarse de algún modo al ESP32 para indicarle el nombre de la Wifi a la que conectarse (la SSID) y la contraseña que debe utilizar. El provisionamiento por tanto suele hacerse la primera vez que se usa el dispositivo y no suele ser necesario repetirlo.

ESP-IDF proporciona un componente que ofrece un servicio de provisionamiento WiFi por dos mecanismos: Bluetooth Low Energy (modo *BLE*) o a través de un punto de acceso WiFi temporal (modo *SoftAP*). ESP-IDF proporciona una serie de APIs (con prototipos `wifi_prov_mgr_*`) para implementar de forma sencilla ambos modos de provisionamiento.

Para completar esta parte de la práctica deberás trabajar con el ejemplo `examples/provisioning/wifi-prov-mgr`.

Inicialización del servicio de provisionamiento

La rutina `wifi_prov_mgr_init()` se utiliza para configurar e inicializar el componente de provisionamiento, y debe invocarse antes de cualquier otra invocación a rutinas de tipo `wifi_prov_mgr*`. Además, es necesario destacar que el componente de provisionamiento confía en las funcionalidades de otros componentes (básicamente NVS, TCP/IP, *Event loop* y mDNS), por lo que éstos deben inicializarse antes del propio componente. Para finalizar el componente de provisionamiento, es suficiente con invocar a la rutina `wifi_prov_mgr_deinit()`.

Un ejemplo de inicialización resultaría en:

```
wifi_prov_mgr_config_t config = {
  .scheme = wifi_prov_scheme_ble,
  .scheme_event_handler = WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM
};

ESP_ERR_CHECK( wifi_prov_mgr_init(config) );
```

La estructura de configuración de tipo `wifi_prov_mgr_config_t` dispone de campos que permiten especificar el comportamiento del componente. El campo `scheme` especifica el esquema (o tipo) de provisionamiento que vamos a usar, mientras que el campo `scheme_event_handler` indica la función callback que tratará los eventos correspondientes al componente de provisionameinto.

Disponemos de tres esquemas:

- `wifi_prov_scheme_softap`: se crea un punto de acceso temporal desde el nodo a provisionar. El provisionador debe conectarse a dicho punto de acceso y transmitir las credenciales via WiFi, utilizando el servidor HTTP creado por el componente de provisionamiento.
- `wifi_prov_scheme_ble`: el nodo crea un servidor GATT al que se puede conectar el provisionador utilizando Bluetooth Low Energy (BLE), que usará la tabla GATT para comunicar las credenciales Wifi.
- `wifi_prov_scheme_console`: transporta la información vía puerto serie

Una vez provisionado el nodo, las credenciales se almacenan en la flash (NVS), quedando disponibles para el siguiente reinicio del nodo.

Comprobación del estado de provisionamiento

Es posible comprobar el estado de provisionamiento de un dispositivo mediante una invocación a `wifi_prov_mgr_is_provisioned()`, que chequea si las credenciales de conexión WiFi están almacenadas en la memoria no volátil (NVS).

Aunque existen distintos métodos para eliminar la información de provisionamento almacenada en la NVS, utilizaremos el mecanismo proporcionado por `idf.py` para eliminar su contenido. Para ello, ejecutaremos:

```
idf.py erase_flash
```

Parámetros de inicialización del servicio de provisionamiento WiFi

Al inicializar el componente de provisionamiento, es necesario especificar un nombre de servicio y una clave. Esto se traduce en:

- SSID y contraseña para el modo SoftAP (es decir, cuando el esquema de provisionamiento se ha configurado como `wifi_prov_scheme_softap`).
- Nombre del dispositivo BLE para el modo BLE (es decir, cuando el esquema de provisionamiento se ha configurado como `wifi_prov_scheme_ble`).

Además, internamente el componente de provisionamiento utiliza el mecanismo de comunicación *protocomm*, que permite dos niveles de seguridad en la comunicación de credenciales de provisionamiento:

- Nivel 1 de seguridad, que consiste en un *handshake* previo entre ambos extremos, con intercambio de claves y utilización de una prueba de posesión (PoP, *proof of possession*), y utilizando encriptación AES para el intercambio de mensajes.
- Nivel 0 de seguridad, que consiste en un intercambio de credenciales utilizando texto plano y sin *PoP*.

Así, un ejemplo de inicialización del servicio de provisionamiento podría resultar en el siguiente código:

```
const char *service_name = "my_device";
const char *service_key  = "password";

wifi_prov_security_t security = WIFI_PROV_SECURITY_1;
const char *pop = "abcd1234";

ESP_ERR_CHECK( wifi_prov_mgr_start_provisioning(security, pop, service_name, service_key) );
```

El servicio de provisionamiento finalizará automáticamente al conectar a un AP con éxito (es decir, al obtener IP desde el mismo). En cualquier caso, puede también detenerse de forma manual en cualquier momento a través de la invocación a `wifi_prov_mgr_stop_provisioning()`.

Espera a la finalización del proceso de provisionamiento

Típicamente, las aplicaciones de usuario en el ESP32 deberán esperar a que el proceso de provisionamiento finalice antes de proceder. En ese momento, y antes de proceder, liberarán los recursos que se alojaron para el proceso de provisionamiento, y comenzarán con su lógica habitual.

Existen dos mecanismos para conseguir este efecto:

- En primer lugar, la solución más simple consiste en utilizar una invocación **bloqueante** a la rutina `wifi_prov_mgr_wait()`:

```
// Inicialización del servicio de provisionamiento
ESP_ERR_CHECK( wifi_prov_mgr_start_provisioning(security, pop, service_name, service_key) );

// Espera a la compleción del provisionamiento
wifi_prov_mgr_wait();

// Liberación de recursos
wifi_prov_mgr_deinit();

// A partir de aquí, comenzaría la lógica habitual de la aplicación
// ...
```

- El segundo mecanismo estaría basado en eventos (es decir, sería *no bloqueante*), interceptando y trabajando sobre eventos de tipo `WIFI_PROV_EVENT` e invocando a `wifi_prov_mgr_deinit()` cuando el identificador de evento sea `WIFI_PROV_END`:

```
static void event_handler(void* arg, esp_event_base_t event_base,
                          int event_id, void* event_data)
{
    if (event_base == WIFI_PROV_EVENT && event_id == WIFI_PROV_END) {
        /* Liberar recursos una vez el proceso de provisionamiento ha finalizado */
        wifi_prov_mgr_deinit();
    }
}
```

Herramientas de provisionamiento para dispositivos móviles

Existen aplicaciones preparadas por Espressif para llevar a cabo el proceso de provisionamiento sobre ESP32. Estas aplicaciones están disponibles tanto para dispositivos Android como IOS, en las versiones con transporte BLE o SoftAP:

- Android:
  - Provisionamiento BLE.
  - Provisionamiento SoftAP.
- IOS:
  - Provisionamiento BLE.
  - Provisionamiento SoftAP.

> **Tarea**
>
> Utilizando las aplicaciones correspondientes a tu dispositivo móvil, tanto para el uso de BLE como de SoftAP, provisiona tu ESP32 utilizando las credenciales que correspondan a tu red WiFi. Recuerda, antes de cada repetición del experimento, utilizar la orden `idf.py erase_flash` para eliminar información de provisionamiento de sesiones anteriores. Comprueba el funcionamiento de los distintos niveles de seguridad.
>
> Añade a tu informe las capturas de pantalla correspondientes a la salida del ESP32 que evidencien que el proceso de provisionamiento se ha realizado correctamente.

Estas aplicaciones funcionan mediante una comunicación muy sencilla con el ESP32 no provisionado, cuyos mecanismos dependen del transporte utilizado; en el caso de BLE, se crea una tabla GATT con distintas características que serán utilizadas para escribir (enviar) datos en el dispositivo. Veremos qué es una tabla GATT en próximas prácticas.

En el caso de `softAP`, se crean una serie de *endpoints* (URIs HTTP) que permiten, de forma sencilla, leer y escribir aquellos datos que deseamos comunicar al otro extremo de la comunicación.

La siguiente tabla resume los *endpoints* creados por las versiones estándar del protocolo de provisionamiento (pueden ser modificados o adaptados en función de la información adicional que deseemos intercambiar):

	Endpoint (BLE + Servidor GATT)	**URI (SoftAP + HTTP)**
Establecimiento de sesión	prov-session	http://IP:80/prov-session
Escaneo de redes disponibles	prov-scan	http://IP:80/prov-scan
Configuración de provisionamiento	prov-config	http://IP:80/prov-config
Versión del protocolo	proto-ver	http://IP:80/proto-ver

Los detalles de este tipo de protocolo de provisionamiento quedan como ejercicio adicional al alumno, y van más allá del objetivo de la práctica. No obstante, es conveniente disponer de algún mecanismo que permita inspeccionar lo que hacen, para determinar por ejemplo si el intercambio de credenciales se realiza como texto plano (en claro) o cifrado. Para ello podemos utilizar una herramienta de línea de comandos proporcionada con el SDK de Espressif (ESP-IDF), llamada `esp_prov.py`, que se encuentra en el directorio `tools/esp_prov` de la instalación.

Nota

Antes de utilizar el programa, debes instalar las dependencias respectivas utilizando las órdenes (desde el propio directorio `tools/esp_prov`):

```
pip install -r requirements.txt
pip install -r requirements_linux_extra.txt
```

Su uso es sencillo, y puede consultarse ejecutando `python esp_prov.py -h`. Básicamente, se trata de usar esta herramienta como provisionador para un dispositivo que usa el componente de provisionamiento en modo `softAP`. Conectaremos entonces el ordenador a la wifi temporal del nodo. El componente de provisionamiento estará escuchando en el puerto 80 del nodo (192.168.4.1), la aplicación se conectará a dicho servidor y realizará el provisionamiento, pasando las credenciales indicadas en la línea de comandos:

```
python esp_prov.py --transport softap --service_name "192.168.4.1:80" --sec_ver 0 --ssid SSID_EJEM
```

Podemos monitorizar los paquetes enviados utilizando cualquier sniffer de red, como por ejemplo WireShark. Esto nos permitirá ver el contenido de los paquetes enviados por la red y determinar si la contraseña se envía en claro o si por el contrario está cifrada.

Tarea

Realiza el proceso de provisionamiento desde línea de comandos siguiendo el procedimiento explicado arriba. Captura el tráfico de red con Wireshark y comprueba que la contraseña se envía en claro (sin cifrar). A continuación, pasa a un modo seguro (opción `--sec_ver 1`) y observa cómo las claves se intercambian cifradas. Documenta esta tarea en el informe de la práctica.

Parte 2. Modos de ahorro de consumo WiFi

En la versión actual de ESP-IDF, el *framework* soporta distintos modos de ahorro de energía, con soporte tanto a nivel de dispositivo (*station*) como de punto de acceso (*AP*). Todos estos modos se basan en las características de ahorro de consumo contempladas en el estándar 802.11

(concretamente en el modo *Modem-sleep*).

El modo *Modem-sleep* trabaja exclusivamente cuando un dispositivo está configurado como *station*, y se encuentra conectado a un AP. Si el modo *Modem-sleep* está activo, el dispositivo varía su estado entre *activo* y *sleep* periódicamente. En el modo *sleep*, tres de los principales componentes del subsistema de comunicacion inalámbrica (PHY, BB y RF) se desconectan para reducir el consumo energético. Pese a permanecer desconectados, la estación sigue conectada al AP en todo momento. Este modo soporta además dos submodos de ahorro de consumo: *mínimo* y *máximo*.

- En el modo *mínimo*, la estación se despierta cada *DTIM* para recibir un beacon. Debido a que los mensajes de difusión (*broadcast*) se transmiten tras cada DTIM, en este caso no se perderán y serán recibidos por las estaciones. Sin embargo, el ahorro energético puede ser reducido si el *DTIM* es breve (además, *DTIM* está determinado por el AP, por lo que la estación no tiene control sobre este parámetro).

- En el modo *máximo*, la estación se despierta tras cada intervalo de escucha para recibir un beacon. Este intervalo de escucha no tiene que coincidir con el valor de *DTIM*, y de hecho suele fijarse a un valor mayor para conseguir mayor ahorro energético. Los datos de *broadcast* podrían perderse usando este modo si la estación está en estado de reposo mientras expira el temporizador *DTIM*. El valor del intervalo de escucha puede configurarse mediante una invocación a `esp_wifi_set_config()` antes de conectar al AP.

Para activar el modo *mínimo*, es neceario invocar a la rutina `esp_wifi_set_ps(WIFI_PS_MIN_MODEM)`; para activar el modo *máximo*, es neceario invocar a la rutina `esp_wifi_set_ps(WIFI_PS_MAX_MODEM)`, ambos tras la invocación de `esp_wifi_init()`. Los modos de ahorro se activarán al conectar al AP, y se desactivarán al desconectar.

Es posible desactivar los modos de ahorro mediante una invocación a `esp_wifi_set_ps(WIFI_PS_NONE)`. Obviamente, esto aumentará el consumo, pero reducirá la latencia en la recepción de mensajes. Con el modo de ahorro activado, la recepción de los mensajes se retrasará tanto como el período *DTIM* (en modo mínimo ahorro) o el período de escucha (modo máximo ahorro). El modo por defecto es `WIFI_PS_MIN_MODEM`.

En modo punto de acceso, ESP-IDF no soporta todos los modos de ahorro dictados en la especificación WiFi. Concretamente, un AP programado vía ESP-IDF sólo cacheará (almacenará temporalmente) los paquetes de tipo *unicast* para las estaciones conectadas a dicho AP, pero no paquetes *multicast* para dichas estacioens. Así, con el modo de ahorro activo, las estaciones podrían perder paquetes *multicast*.

El ejemplo `examples/wifi/power_save` ilustra mediante un código sencillo la configuración de una estación en los dos modos de ahorro energético. Estos modos pueden configurarse a través del menú de configuración; además, se ofrece una opción para modificar el tiempo de escucha en el caso del modo de ahorro *máximo*.

Tarea

Compila, flashea y ejecuta el código de ejemplo utilizando los tres modos disponibles (sin ahorro, con ahorro mínimo y con ahorro máximo). En todos los casos, conecta tu ESP32 a un punto de acceso y, desde un portátil conectado al mismo AP, ejecuta una serie de `pings` hacia el nodo y observa como cambia el tiempo de respuesta de los pings. En el caso del ahorro máximo, varía el tiempo de escucha y mira si afecta al tiempo de recepción de los pings.

Para cada modo, representa gráficamente el tiempo de respuesta de la estación en una gráfica para cada petición `ping`, relacionando su comportamiento con los tiempos *DTIM* y de escucha. Añade a tu informe de la práctica una descripción del ejercicio. Comenta y discute los resultados que observes.
