



## 1. DIY Mode:

SONOFF Basic R3, RFR3 and Mini can operate as either eWeLink Mode or DIY Mode. The device is able to switch to DIY Mode when the GPIO 16 is at low-level.

Operation instruction as below:

Operating Mode Switch			
Basic R3/RFR3	GPIO 16 at High-level ( factory default)	eWeLink Mode	
	GPIO 16 at Low-level	DIY Mode	

Note: the user settings will be cleaned when change the operating mode from one to another.

## 2. eWeLink LAN Discovery Mechanism

eWeLink LAN Discovery implements IETF Multicast DNS protocol and DNS-Based Service Discovery protocol. [1]

### 2.1 Device Service Publish Process

The device must publish its own service (i.e. device capability) according to the mDNS/DNS-SD standard discovery protocol when the device is connected to LAN (Local Area Network).

The fields defined by eWeLink are as follows:

Attribute	Description	Example
IP Address	The LAN IP Address is obtained through DHCP instead of the Link-Local address of IPv4/IPv6	
Host Name	The Host Name must be unique in LAN; Format: eWeLink_[Device ID]	eWeLink_10000000d0
Service Type	_ewelink._tcp	
Service Instance Name	The Service Instance Name must be unique in LAN; Max. 63 bytes ( 21 UTF8 Characters)	
TXT Record	One or more strings; No exceeded 255 bytes for each string; No exceeded 1300 bytes for the entire TXT record;	

TXT Record note:

- TXT Record must contain below strings:  
"txtvers=1", "id=[deviceId]", "type=[device type]", "apivers=[device API interface version]", "encrypt=[Boolean]",  
"seq=[TXT Record serial number]", "data1=[device information]";
- Optional strings:

"data2=[device information]", "data3=[device information]", "data4=[device information]",  
"iv=[Base64 encoded IV for encryption]";

- “seq=[TXT record sequence number]” indicates the order in which the TXT records are updated (the order in which the device status is updated). It is recommended to be a positive integer that increments from 1 (reset to 1 when the device restarts);
- "encrypt=true" indicates that the device information (data1, data2, data3, data4) is encrypted and then encoded in Base64. In this case, the string "iv=[Base64 encoded IV for encryption]" is mandatory;
- The default password must be the API Key of the device. The device must provide an API interface for the device owner to change the password and enable/disable the device information encryption function. The key used for encryption is the MD5 hash of the device password (16 bytes);
- The initialization vector iv used for encryption is a 16-byte random number, Base64 encoded as a string in a TXT Record or JSON;
- The encryption algorithm must be "AES-128-CBC/PKCS7Padding" (AES 128 Cipher Block Chaining (CBC) with PKCS7 Padding);
- When the device information (unencrypted or encrypted string) is longer than 249 bytes, the first 249 bytes must be stored in data1, and the remaining bytes are divided by length 249, which are stored in data2, data3, and data4.

## 2.2 Discovery Process for Device Service

### 2.2.1 Device connects to a specified default SSID and password WiFi network

A default SSID and Password WiFi is required as LAN.

LAN Attribute	Description
Default SSID	sonoffDiy
Default Password	20170618sn

### 2.2.2 Discovery of device status and info

The discovery process must follow the mDNS/DNS-SD Discovery protocol to discover the eWeLink device services. Such "Discovery" works in Global Search way.

**Global Search:** The discovery process should search in the LAN for all devices with the service type \_ewelink.\_tcp through the DNS PTR record, parses out the corresponding “Device ID” through the DNS TXT Record, and then gets the device configuration and status information according to “Device Information” in DNS TXT Record.

More info refers to demo application source code [2]:

When the device and the PC (DIY demo application installed) are in the same LAN (SSID: sonoffDiy; Password: 20170618sn), the DIY demo is able to discover eWeLink device.

Note:

- When the “device type” of the device service does not match with the “device type” of the corresponding device in the App

(associated by the device ID), or the device API version of the device service is higher than the device API version of the corresponding device in the App. the App must not parse the “Device Information” and call the device API. Instead, it should notify the user of the “unavailable LAN control for the device” reason and suggest users to upgrade the App;

- The App should obtain the IP address of the device through the DNS A record when the device API is about to be called;
- Both device (e.g. SONOFF Basic R3) and App should be in the same WiFi network considering as LAN.

### 3. Device Control Protocol

The device must open the HTTP server in the port declared by the DNS SRV record before the device publishes its services; the device publishes the capabilities through a HTTP-based RESTful API. Because of the LAN's security and device's limited computing power, this document recommends that the device provides HTTP instead of HTTPS interface.

#### 3.1 RESTful API Request and Response Format

RESTful API Request works in POST method and JSON formatted request body.

For example:

```
{
  "sequence": "1546954710268",
  "deviceid": "100000140e",
  "encrypt": false,
  "data": {"switch": "on"}
}
```

Attribute	Type	Optional	Description
sequence	String	No	The serial number of this request, it is recommended to use the timestamp
deviceid	String	No	The device ID for this request. After receiving the request, the device must determine whether the deviceid is supported or not
encrypt	Boolean	No	True means the data field is encrypted and encoded in Base64, false means the data field is not encrypted. When the device information encryption function is enabled, the request body encryption must be true, otherwise it must be false
iv	String	Yes	Required when encrypt is true. The Base64 value of initialization vector when encrypting
data	String/Object	No	Encrypt is true, String type, encrypt is false, Object type, Specific device information setting when controlling the device

RESTful API Response works in 200 OK HTTP response code and JSON formatted response body.

For example:

```
{
  "seq": 2,
```

```

"sequence": "1546954710268",
"error": 0,
"encrypt": false,
"data": {"signalStrength": -67}
}

```

Attribute	Type	Optional	Description
seq	Number	No	The order of device status update (also the order of TXT Record update)
sequence	String	No	The serial number of the request corresponding to this response (Sequence field value in the request body)
error	Number	No	<p>Whether the device successfully sets the specified device information</p> <ul style="list-style-type: none"> <li>- <b>0</b>: successfully</li> <li>- <b>400</b>: The operation failed and the request was formatted incorrectly. The request body is not a valid JSON format</li> <li>- <b>401</b>: The operation failed and the request was unauthorized. Device information encryption is enabled on the device, but the request is not encrypted</li> <li>- <b>404</b>: The operation failed and the device does not exist. The device does not support the requested deviceid</li> <li>- <b>422</b>: The operation failed and the request parameters are invalid. For example, the device does not support setting specific device information.</li> </ul>
encrypt	Boolean	Yes	Required when the data field is present. True means the data field is encrypted and encoded in Base64, false means the data field is not encrypted. The encryption of the response body must be true when device information encryption is enabled, otherwise it must be false
iv	String	Yes	<p>Required when encrypt is true.</p> <p>The Base64 value of initialization vector when encrypting</p>
data	String/Object	No	Encrypt is true, String type, encrypt is false, Object type, it returns specific device information when controlling the device

### 3.2 Device Control API (SONOFF Basic R3, RFR3 and Mini)

The device type is diy\_plug(type=diy\_plug) and the device API interface version is 1 (apivers=1).

#### 3.2.1 On/Off Status

```

POST /zeroconf/switch
{
  "sequence": "1546954710268",
  "deviceid": "100000140e",
  "encrypt": false,
  "data": {"switch": "on"}
}

```

Attribute	Type	Optional	Description
switch	String	No	on: turn the switch on, off: turn the switch off

### 3.2.2 Power On State

```
POST /zeroconf/startup
{
  "sequence": "1546954710268",
  "deviceid": "100000140e",
  "encrypt": false,
  "data": {"startup": "stay"}
}
```

Attribute	Type	Optional	Description
startup	String	No	on: the device is on when power supply is recovered off: the device is off when power supply is recovered stay: the device status keeps as the same as the state before power supply is gone

### 3.2.3 WiFi Signal Strength

POST /zeroconf/signal\_strength

Request body:

```
{
  "sequence": "1546954710268",
  "deviceid": "100000140e",
  "encrypt": false,
  "data": {}
}
```

Response body:

```
{
  "seq": 2,
  "sequence": "1546954710268",
  "error": 0,
  "encrypt": false,
  "data": {"signalStrength": -67}
}
```

Attribute	Type	Optional	Description
signalStrength	Number	No	The WiFi signal strength currently received by the device, negative integer, dBm

### 3.2.4 Inching

POST /zeroconf/pulse

```
{
  "sequence": "1546954710268",
  "deviceid": "100000140e",
  "encrypt": false,
  "data":
  {
    "pulse": "on",
    "pulseWidth": 2000
  }
}
```

Attribute	Type	Optional	Description
pulse	String	No	on: activate the inching function, off: disable the inching function
pulseWidth	Number	Yes	Required when "pulse" is on, pulse time length, positive integer, ms, only supports multiples of 500 in range of 500~36000000

### 3.2.5 WiFi SSID and Password Setting

POST /zeroconf/wifi

```
{
  "sequence": "1546954710268",
  "deviceid": "100000140e",
  "encrypt": false,
  "data":
  {
    "ssid": "eWeLink",
    "password": "WeLoveIoT"
  }
}
```

Attribute	Type	Optional	Description
ssid	String	No	SSID of the WiFi network to which the device will connect
password	String	No	password of the WiFi network to which the device will connect

### 3.2.6 OTA Function Unlocking

POST /zeroconf/ota\_unlock

```
{
  "sequence": "1546954710268",
  "deviceid": "100000140e",
  "encrypt": false,
```

```
"data": {}  
}
```

The following failure codes are added to the error field of the response body:

- 500: The operation failed and the device has errors. For example, the device ID or API Key error which is not authenticated by the vendor's OTA unlock service;
- 503: The operation failed and the device is not able to request the vendor's OTA unlock service. For example, the device is not connected to WiFi, the device is not connected to the Internet, the manufacturer's OTA unlock service is down, etc.

### 3.2.7 OTA New Firmware

```
POST /zeroconf/ota_flash  
{  
  "sequence": "1546954710268",  
  "deviceid": "100000140e",  
  "encrypt": false,  
  "data":  
  {  
    "downloadUrl": "http://192.168.1.184/ota/new_rom.bin",  
    "sha256sum": "3213b2c34cecb3bb817030c7f025396b658634c0cf9c4435fc0b52ec9644667"  
  }  
}
```

Attribute	Type	Optional	Description
downloadUrl	String	No	The download address of the new firmware, only supports the HTTP protocol, the HTTP server must support the Range request header.
sha256sum	String	No	SHA256 checksum (hash) of the new firmware, it is used to verify the integrity of the new firmware downloaded

The following failure codes are added to the error field of the response body:

- 403: The operation failed and the OTA function was not unlocked. The interface "3.2.6 OTA function unlocking" must be successfully called first.
- 408: The operation failed and the pre-download firmware timed out. You can try to call this interface again after optimizing the network environment or increasing the network speed.
- 413: The operation failed and the request body size is too large. The size of the new OTA firmware exceeds the firmware size limit allowed by the device.
- 424: The operation failed and the firmware could not be downloaded. The URL address is unreachable (IP address is unreachable, HTTP protocol is unreachable, firmware does not exist, server does not support Range request header, etc.)
- 471: The operation failed and the firmware integrity check failed. The SHA256 checksum of the downloaded new firmware does not match the value of the request body's sha256sum field. Restarting the device will cause bricking issue.

**Note: The maximum firmware size is 508KB.**

## Reference:

[1]

- Multicast DNS protocol: IETF RFC 6762, <https://tools.ietf.org/html/rfc6762>
- DNS-Based Service Discovery protocol: IETF RFC 6763, <https://tools.ietf.org/html/rfc6763>
- Zero Configuration Networking: Zeroconf, <http://www.zeroconf.org/>
- Apple Bonjour Network Discovery and Connectivity: <https://developer.apple.com/bonjour/>
- Android Network Service Discovery: <https://developer.android.com/training/connect-devices-wirelessly/nsd>

[2]

- Sonoff DIY Mode Demo Application on Github, [https://github.com/itead/Sonoff\\_Devices\\_DIY\\_Tools](https://github.com/itead/Sonoff_Devices_DIY_Tools)