Wi-Fi for EPOSMote III with the ESP8266 Module

Note: read this article on MicroPython for the ESP8266 before continuing!

This page is a step-by-step guide for uploading the firmware of the ESP module. The factory firmware could be used, but it cannot handle secure connections properly. The solution is to create a custom firmware capable of handling secure connections and provide a UFSC self-signed certificate to the server.

Our goal on this page is to establish a secure connection between EPOSMote3 (eMote) and the IoT Server. To achieve this, a micro-controller will be used as an interface between the eMote and the network.

Installation and Setup

Software Requirements

The following list shows what you need to install/setup/prepare:

1. Install the Arduino IDE. The version used here is **v1.8.3**
2. Install the ESP8266 library on your system. The tutorial can be found at https://github.com/esp8266/Arduino. The version used for this tutorial is **v2.2.0**
3. You need to emit a P1 certificate to ensure your identity, the server uses this certificate to verify if you are part of the university. This can be done at https://p1.icpedu.rnp.br/default/public/default. The certificate will be saved in your browser, and it should be exported to a folder on your computer.
4. To upload the certificate to the ESP’s flash memory, you need a plugin into your Arduino IDE. Installation and Usage can be found at https://github.com/esp8266/arduino-esp8266fs-plugin. Link for the ESP File System plugin: ESP8266FS-0.3.0.zip

Hardware Requirements

A ESP module is required, there are many module versions for this chip and the one used on this article is the ESP-01. Use this link as reference https://github.com/esp8266/esp8266-wiki/wiki/Hardware_versions for module versions. The version 12 of the ESP module has SPI communication pins, but due to its pin size, this version couldn't be tested.

It is important to note that when you are uploading the certificate files or the firmware, the GPIO0 pin on the module must be connected to GROUND. And when the code is ready for production, GPIO0 must be disconnected from the GROUND
To write data into the ESP, a FTDI board is required. And the connections between the ESP and the FTDI are shown below.

Preparing the ESP

Make sure the GPIO0 of the module is connected to ground before you start the procedures.

Adding the certificate

As explained before, the certificate must be exported into your computer. Assuming this step is done and you have a certificate.p12 file ready, you need to break this file into a key and a certificate using openssl. Go to the folder where the certificate is, and type the following:

1. openssl pkcs12 -in certificate.p12 -nocerts -out key.pem -nodes
2. openssl rsa -outform der -in key.pem -out key.der
3. openssl pkcs12 -in certificate.p12 -nokeys -out cert.pem -nodes
4. openssl x509 -outform der -in cert.pem -out cert.der

This creates 2 new files that are going to be added into the ESP (cert.der and key.der). Copy those files inside your sketch/data directory and upload it to the ESP, as explained on the item 3 in Software Requirements section.

If you are successful, the certificates are written inside the ESP's flash memory.

Uploading the Firmware

Get the new firmware and add it to your sketch. Uploading the code is the same as it is for any Arduino board.

Current firmware version is 0.4: ESP-LishaFirmware_secure-v0.4
Older Version: ESP-LishaFirmware_secure-v0.3

The default Wifi network on the firmware is:
Name: LishaJoinville
Pass: 12345678

Before connecting the ESP module with EPOSMote module

Using the Arduino Serial IDE to test if your setup was done correctly. Put the commands and check if the response is expected.

The following image shows an example of use of the AT+SENDTSTP command to iot.lisha.ufsc.br/api/put.php. You can see that the server response is what we were expecting.
Connecting ESP and eMote 3

Pin Connections

The ESP and the eMote3 communicate through UART, using the Rx-Tx pins available on each board. Make sure you have the following connections ready:

1. ESP's RX pin connected to eMote's C4 pin (UART1 TX)
2. ESP's TX pin connected to eMote's C3 pin (UART1 RX)
3. ESP's CH and Vin connected to external 3v3 source (ESP module requires over 100mA to run and eMote can't provide that)
4. ESP's GND connected to eMote's GND pin and the external source's GND

Available Commands and Responses

It is important to note that when the ESP module is powered on, it connects to the default network (redeUFSCSemFio), and all the commands need to contain "\r" as the last character. The following table shows the available commands and responses.

For the latest firmware version (v0.4), the ESP's use is simplified. Instead of sending raw commands through UART, just import <esp8266.h> and use the class ESP8266.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Response</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT+SYSTEMCHECK</td>
<td>Gets the status of the ESP Module</td>
<td>CONNECTION=(OK/FAIL)&amp;FILESYSTEM=(OK/FAIL)</td>
<td>eMote: AT+SYSTEMCHECK\r -&gt; ESP: CONNECTION=FAIL&amp;FILESYSTEM=OK\r</td>
</tr>
<tr>
<td>AT+SYSTEMREADY</td>
<td>Checks if the module is ready</td>
<td>OK/FAIL</td>
<td>eMote: AT+SYSTEMREADY\r -&gt; ESP: OK\r</td>
</tr>
<tr>
<td>AT+RESPONSETIME</td>
<td>Gets the response time of the last request</td>
<td>OK=(integer in ms)</td>
<td>eMote: AT+RESPONSETIME\r -&gt; ESP: OK=25\r</td>
</tr>
<tr>
<td>AT+GETHOST</td>
<td>Gets the host of the connection. Default is iot.ufsc.br</td>
<td>OK=(host name)</td>
<td>eMote: AT+GETHOST\r -&gt; ESP: OK=iot.ufsc.br\r</td>
</tr>
<tr>
<td>AT+GETROUTE</td>
<td>Gets the route reached by the connection. Default is /api/put.php</td>
<td>OK=(route)</td>
<td>eMote: AT+GETROUTE\r -&gt; ESP: OK=/api/put.php\r</td>
</tr>
<tr>
<td>AT+GETPORT</td>
<td>Gets the port used on the connection. Default is 443.</td>
<td>OK=(port)</td>
<td>eMote: AT+GETPORT\r -&gt; ESP: OK=443\r</td>
</tr>
<tr>
<td>AT+SETHOST</td>
<td>Sets the host of the connection</td>
<td>OK</td>
<td>eMote: AT+SETHOST=www.google.com\r -&gt; ESP: OK\r</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>ESP Response</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>AT+SETROUTE</td>
<td>Sets the route of the connection</td>
<td>eMote: AT+SETROUTE=/api/get_data.php\r -&gt; ESP: OK\r</td>
<td></td>
</tr>
<tr>
<td>AT+SETPORT</td>
<td>Sets the port used for the connection</td>
<td>eMote: AT+SETPORT=80\r -&gt; ESP: OK\r</td>
<td></td>
</tr>
<tr>
<td>AT+SETSSID</td>
<td>Sets the default wifi network name. Default is <em>RedeWifi</em>, pass: 12345678.</td>
<td>eMote: AT+SETSSID=eduroam\r -&gt; ESP: OK\r</td>
<td></td>
</tr>
<tr>
<td>AT+SETPASSWORD</td>
<td>Sets the default wifi password</td>
<td>eMote: AT+SETPASSWORD=12345678\r -&gt; ESP: OK\r</td>
<td></td>
</tr>
<tr>
<td>AT+CONNECTWIFI</td>
<td>Connects to the chosen network</td>
<td>eMote: AT+CONNECTWIFI\r -&gt; ESP: OK\r</td>
<td></td>
</tr>
<tr>
<td>AT+GETTIMESTAMP</td>
<td>Gets the actual timestamp from network</td>
<td>eMote: AT+GETTIMESTAMP\r -&gt; ESP: (unix timestamp)\r</td>
<td></td>
</tr>
<tr>
<td>AT+GETHEAPSIZE</td>
<td>Gets the esp heap size in bytes</td>
<td>eMote: AT+GETHEAPSIZE\r -&gt; ESP: (heap size in bytes)\r</td>
<td></td>
</tr>
<tr>
<td>AT+SENDTSTP</td>
<td>Sends data over the selected route to the selected host. It is used to authenticate the user.</td>
<td>eMote: AT+AUTH=(82 bytes string)\r -&gt; ESP: OK=DENIED\r</td>
<td></td>
</tr>
<tr>
<td>AT+SEND!32</td>
<td>Sends data over the selected route to the selected host. The number is used to define the message size in bytes. The response will be the server http response</td>
<td>eMote: AT+SEND!4=1234\r -&gt; ESP: OK=HTTP/1.1 400 Bad Request\r</td>
<td></td>
</tr>
</tbody>
</table>
Example
The following example shows how to declare the UART inside EPOS and send commands to the ESP module.

This example works with the version 0.4 of the ESP firmware.

```cpp
#include <utility/ostream.h>
#include <utility/string.h>
#include <alarm.h>
#include <uart.h>
#include <machine/cortex/esp8266.h>
#include <riffs.h>

using namespace EPOS;
OStream cout;

class Data_Record
{
public:
    Data_Record() {
        for(int i = 0; i < 84; i++)
            a[i] = 0;
    }

    char a[84];
};

int main()
{
    Delay(2000000);

    UART uart(1, 115200, 8, 0, 1);
    GPIO rst('B', 3, GPIO::OUT);

    char host[] = "iot.lisha.ufsc.br";
    char route[] = "/api/put_hydro.php";

    cout << "ESP Com TEST" << endl;
    ESP8266 esp(&uart, &rst);
    cout << "ESP8266 created" << endl;

    esp.command_mode();

    esp.config_endpoint(443, host, sizeof(host) - 1, route, sizeof(route) - 1);
    cout << "End point configured" << endl;

    esp.connect("LishaJoinville", 14, "12345678", 8);
    Data_Record aux;
    char res[255];

    cout << endl << "Posting..." << endl;
    int res_size = esp.post(&aux, sizeof(aux), res);
```
Troubleshooting guide

If problems are found, here are some things you can double check to make sure everything is setup correctly.

1. Make sure the ESP is powered with 3.3V and the source supply can provide at least 200mA.
2. The firmware default wifi network is LishaJoinville (pass: 12345678). Make sure this network exists, or change this setting on the firmware.
3. Make sure the traits file of your application is setup correctly according to the example above.
4. Are the components of your EPOS branch working as expected? A reliable UART is needed to transfer data between the EPOS mote and the ESP.
5. Is the ESP's GPIO-0 disconnected? If you power the ESP with GPIO-0 connected to ground, the module will boot on the flashing mode and the firmware won't run. Make sure GPIO-0 is not connected to anything.
6. Double check the connections between the ESP and the eMote. So many wires can get us confused sometimes. **ESP's RX on eMote's TX** and **ESP's TX on eMote's RX**.
7. Certificates load should be done inside the setup method, connection might fail if the certificates are loaded during execution time.
8. Due to a wrong implementation of an internal TLS library, there was a memory leak causing the ESP to crash and restart. If you face some reset issues, make sure you are using the same version described in this tutorial.