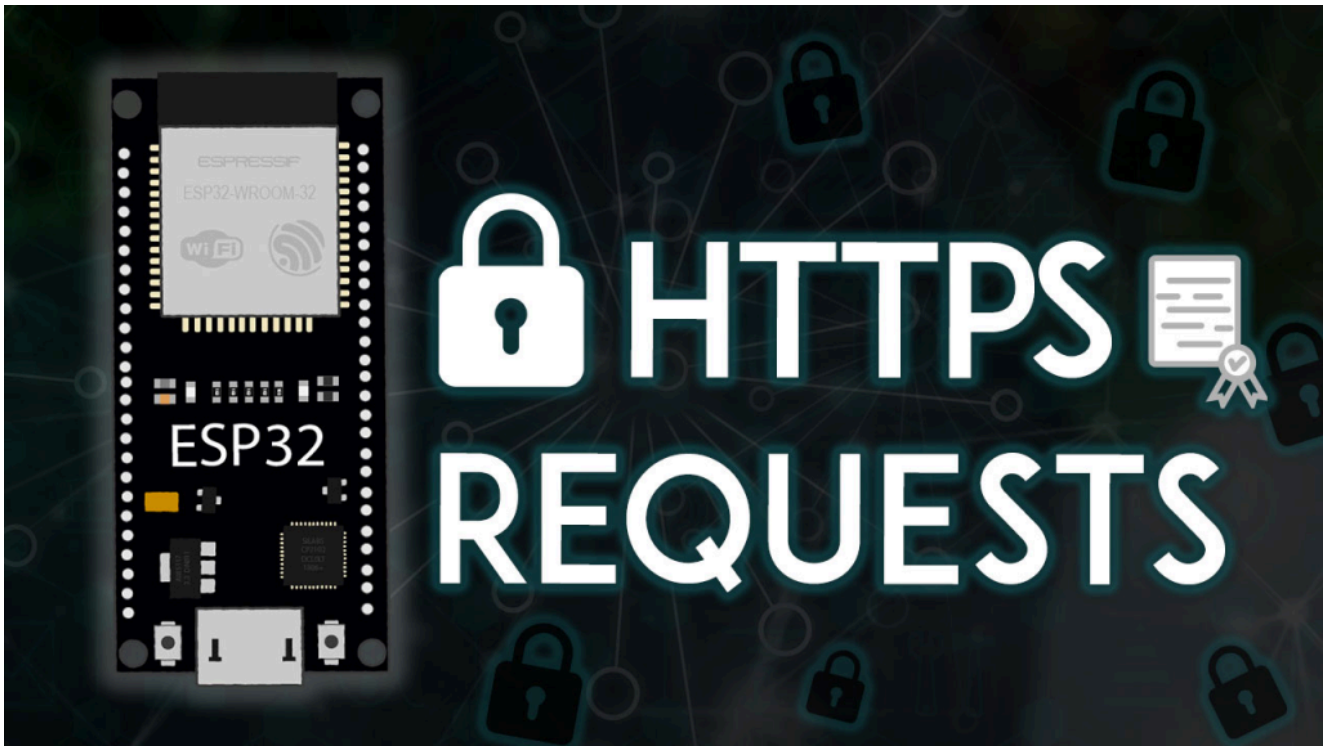
ESP32 HTTPS Requests (Arduino IDE)

In this guide, you'll learn how to make HTTPS requests with the ESP32. We'll introduce you to some HTTPS fundamental concepts and provide several examples (with and without certificates) using two different libraries: `HttpClient` and `WiFiClientSecure`.



Using an ESP8266 board? Check this tutorial instead: ESP8266 NodeMCU HTTPS Requests

Table of Contents

Throughout this article, we'll cover the following subjects:

- What is HTTPS?
  - Why do you need HTTPS with the ESP32?
- SSL/TLS Certificates
  - Certificate Chain
  - Certificates Expiration Date

- ESP32 HTTPS Requests with Certificate
- ESP32 HTTPS Requests without Certificate
- HTTPS Requests with the ESP32 (HTTPClient)
  - ESP32 HTTPS Requests with Certificate
  - ESP32 HTTPS Requests without Certificate

Introduction

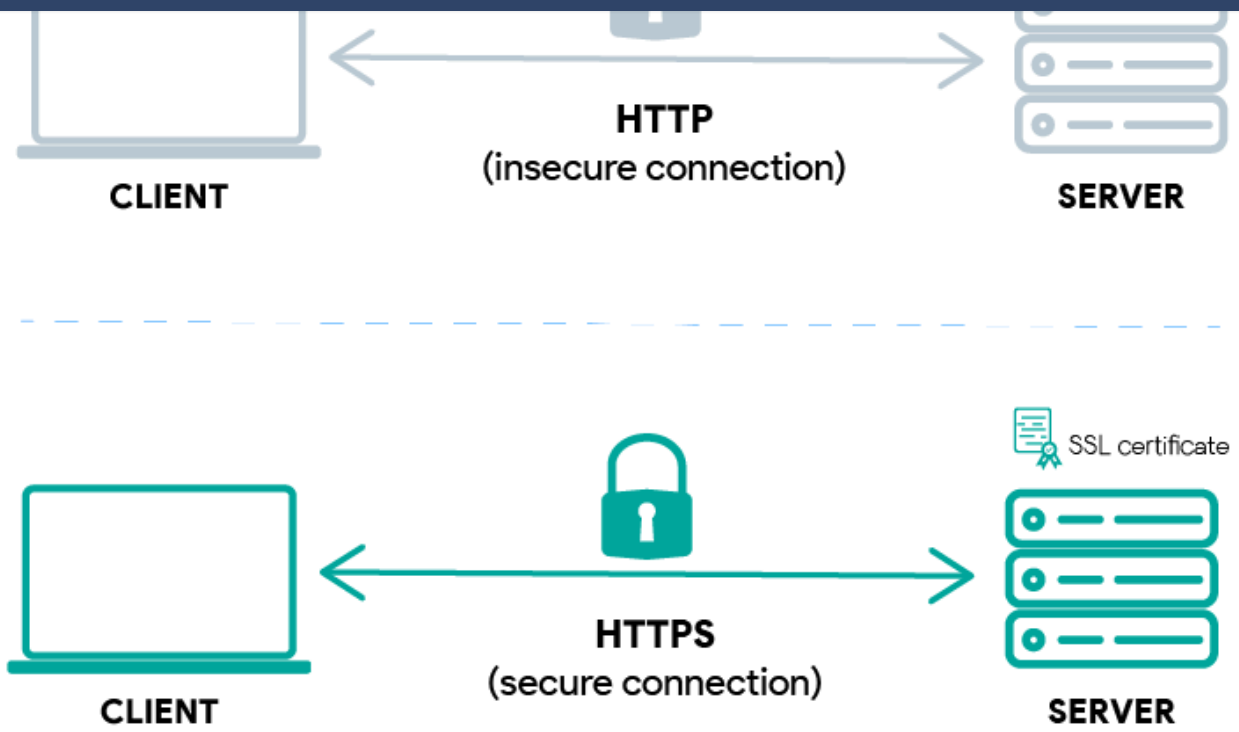
To understand how to make HTTPS requests with the ESP32, it's better to be familiar with some fundamental concepts that we'll explain next. We also recommend taking a look at the following article:

- ESP32/ESP8266 with HTTPS and SSL/TLS Encryption: Basic Concepts

What is HTTPS?

HTTPS is the secure version of the HTTP protocol, hence the "S", which stands for secure.

HTTP is a protocol to transfer data over the internet. When that data is encrypted with SSL/TLS, it's called HTTPS.

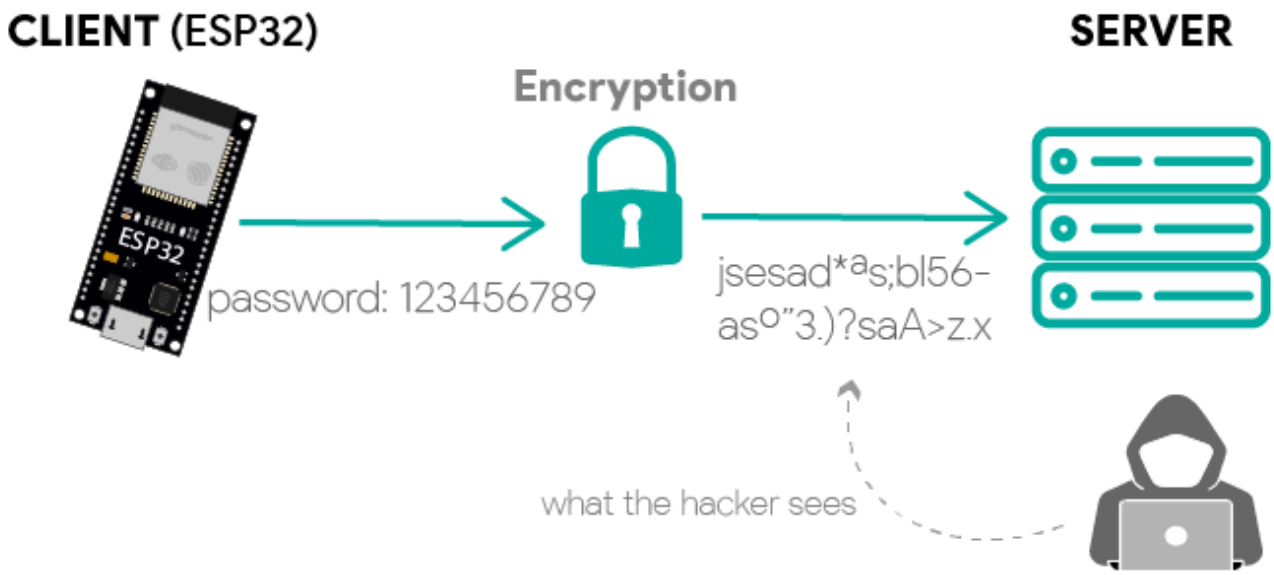


To simplify, HTTPS is just the HTTP protocol but with encrypted data using SSL/TLS.

Why do you need HTTPS with the ESP32?

Using HTTPS ensures the following:

1) Encryption: all traffic between the ESP32 and a server will be encrypted—no one can spy on your requests and passwords, they will only see gibberish.



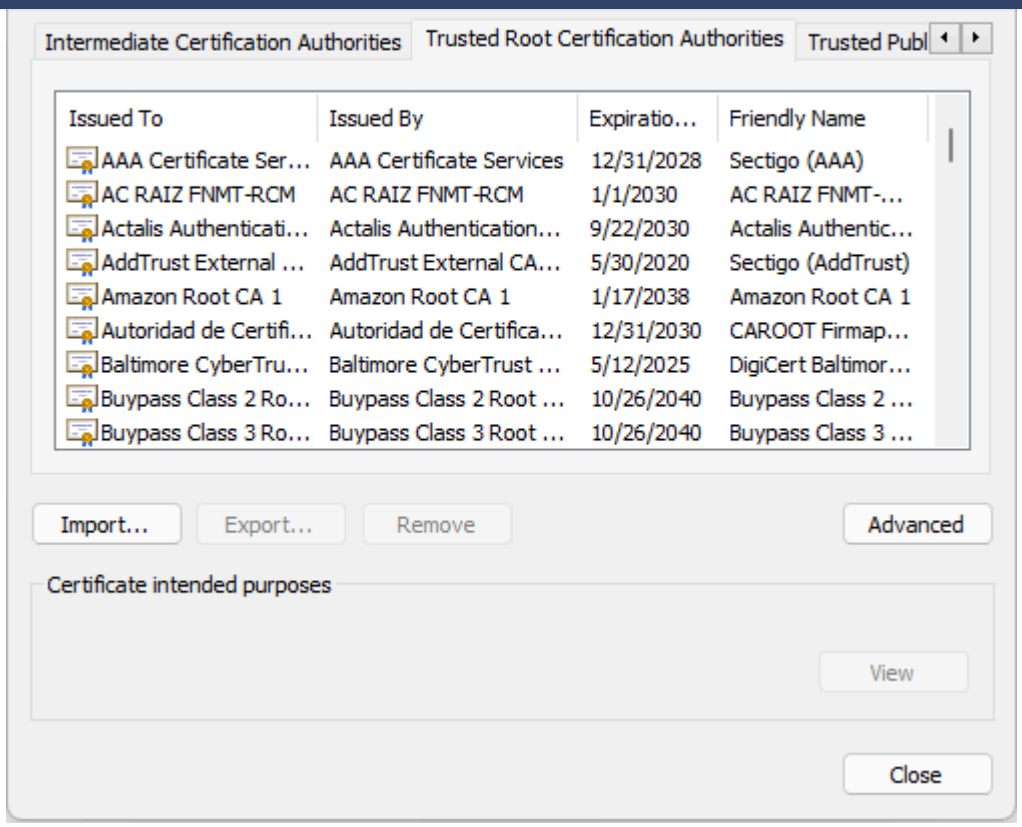
2) Server trust (identification): when using HTTPS, via TLS/SSL certificates, you ensure you are connected to the server you would expect—this means, you always know to who you are connected to.

To make sure we are connected to the right server, we need to check the server certificate on the ESP32. This means we need to download the server certificate and hard code it on our sketch so that we can check if we're actually connected to the server we are expecting.

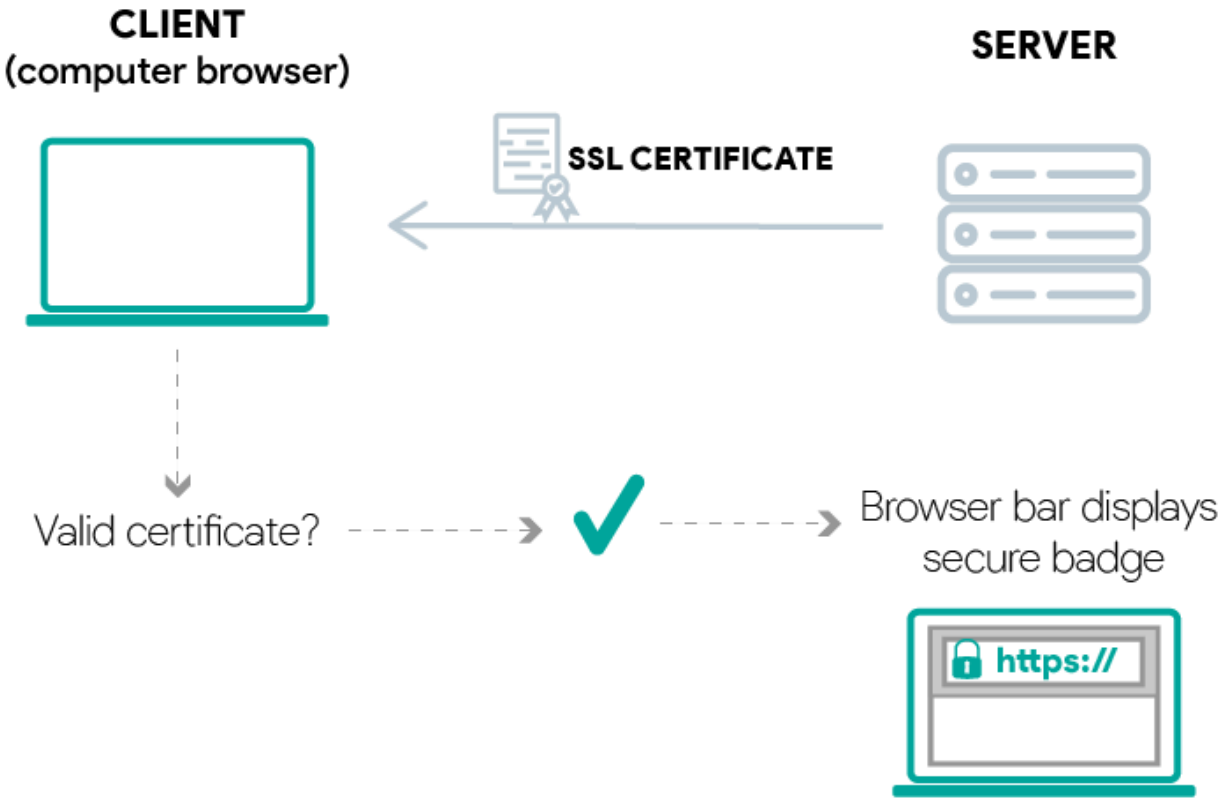
TLS/SSL Certificates

SSL certificates are issued by legitimate **Certificate Authorities**. One of the most known is LetsEncrypt. Certificate Authorities confirm the identity of the certificate owner and provide proof that the certificate is valid. The certificate also contains the server's public key for asymmetrically encrypted communication with a client.

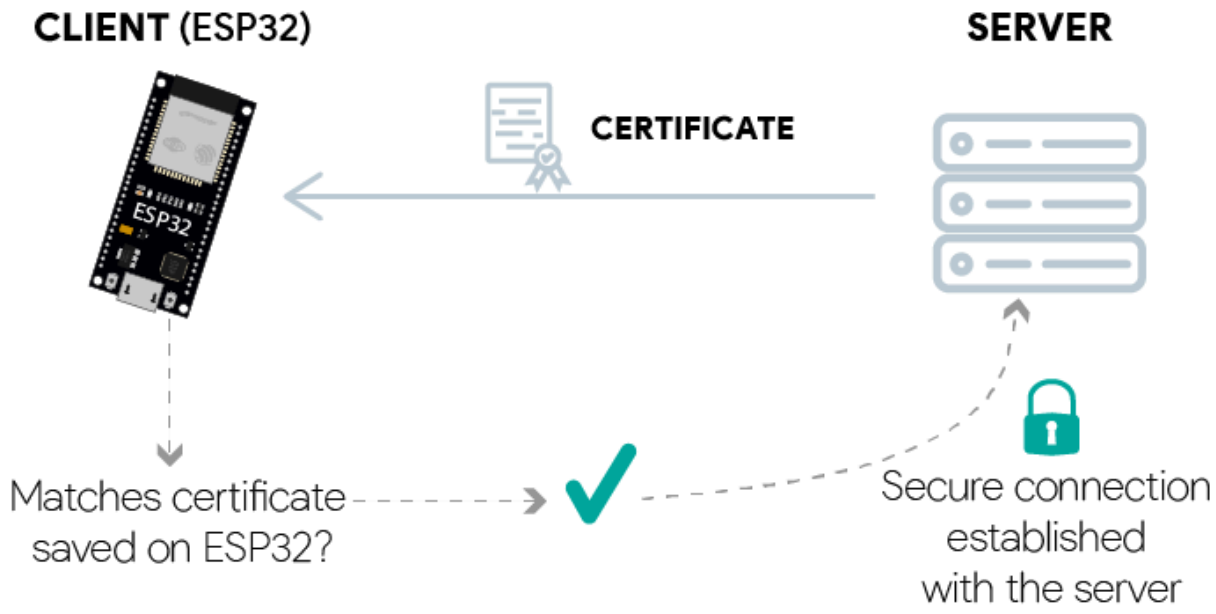
When a Certificate Authority issues a certificate, it signs the certificate with its root certificate. This root certificate should be on the database of trusted certificates called a **root store**. Your browser and the operating system contain a database of root certificates that they can trust (root store). The following screenshot shows some of the trusted root certificates.



So, when you connect to a website using your browser, it checks if its certificate was signed by a root certificate that belongs to its root store. New root certificates are added or deleted to the root store with each browser update.



connect to.



But, it's also possible to upload a root store to your board to have more options, and don't have to worry about searching for a specific website's certificate.

Certificate Chain

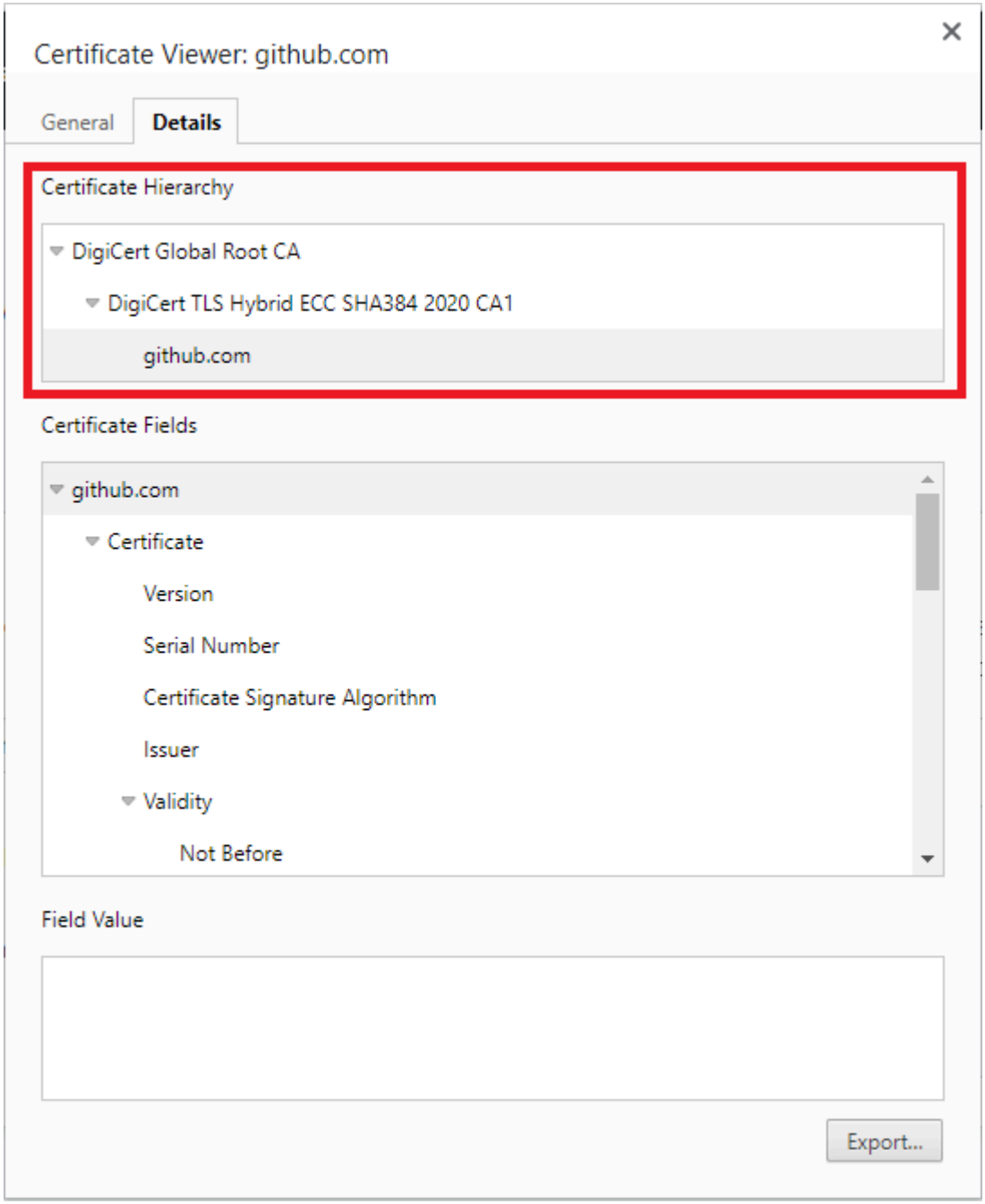
An SSL certificate is part of an SSL certificate chain. **What is a certificate chain?**

A certificate chain includes the following:

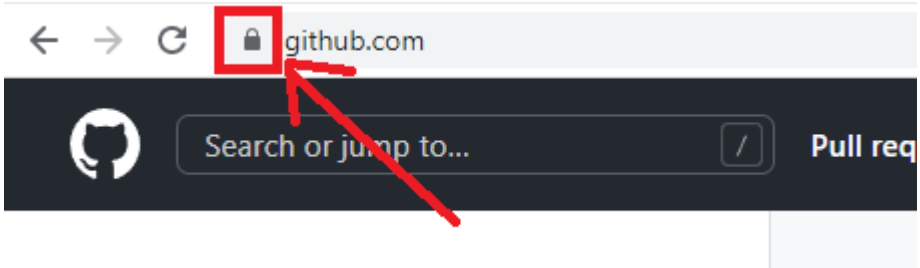
- root certificate (from a Certificate Authority);
- one or more intermediate certificates;
- the server certificate.

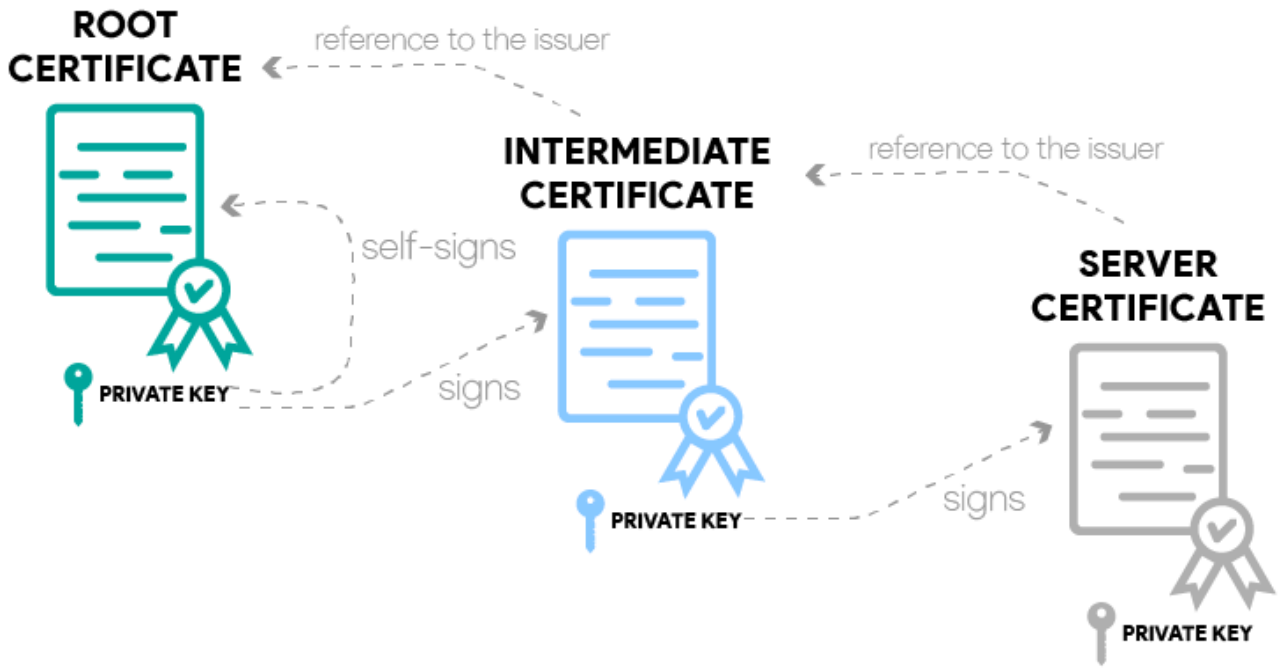
The server certificate is what makes your browser show a secure padlock icon when you visit a website. It means the server has a **valid** SSL/TLS certificate and all the connections with the website are encrypted. A valid SSL/TLS certificate is a certificate trusted by your browser. What makes it trustable?

As we've mentioned previously, SSL/TLS certificates are issued by Certificate Authorities. However, these authorities don't issue certificates directly to websites. They use intermediates that will issue the server certificate (**Certificate Authority** > **Intermediate certificate** > **server certificate**). The following screenshot shows



Your browser checks this certificate chain until it finds the root certificate. If that certificate is in the browser's root store, then it considers the certificate to be valid. In this case, the DigiCert Global Root CA is in the browser's root store. So, it will display the "secure" icon on the browser bar.





In summary:

- **root certificate**: it's a self-signed certificate issued by a Certificate Authority. The private key of this certificate is used to sign the next certificate in the hierarchy of certificates. Root certificates are loaded in the trust stores of browsers and operating systems.

- **intermediate certificate**: it's signed by the private key of the root certificate. The private key of the intermediate certificate is the one that signs the server certificate. There can be more than one intermediate certificate.

- **server certificate**: this certificate is issued to a specific domain name on a server. It's signed by the intermediate certificate private key. If it is valid (trustable certificate chain), the browser displays a secure padlock badge on the search bar next to the website domain.

With the ESP32, to check the validity of a server, you can load any of those certificates: root, intermediate, or server certificate.

Certificates Expiration Date

short-term validity.

So, if you want to use it in your ESP32 projects, you'll need to update your code quite frequently. If you want your code to run for years without worrying, you can use the website's root certificate, which usually has a validity of five to ten years or more.

Getting a Server's Certificate

There are different ways to get the server's certificate. One of the easiest ways is to download the certificate directly from your browser. You can also use OpenSSL and get all the certificate information you need using the command line (we won't cover this method in this tutorial).

In this section, you'll learn how to get the server's certificate. We'll generally use the root certificate, but you can use any of the other certificates on the certificate chain—you just need to be aware of the certificate expiry date.

Getting a Server's Certificate using Google Chrome

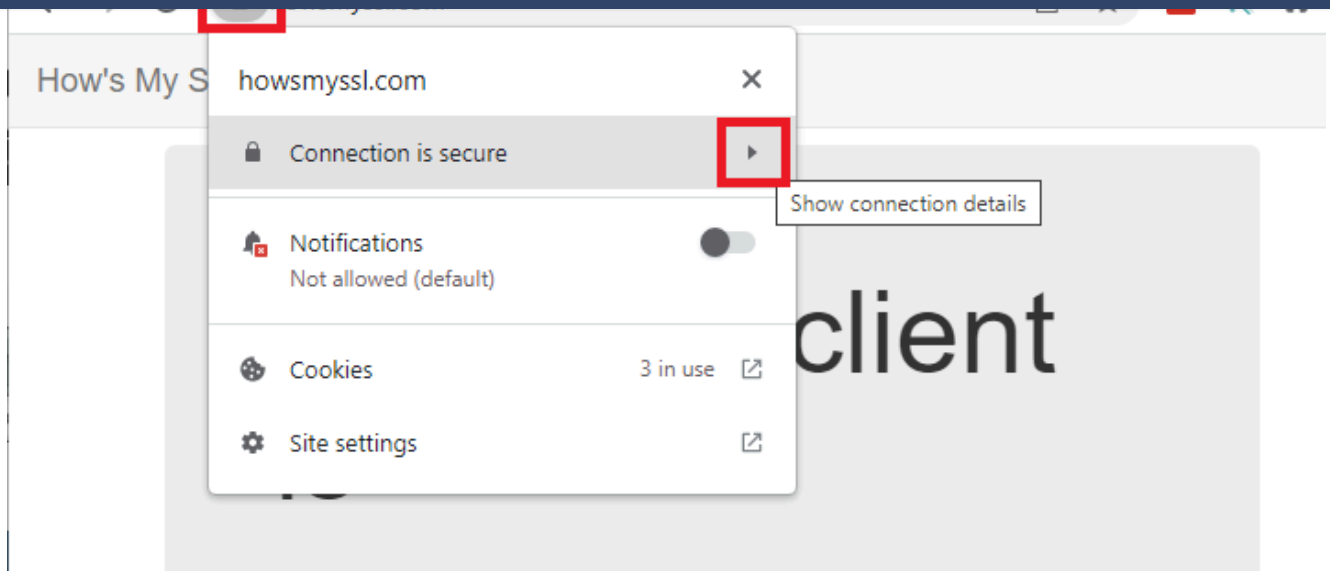
In this section, we'll show you how to get the certificate for a server using Google Chrome (that's the web browser we use more often). Instructions for other web browsers should be similar.

One of the examples we'll use later is to make an HTTPS request to the howmyssl.com website. So, for demonstration purposes, we'll show you how to get its root certificate. It is similar for other websites.

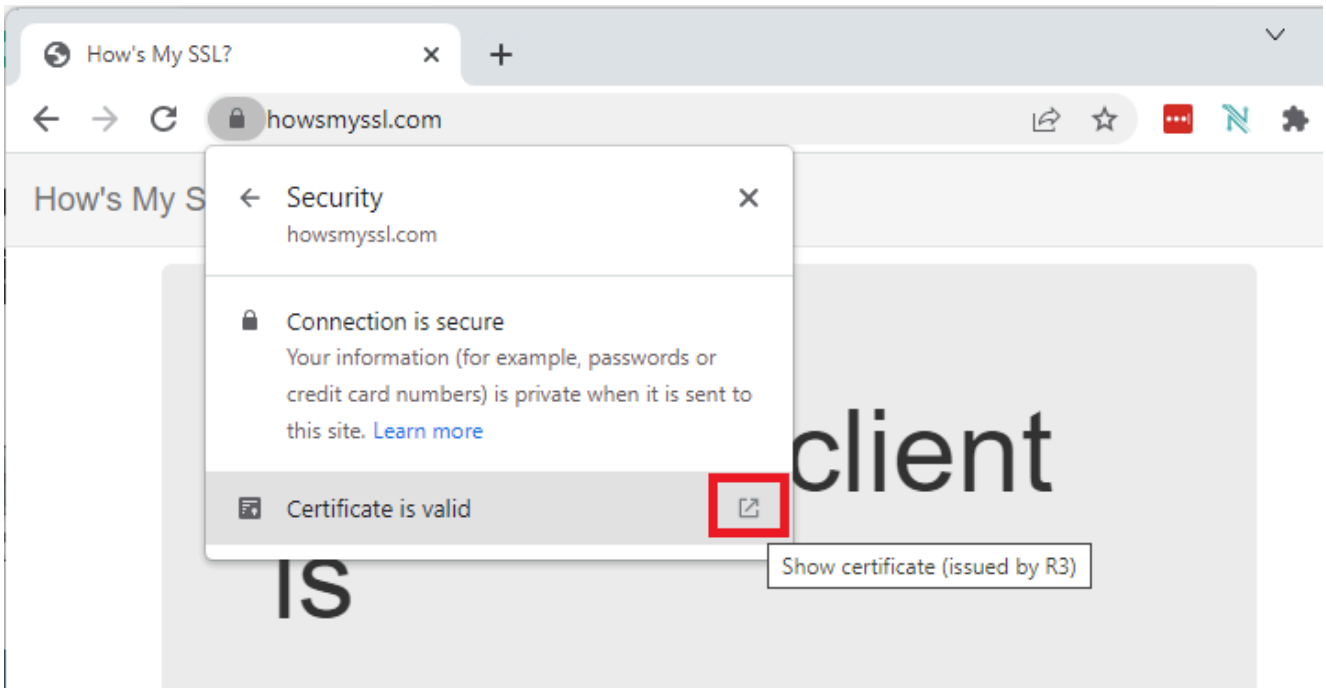
How to Get Websites's Certificate using Google Chrome?

1. Go to the website that you want to get the certificate for.

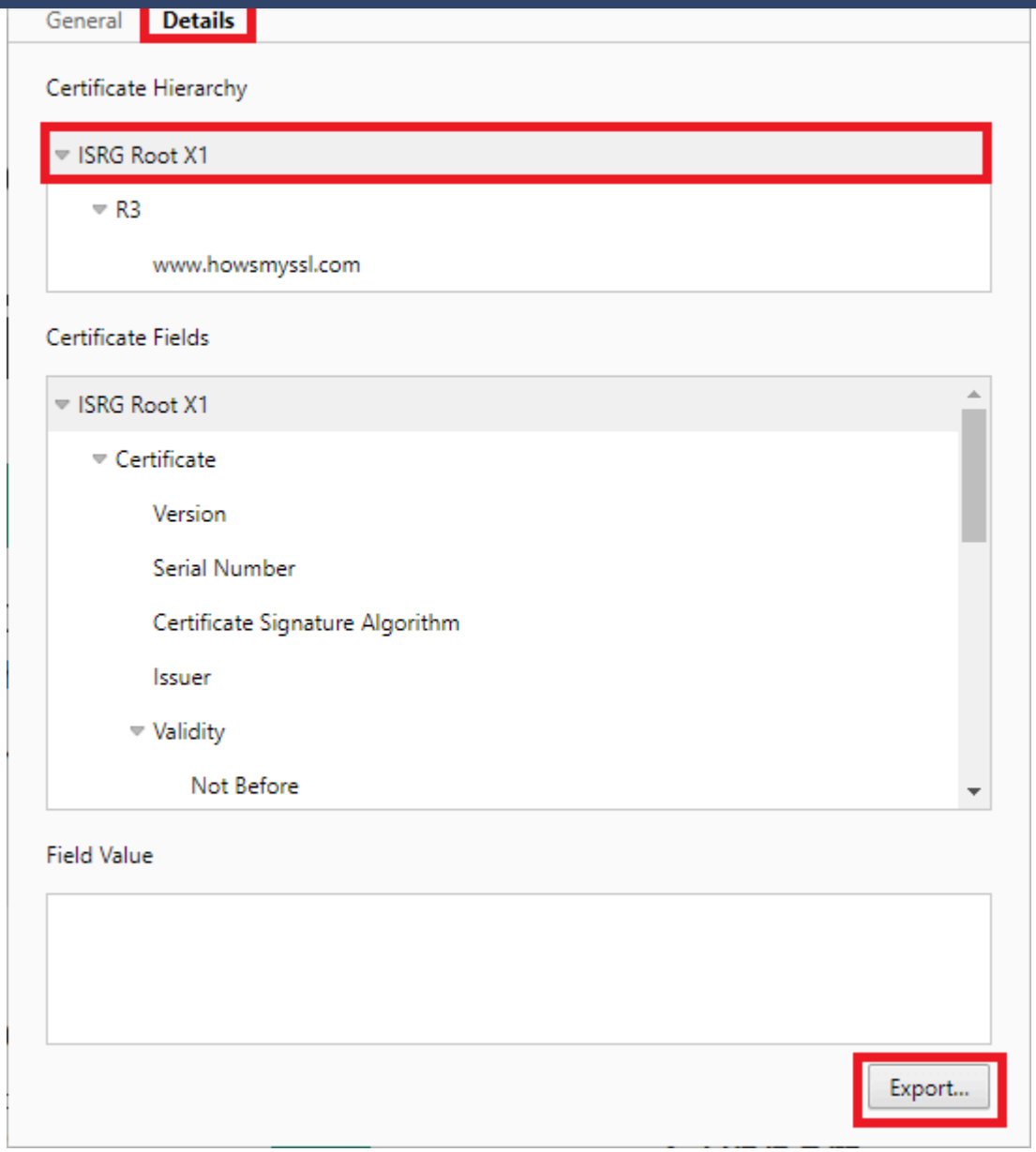
2. Click on the padlock icon and then click on **Show connection details**.



3. Then, click on **Show certificate**.

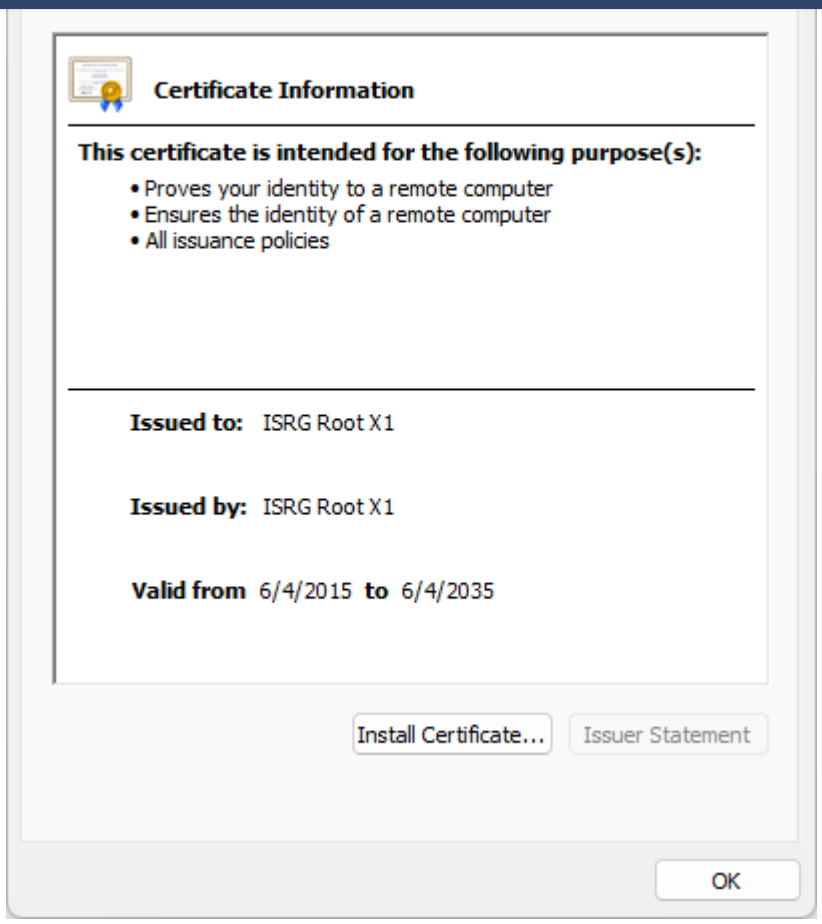


4. A new window will open the all the information about the website's certificate. Click on the Details tab, make sure you select the root certificate (that's what we're looking for in this example), then click on **Export…**



5. Select a place on your computer to save the certificate. Save it on the default format: `Base64-encoded ASCII, single certificate (*.pem, .crt)`. And that's it.

You can double-click on the certificate to check it's details, including the expiration date.



Open the certificate using Notepad or other similar software. You should get something similar as shown below.

ISRG Root X1.crt

```
-----BEGIN CERTIFICATE-----
MIIFazCCA1OgAwIBAgIRAIIQz7DSQONZRGPgu2OCiwAwDQYJKoZIhvcNAQELBQAw
TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh
cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMTUwNjA0MTEwNDM4
WhcNMzUwNjA0MTEwNDM4WjBPMQswCQYDVQQGEwJVUzEpMCcGA1UEChMgSW50ZXJu
ZXQgU2VjdXJpdHkgUmVzZWFyY2ggR3JvdXAxFTATBgNVBAMTDE1TUkcgUm9vdCBY
MTCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAK3oJHP0FDfzm54rVygc
h77ct984kIxuPOZXoHj3dcKi/vVqbvYATyjb3miGbESTtrFj/RQSa78f0uoxmyF+
0TM8ukj13Xnfs7j/EvEhmkvBioZxaUpmZmyPfjxwv60pIgbz5MDmgK7iS4+3mX6U
A5/TR5d8mUgjU+g4rk8Kb4Mu0UlXjIB0ttov0DiNewNwIRt18jA8+o+u3dpjq+sW
T8KOEUt+zwvo/7V3LvSye0rgTBIlDHCNAymg4VMk7BPZ7hm/ELNKjD+Jo2FR3qyH
B5T0Y3HsLuJvW5iB4YlcNHlsdu87kGJ55tukmi8mxdAQ4Q7e2RCOFvu396j3x+UC
B5iPNgiV5+I3lg02dZ77DnKxHZu8A/lJBdiB3QW0KtZB6awBdpUKD9jf1b0SHzUv
KBds0pjBqAlkd25HN7rOrFleaJ1/ctaJxQZBKT5ZPt0m9STJEadao0xAH0ahmbWn
OlFuhjuefXKnEgV4We0+UXgVCwOPjdAvBbI+e0ocS3MFEvzG6uBQE3xDk3SzynTn
jh8BCNAw1FtxNrQHusEwMFxIt4I7mKZ9YIqioymCzLq9gwQbooMDQaHWBfEbwrbw
qHyGO0aoSCqI3Haadr8faqU9GY/rOPNk3sgrDQoo//fb4hVC1CLQJ13hef4Y53CI
rU7m2Ys6xt0nUW7/vGT1M0NPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV
HRMBAf8EBTADAQH/MB0GA1UdDgQWBBR5tFnme7bl5AFzgAiIyBpY9umbbjANBgkq
hkiG9w0BAQsFAAOCAgEAVR9YqbyyqFDQDLHYGmkgJykIrGF1XIpu+ILlaS/V9lZL
ubhzEFnTIZd+50xx+7LSYK05qAvqFyFWhfFQDlnrzuBZ6brJFe+GnY+EgPbk6ZGQ
3BebYhtF8GaV0nxvwuo77x/Py9auJ/GpsMiu/X1+mvoiBOv/2X/qkSsisRcOj/KK
NFtY2PwByVS5uCbMiogziUwthDyC3+6WVwW6LLv3xLfHTjuCvjHIInNzktHCgKQ5
ORAzI4JMPJ+GslWYHb4phowim57iaztXOoJwTdwJx4nLCgdNbOhdjsnvzqvHu7Ur
TkXWStAmzOVyyghqpZXjFaH3pO3JLF+l+/+sKAIuvtd7u+Nxe5AW0wdeRlN8NwdC
jNPElpzVmbUq4JUagEiuTDkHzsxHpFKVK7q4+63SM1N95R1NbdWhscdCb+ZAJzVc
oyi3B43njTOQ5yOf+1CceWxG1bQVs5ZufpsMljq4Ui0/1lvh+wjChP4kqKOJ2qxq
4RgqsahDYVvTH9w7jXbyLeiNdd8XM2w9U/t7y0Ff/9yi0GE44Za4rF2LN9d11TPA
mRGunUHBcnWEvgJBQl9nJEiU0Zsnvgc/ubhPgXRR4Xq37Z0j4r7g1SgEEzwxA57d
emyPxgcYxn/eR44/KJ4EBs+lVDR3veyJm+kXQ99b21/+jh5Xos1AnX5iItreGCc=
-----END CERTIFICATE-----
```

G:\.shortcut-targets-by-id\1CULU9CWUp8PccwHOJdnlQCN CRLF UTF-8 YAML GitHub Git (0)

We need to convert this to Arduino multi-line string, so that we can use it in our sketch. Basically, you need to add a `“` at the beginning of each line and a `\n” \` at the end of each line, except the last line that you should add `\n”`. So, you’ll get something as shown below:

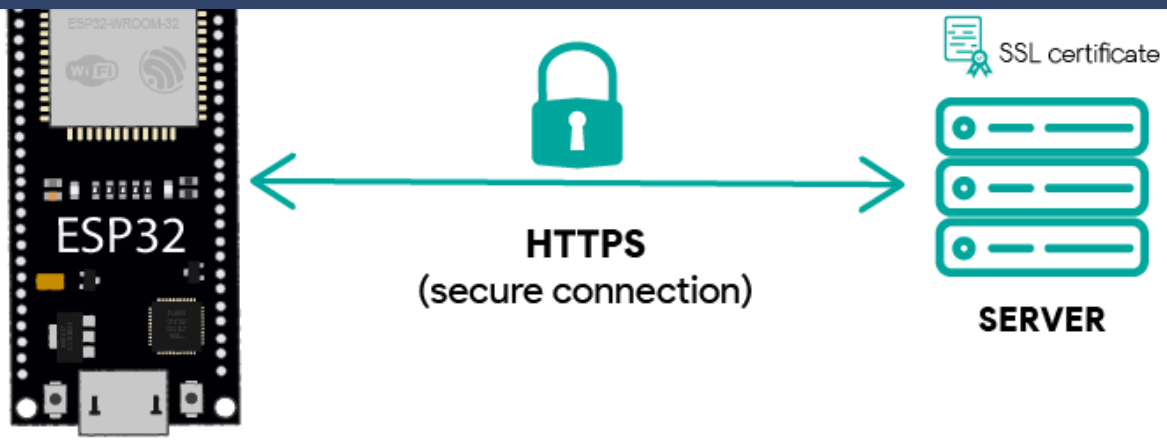
ISRG Root X1.crt

```
"-----BEGIN CERTIFICATE-----\n" \
"MIIFazCCA1OgAwIBAgIRAIIQz7DSQONZRGPgu2OCiwAwDQYJKoZIhvcNAQELBQAw\n" \
"TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh\n" \
"cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMTUwNjA0MTEwNDM4\n" \
"WhcNMzUwNjA0MTEwNDM4WjBPMQswCQYDVQQGEwJVUzEpMCcGA1UEChMgSW50ZXJu\n" \
"ZXQgU2VjdXJpdHkgUmVzZWFyY2ggR3JvdXAxFTATBgNVBAMTDElTUkcgUm9vdCBY\n" \
"MTCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAK3oJHP0FDfzm54rVygc\n" \
"h77ct984kIxuPOZXoHj3dcKi/vVqbvYATyjb3miGbESTtrFj/RQSa78f0uoxmyF+\n" \
"0TM8ukj13Xnfs7j/EvEhmkvBioZxaUpmZmyPfjxwv60pIgbz5MDmgK7iS4+3mX6U\n" \
"A5/TR5d8mUgjU+g4rk8Kb4Mu0UlXjIB0ttov0DiNewNwIRt18jA8+o+u3dpjq+sW\n" \
"T8KOEUt+zwvo/7V3LvSye0rgTBIlDHCNAymg4VMk7BPZ7hm/ELNKjD+Jo2FR3qyH\n" \
"B5T0Y3HsLuJvW5iB4YlcNHlsdu87kGJ55tukmi8mxdAQ4Q7e2RCOFvu396j3x+UC\n" \
"B5iPNgiV5+I3lg02dZ77DnKxHZu8A/lJBdiB3QW0KtZB6awBdpUKD9jf1b0SHzUv\n" \
"KBds0pjBqAlkd25HN7rOrFleaJ1/ctaJxQZBKT5ZPt0m9STJEadao0xAH0ahmbWn\n" \
"OlFuhjuefXKnEgV4We0+UXgVCwOPjdAvBbI+e0ocS3MFEvzG6uBQE3xDk3SzynTn\n" \
"jh8BCNAw1FtxNrQHusEwMFxIt4I7mKZ9YIqioymCzLq9gwQbooMDQaHWBfEbwrbw\n" \
"qHyGO0aoSCqI3Haadr8faqU9GY/rOPNk3sgrDQoo//fb4hVC1CLQJ13hef4Y53CI\n" \
"rU7m2Ys6xt0nUW7/vGT1M0NPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV\n" \
"HRMBAf8EBTADAQH/MB0GA1UdDgQWBBR5tFnme7bl5AFzgAiIyBpY9umbbjANBgkq\n" \
"hkiG9w0BAQsFAAOCAgEAVR9YqbyyqFDQDLHYGmkgJykIrGF1XIpu+ILlaS/V9lZL\n" \
"ubhzEFnTIZd+50xx+7LSYK05qAvqFyFWhfFQDlnrzuBZ6brJFe+GnY+EgPbk6ZGQ\n" \
"3BebYhtF8GaV0nxvwuo77x/Py9auJ/GpsMiu/X1+mvoiBOv/2X/qkSsisRcOj/KK\n" \
"NFtY2PwByVS5uCbMiogziUwthDyC3+6WVwW6LLv3xLfHTjuCvjHIInNzktHCgKQ5\n" \
"ORAzI4JMPJ+GslWYHb4phowim57iaztXOoJwTdwJx4nLCgdNbOhdjsnvzqvHu7Ur\n" \
"TkXWStAmzOVyyghqpZXjFaH3pO3JLF+l+/+sKAIuvtd7u+Nxe5AW0wdeRlN8NwdC\n" \
"jNPElpzVmbUq4JUagEiuTDkHzsxHpFKVK7q4+63SM1N95R1NbdWhscdCb+ZAJzVc\n" \
"oyi3B43njTOQ5yOf+1CceWxG1bQVs5ZufpsMljq4Ui0/1lvh+wjChP4kqKOJ2qxq\n" \
"4RgqsahDYVvTH9w7jXbyLeiNdd8XM2w9U/t7y0Ff/9yi0GE44Za4rF2LN9d11TPA\n" \
"mRGunUHBcnWEvgJBQl9nJEiU0Zsnvgc/ubhPgXRR4Xq37Z0j4r7g1SgEEzwxA57d\n" \
"emyPxgcYxn/eR44/KJ4EBs+lVDR3veyJm+kXQ99b21/+jh5Xos1AnX5iItreGCc=\n" \
"-----END CERTIFICATE-----\n"
```

G:\.shortcut-targets-by-id\1CULU9CWUp8PccwHOJdnIQCMzzh3k\ CRLF UTF-8 YAML GitHub Git (0)

HTTPS Requests with the ESP32

Now that you know all the major important aspects of certificates and how to get a server's certificate, let's finally take a look at how to make HTTPS requests on the ESP32 using the Arduino core. We'll cover different methods using two different libraries: `WiFiClientSecure` and `HTTPClient`.



ESP32 HTTPS Requests using WiFiClientSecure Library

You can find a simple example showing how to make HTTPS requests with the WiFiClientSecure library on your Arduino IDE.

ESP32 HTTPS Requests with Certificate

Make sure you have an ESP32 board selected in **Tools** > **Board**. Then, go to **File** > **Examples** > **WiFiClientSecure** > **WiFiClientSecure**.

You can modify the following code with the certificate we got from the previous steps, which is valid until 2035.

```
/*
  Complete project details: https://RandomNerdTutorials.com/esp

  Wifi secure connection example for ESP32 - Running on TLS 1.2
  Suporting the following chipersuites:
  "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384","TLS_ECDHE_RSA_WITH
  2017 - Evandro Copercini - Apache 2.0 License.
*/

#include <WiFiClientSecure.h>

const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

```
// www.howsmyssl.com root certificate authority, to verify the
// change it to your server root CA
// SHA1 fingerprint is broken now!

const char* test_root_ca= \
  "-----BEGIN CERTIFICATE-----\n" \
  "MIIFazCCA1OgAwIBAgIRAIIQz7DSQONZRGPgu2OCiwAwDQYJKoZIhvcNAQEL
  "TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJl
  "cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMTUwNjA0MTEw
  "WhcNMzUwNjA0MTEwNDM4WjBPMQswCQYDVQQGEwJVUzEpMCcGA1UEChMgSW50
  "ZXQgU2VjdXJpdHkgUmVzZWFyY2ggR3JvdXAxFTATBgNVBAMTDElTUkcgUm9v
```

View raw code

This example establishes a secure connection with the `www.howsmyssl.com` website and checks its certificate to ensure we're connected to the server that we expect.

If you're used to making HTTP requests with the ESP32 using the `WiFiClient` library, this example is not much different.

How does the Code Work?

You need to include the `WiFiClientSecure` library.

```
#include <WiFiClientSecure.h>
```

Insert your network credentials in the following lines.

```
const char* ssid     = "REPLACE_WITH_YOUR_SSID";     // your netw
const char* password = "REPLACE_WITH_YOUR_PASSWORD"; // your netw
```

```
const char*  server = "www.howsmyssl.com";  // Server URL
```

Then, you need to insert the server certificate. We're using the root certificate.

```
const char* test_root_ca= \
     "-----BEGIN CERTIFICATE-----\n" \
     "MIIFazCCA1OgAwIBAgIRAIIQz7DSQONZRGPgu2OCiwAwDQYJKoZIhvcNAQE
     "TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJ
     "cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMTUwNjA0MTE
     "WhcNMzUwNjA0MTEwNDM4WjBPMQswCQYDVQQGEwJVUzEpMCcGA1UEChMgSW5
     "ZXQgU2VjdXJpdHkgUmVzZWFyY2ggR3JvdXAxFTATBgNVBAMTDElTUkcgUm9
     "MTCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAK3oJHP0FDfzm54
     "h77ct984kIxuPOZXoHj3dcKi/vVqbvYATyjb3miGbESTtrFj/RQSa78f0uo
     "0TM8ukj13Xnfs7j/EvEhmkvBioZxaUpmZmyPfjxwv60pIgbz5MDmgK7iS4+
     "A5/TR5d8mUgjU+g4rk8Kb4Mu0UlXjIB0ttov0DiNewNwIRt18jA8+o+u3dp
     "T8KOEUt+zwvo/7V3LvSye0rgTBIlDHCNAymg4VMk7BPZ7hm/ELNKjD+Jo2F
     "B5T0Y3HsLuJvW5iB4YlcNHlsdu87kGJ55tukmi8mxdAQ4Q7e2RCOFvu396j
     "B5iPNgiV5+I3lg02dZ77DnKxHZu8A/lJBdiB3QW0KtZB6awBdpUKD9jf1b0
     "KBds0pjBqAlkd25HN7rOrFleaJ1/ctaJxQZBKT5ZPt0m9STJEadao0xAH0a
     "OlFuhjuefXKnEgV4We0+UXgVCwOPjdAvBbI+e0ocS3MFEvzG6uBQE3xDk3S
     "jh8BCNAw1FtxNrQHusEwMFxIt4I7mKZ9YIqioymCzLq9gwQbooMDQaHWBfE
     "qHyGO0aoSCqI3Haadr8faqU9GY/rOPNk3sgrDQoo//fb4hVC1CLQJ13hef4
     "rU7m2Ys6xt0nUW7/vGT1M0NPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjA
     "HRMBAf8EBTADAQH/MB0GA1UdDgQWBBR5tFnme7bl5AFzgAiIyBpY9umbbjA
     "hkiG9w0BAQsFAAOCAgEAVR9YqbyyqFDQDLHYGmkgJykIrGF1XIpu+ILlaS/
     "ubhzEFnTIZd+50xx+7LSYK05qAvqFyFWhfFQDlnrzuBZ6brJFe+GnY+EgPb
     "3BebYhtF8GaV0nxvwuo77x/Py9auJ/GpsMiu/X1+mvoiBOv/2X/qkSsisRc
     "NFtY2PwByVS5uCbMiogziUwthDyC3+6WVwW6LLv3xLfHTjuCvjHIInNzktH
     "ORAzI4JMPJ+GslWYHb4phowim57iaztXOoJwTdwJx4nLCgdNbOhdjsnvzqv
     "TkXWStAmzOVyyghqpZXjFaH3pO3JLF+l+/+sKAIuvtd7u+Nxe5AW0wdeRlN
     "jNPElpzVmbUq4JUagEiuTDkHzsxHpFKVK7q4+63SM1N95R1NbdWhscdCb+Z
     "oyi3B43njTOQ5yOf+1CceWxG1bQVs5ZufpsMljq4Ui0/1lvh+wjChP4kqKO
     "4RgqsahDYVvTH9w7jXbyLeiNdd8XM2w9U/t7y0Ff/9yi0GE44Za4rF2LN9d
     "mRGunUHBcnWEvgJBQl9nJEiU0Zsnvgc/ubhPgXRR4Xq37Z0j4r7g1SgEEzw
```

Create a new client called `client` using `WiFiClient` secure.

```
WiFiClientSecure client;
```

In the `setup()`, initialize the Serial Monitor and connect to your network.

```
//Initialize serial and wait for port to open:
Serial.begin(115200);
delay(100);

Serial.print("Attempting to connect to SSID: ");
Serial.println(ssid);
WiFi.begin(ssid, password);

// attempt to connect to Wifi network:
while (WiFi.status() != WL_CONNECTED) {
  Serial.print(".");
  // wait 1 second for re-trying
  delay(1000);
}

Serial.print("Connected to ");
Serial.println(ssid);
```

The following line set the client certificate using the `setCACert()` method on the `client`.

```
client.setCACert(test_root_ca);
```

Then, the client connects to the server. For HTTPS, you need to use port `443`.

```
    Serial.println("Connection failed!");
```

If the connection is successful, we can make the HTTP request. In this case, we're making a GET request. Note that you need to use the `https://` before the URL you'll make a request to.

```
  else {
    Serial.println("Connected to server!");
    // Make a HTTP request:
    client.println("GET https://www.howsmyssl.com/a/check HTTP/1.
    client.println("Host: www.howsmyssl.com");
    client.println("Connection: close");
    client.println();
```

Finally, we get and print the response from the server:

```
while (client.connected()) {
      String line = client.readStringUntil('\n');
      if (line == "\r") {
        Serial.println("headers received");
        break;
      }
    }
    // if there are incoming bytes available
    // from the server, read them and print them:
    while (client.available()) {
      char c = client.read();
      Serial.write(c);
    }
```

In the end, we close the connection with the client.

In this example, we make the request once in the `setup()`. The `loop()` is empty, but you can add any other tasks that you need in your project. Or, depending on the application, you can make the request on the `loop()`.

```
void loop() {
  // do nothing
}
```

In summary, to make HTTPS requests:

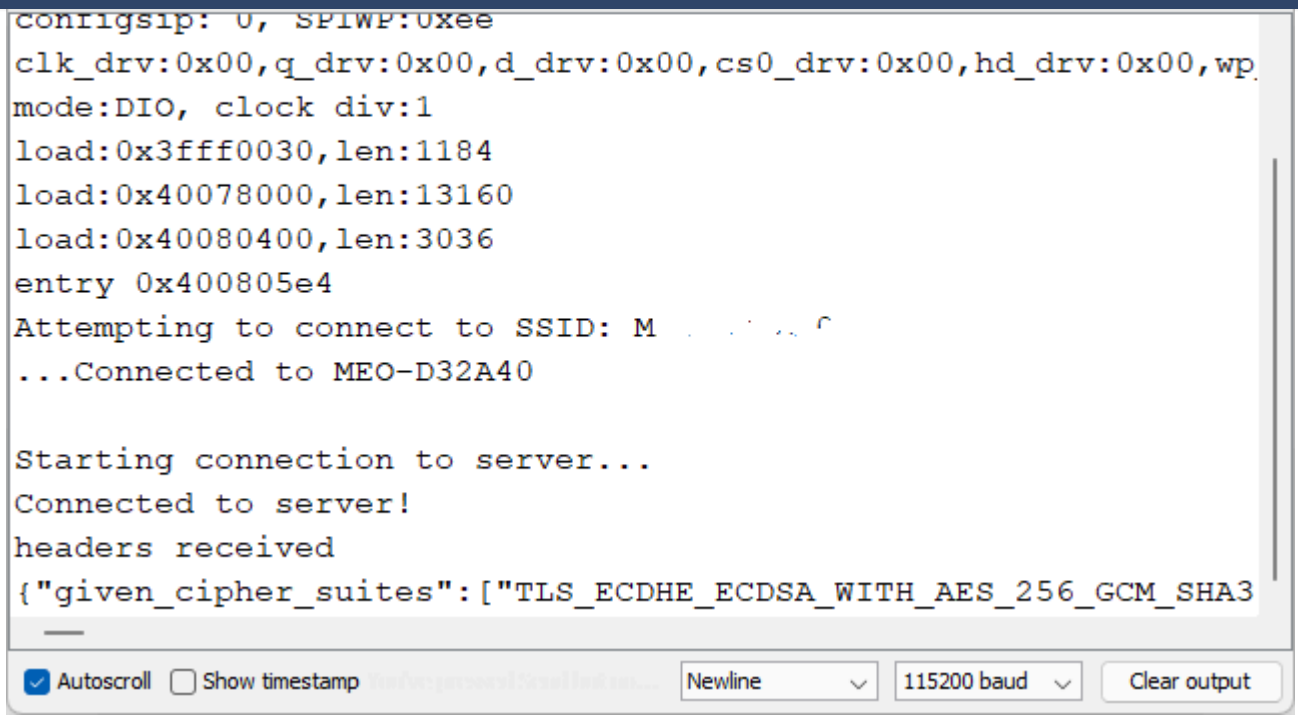
- Include the `WiFiClientSecure` library;
- Create a `WiFiClientSecure` client;
- Use port `443`;
- Use the `setCACert()` function to set the client certificate.
- Use `https` on the URL when making the HTTPS request.

Demonstration

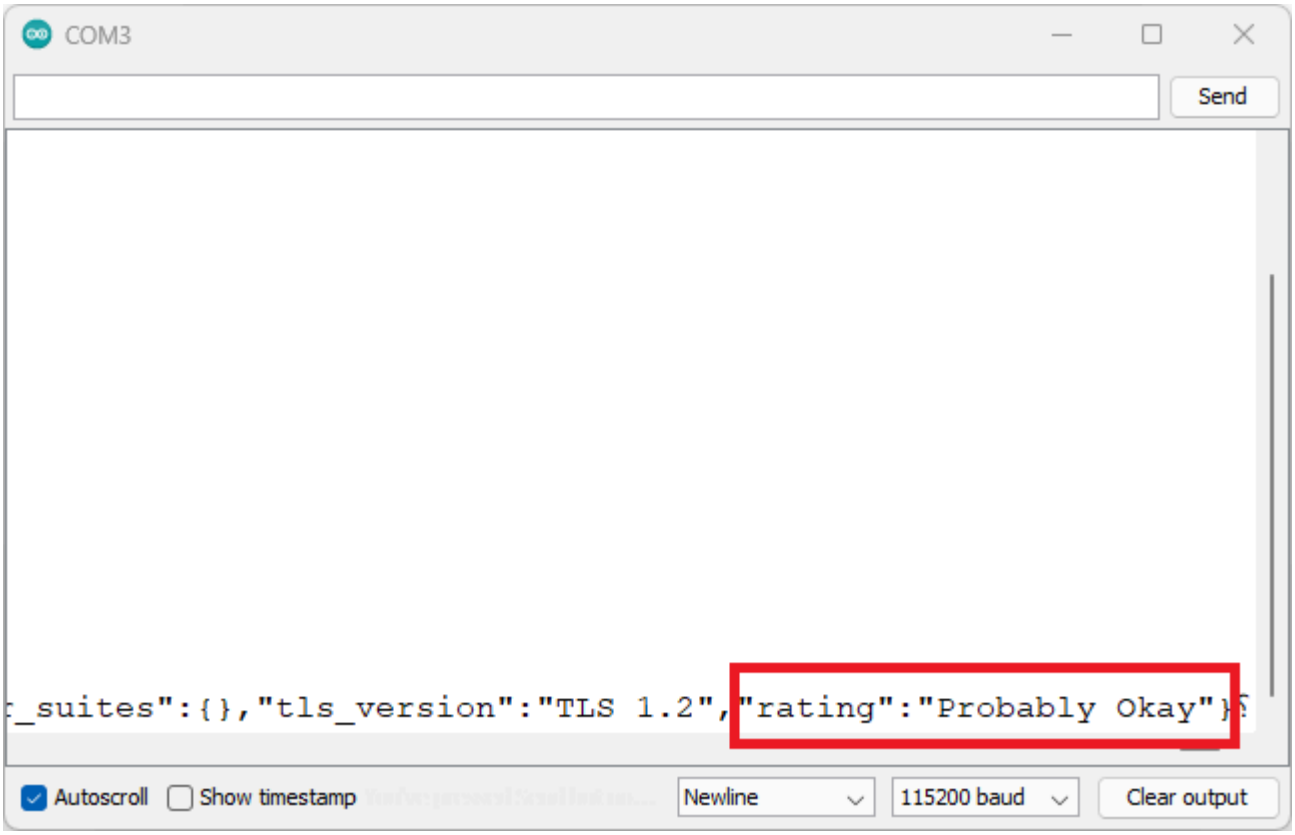
Upload the code to your board.

Open the Serial Monitor at a baud rate of 115200 and press the onboard RST button.

You should get something as shown in the following screenshot.



If you scroll to the right, you'll get the result of how secure the connection is. You should get a "Probably Okay".



ESP32 HTTPS Requests without Certificate

```
client.setCACert(test_root_ca);
```

And add the following line before connecting with the client:

```
client.setInsecure();
```

The complete example can be found below.

```
/*
  Complete project details: https://RandomNerdTutorials.com/esp

  Based on the WiFiClientSecure example HTTPS Requests without
  Wifi secure connection example for ESP32
  Running on TLS 1.2 using mbedTLS
  Suporting the following chipersuites:
  "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384","TLS_ECDHE_RSA_WITH_
  2017 - Evandro Copercini - Apache 2.0 License.
*/

#include <WiFiClientSecure.h>

const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

const char*  server = "www.howsmyssl.com";  // Server URL

WiFiClientSecure client;

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(115200);
  delay(100);
```

View raw code

With this example, your connection is still encrypted, but you won't be sure if you're talking to the right server. This scenario is useful for testing purposes.

ESP32 HTTPS Requests with Certificate Bundle

Instead of just using one certificate, you can use a certificate bundle: a collection of trusted certificates that you can load into your board. Then, you don't have to worry about getting the certificate for a specific server.

The `WiFiClient` library provides some information about how to use a certificate bundle on the following link:

- https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFiClientSecure/README.md#using-a-bundle-of-root-certificate-authority-certificates

I followed all the instructions provided, and got the following issue:

```
[  1799][E][ssl_client.cpp:37] _handle_error():
[start_ssl_client():276]: (-12288) X509 - A fatal error
occurred, eg the chain is too long or the vrfy callback failed
```

If anyone knows how to fix this issue, please share in the comments below.

ESP32 HTTP Requests using HTTPClient Library

The HTTPClient library provides a simple example showing how to make HTTPS requests with the ESP32. You can find the example in your Arduino IDE. First, make sure you have an ESP32 board selected in **Tools** > **Board**. Then, go to **File** > **Examples** > **HTTPClient** > **BasicHttpsClient**. We created new sketches based on that example. See the code below.

ESP32 HTTPS Requests with Certificate

use the root certificate we've gotten in previous steps.

```
/*
  Complete project details: https://RandomNerdTutorials.com/esp
  Based on the BasicHTTPSClient.ino example found at Examples >
*/

#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <HTTPClient.h>

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// www.howsmyssl.com root certificate authority, to verify the
// change it to your server root CA
const char* rootCACertificate = \
     "-----BEGIN CERTIFICATE-----\n" \
     "MIIFazCCA1OgAwIBAgIRAIIQz7DSQONZRGPgu2OCiwAwDQYJKoZIhvcNA
     "TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5I
     "cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMTUwNjA0M
     "WhcNMzUwNjA0MTEwNDM4WjBPMQswCQYDVQQGEwJVUzEpMCcGA1UEChMgSI
     "ZXQgU2VjdXJpdHkgUmVzZWFyY2ggR3JvdXAxFTATBgNVBAMTDElTUkcgU
     "MTCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAK3oJHP0FDfzm
     "h77ct984kIxuPOZXoHj3dcKi/vVqbvYATyjb3miGbESTtrFj/RQSa78f0
     "0TM8ukj13Xnfs7j/EvEhmkvBioZxaUpmZmyPfjxwv60pIgbz5MDmgK7iS
     "A5/TR5d8mUgjU+g4rk8Kb4Mu0UlXjIB0ttov0DiNewNwIRt18jA8+o+u3
```

View raw code

How does the Code Work?

Start by including the required libraries: `WiFi.h`, `WiFiClientSecure.h`, and `HTTPClient.h`.

```
#include <WiFiClientSecure.h>
#include <HTTPClient.h>
```

Insert your network credentials in the following lines:

```
// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Next, you need to add the server certificate. We're using the root certificate for howsmyssl.com (see previous steps).

```
const char* rootCACertificate = \
     "-----BEGIN CERTIFICATE-----\n" \
     "MIIFazCCA1OgAwIBAgIRAIIQz7DSQONZRGPgu2OCiwAwDQYJKoZIhvcNAQE
     "TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJ
     "cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMTUwNjA0MTE
     "WhcNMzUwNjA0MTEwNDM4WjBPMQswCQYDVQQGEwJVUzEpMCcGA1UEChMgSW5
     "ZXQgU2VjdXJpdHkgUmVzZWFyY2ggR3JvdXAxFTATBgNVBAMTDElTUkcgUm9
     "MTCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAK3oJHP0FDfzm54
     "h77ct984kIxuPOZXoHj3dcKi/vVqbvYATyjb3miGbESTtrFj/RQSa78f0uo
     "0TM8ukj13Xnfs7j/EvEhmkvBioZxaUpmZmyPfjxwv60pIgbz5MDmgK7iS4+
     "A5/TR5d8mUgjU+g4rk8Kb4Mu0UlXjIB0ttov0DiNewNwIRt18jA8+o+u3dp
     "T8KOEUt+zwvo/7V3LvSye0rgTBIlDHCNAymg4VMk7BPZ7hm/ELNKjD+Jo2F
     "B5T0Y3HsLuJvW5iB4YlcNHlsdu87kGJ55tukmi8mxdAQ4Q7e2RCOFvu396j
     "B5iPNgiV5+I3lg02dZ77DnKxHZu8A/lJBdiB3QW0KtZB6awBdpUKD9jf1b0
     "KBds0pjBqAlkd25HN7rOrFleaJ1/ctaJxQZBKT5ZPt0m9STJEadao0xAH0a
     "OlFuhjuefXKnEgV4We0+UXgVCwOPjdAvBbI+e0ocS3MFEvzG6uBQE3xDk3S
     "jh8BCNAw1FtxNrQHusEwMFxIt4I7mKZ9YIqioymCzLq9gwQbooMDQaHWBfE
     "qHyGO0aoSCqI3Haadr8faqU9GY/rOPNk3sgrDQoo//fb4hVC1CLQJ13hef4
     "rU7m2Ys6xt0nUW7/vGT1M0NPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjA
     "HRMBAf8EBTADAQH/MB0GA1UdDgQWBBR5tFnme7bl5AFzgAiIyBpY9umbbjA
     "hkiG9w0BAQsFAAOCAgEAVR9YqbyyqFDQDLHYGmkgJykIrGF1XIpu+ILlaS/
```

```
      "NFtY2PwByVS5uCbMiogziUwthDyC3+6WVwW6LLv3xLfHTjuCvjHIInNzktH
      "ORAzI4JMPJ+GslWYHb4phowim57iaztXOoJwTdwJx4nLCgdNbOhdjsnvzqv
      "TkXWStAmzOVyyghqpZXjFaH3pO3JLF+l+/+sKAIuvtd7u+Nxe5AW0wdeRlN
      "jNPElpzVmbUq4JUagEiuTDkHzsxHpFKVK7q4+63SM1N95R1NbdWhscdCb+Z
      "oyi3B43njTOQ5yOf+1CceWxG1bQVs5ZufpsMljq4Ui0/1lvh+wjChP4kqKO
      "4RgqsahDYVvTH9w7jXbyLeiNdd8XM2w9U/t7y0Ff/9yi0GE44Za4rF2LN9d
      "mRGunUHBcnWEvgJBQl9nJEiU0Zsnvgc/ubhPgXRR4Xq37Z0j4r7g1SgEEzw
      "emyPxgcYxn/eR44/KJ4EBs+lVDR3veyJm+kXQ99b21/+jh5Xos1AnX5iItr
      "-----END CERTIFICATE-----\n";
```

In the setup () initialize the Serial Monitor and connect to Wi-Fi.

```
void setup() {
  Serial.begin(115200);
  Serial.println();
  // Initialize Wi-Fi
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
  Serial.println(WiFi.localIP());
}
```

In the `loop()`, create a pointer to `WiFiClientSecure` called `client`.

```
WiFiClientSecure *client = new WiFiClientSecure;
```

Set a secure client with the certificate using the `setCACert()` method:

Then, create an `HTTPClient` instance called `https`.

```
//create an HTTPClient instance
HTTPClient https;
```

Initialize the https client on the host specified using the `begin()` method. In this case, we're making a request on the following URL: `https://www.howsmyssl.com/a/check`.

```
if (https.begin(*client, "https://www.howsmyssl.com/a/check")) {
```

Get the server response code.

```
int httpCode = https.GET();
```

If the response code is a positive number, it means the connection was established successfully, so we can read the response payload using the `getString()` method on the `https` object. Then, we can print the payload in the Serial Monitor. In a practical application, you can do whatever task you need with the ESP32 depending on the received payload.

```
if (https.begin(client, "https://www.howsmyssl.com/a/check")) {
  Serial.print("[HTTPS] GET...\n");
  // start connection and send HTTP header
  int httpCode = https.GET();
  // httpCode will be negative on error
  if (httpCode > 0) {
    // HTTP header has been send and Server response header has b
    Serial.printf("[HTTPS] GET... code: %d\n", httpCode);
    // file found at server
```

```
        String payload = https.getString();
        Serial.println(payload);
      }
    }
```

If the response code is a negative number, it means we have an error. We'll print the error code.

```
else {
   Serial.printf("[HTTPS] GET... failed, error: %s\n", https.erro
}
```

Finally, close the HTTPS connection using the `end()` method:

```
https.end();
```

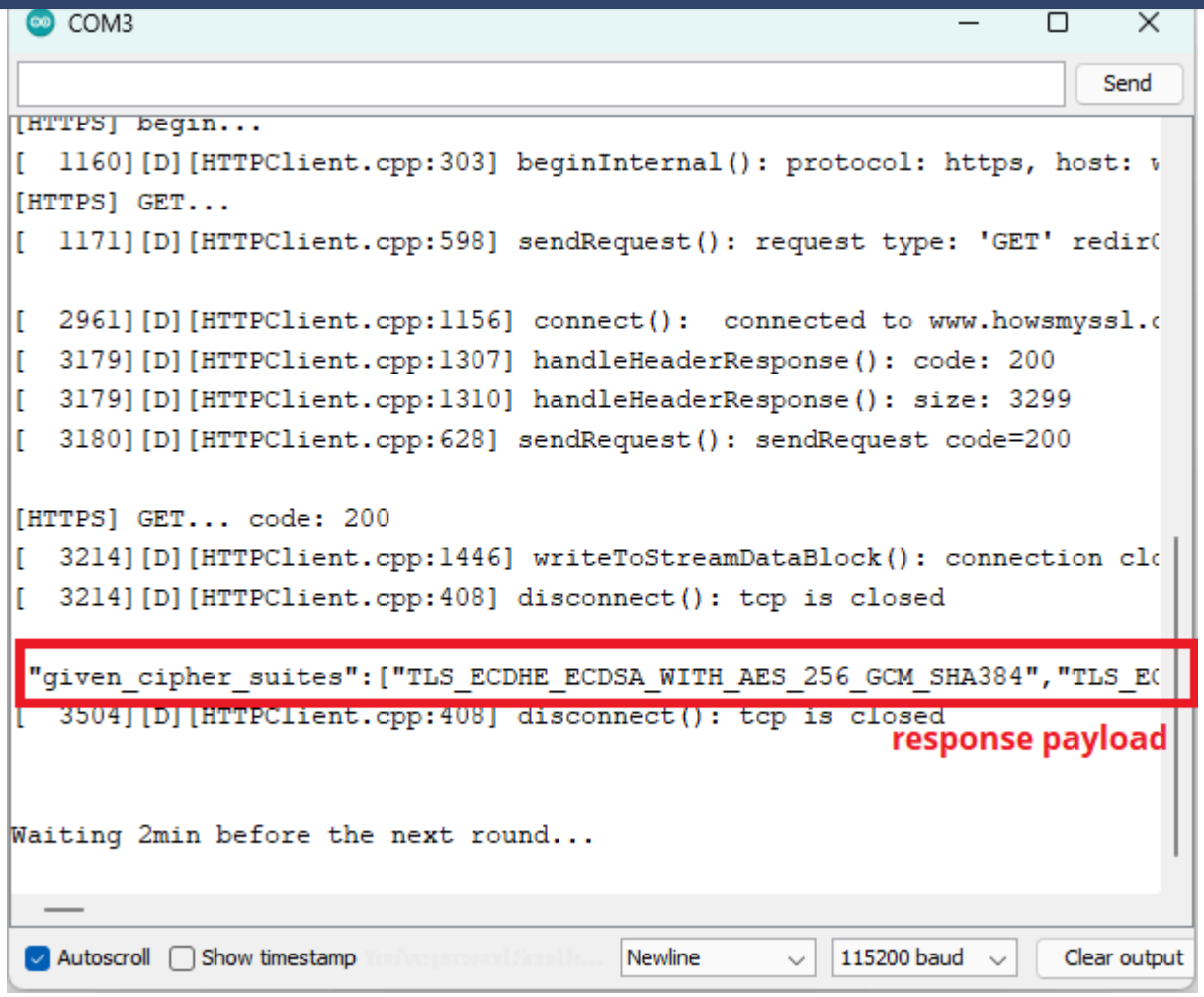
This specific example makes a request every two minutes. You can change it depending on your project requirements.

```
  Serial.println("Waiting 2min before the next round...");
  delay(120000);
```

Demonstration

You can change the debug level to get more information about what's going on in the process. Go to **Tools** > **Core Debug Level** > **Debug**. Then, you can upload the code to the ESP32.

After uploading the code, open the Serial Monitor at a baud rate of 115200. Press the on-board RST board to start running the newly uploaded code.



If you scroll to the right, you'll get the result of how secure the connection is. You should get a "Probably Okay".

ESP32 HTTPS Requests without Certificate

If you want to skip the SSL server certificate verification, but you still want to have encrypted communication, you can remove the following line:

```
client.setCACert(test_root_ca);
```

And add the following line before starting the HTTP client:

```
client.setInsecure();
```

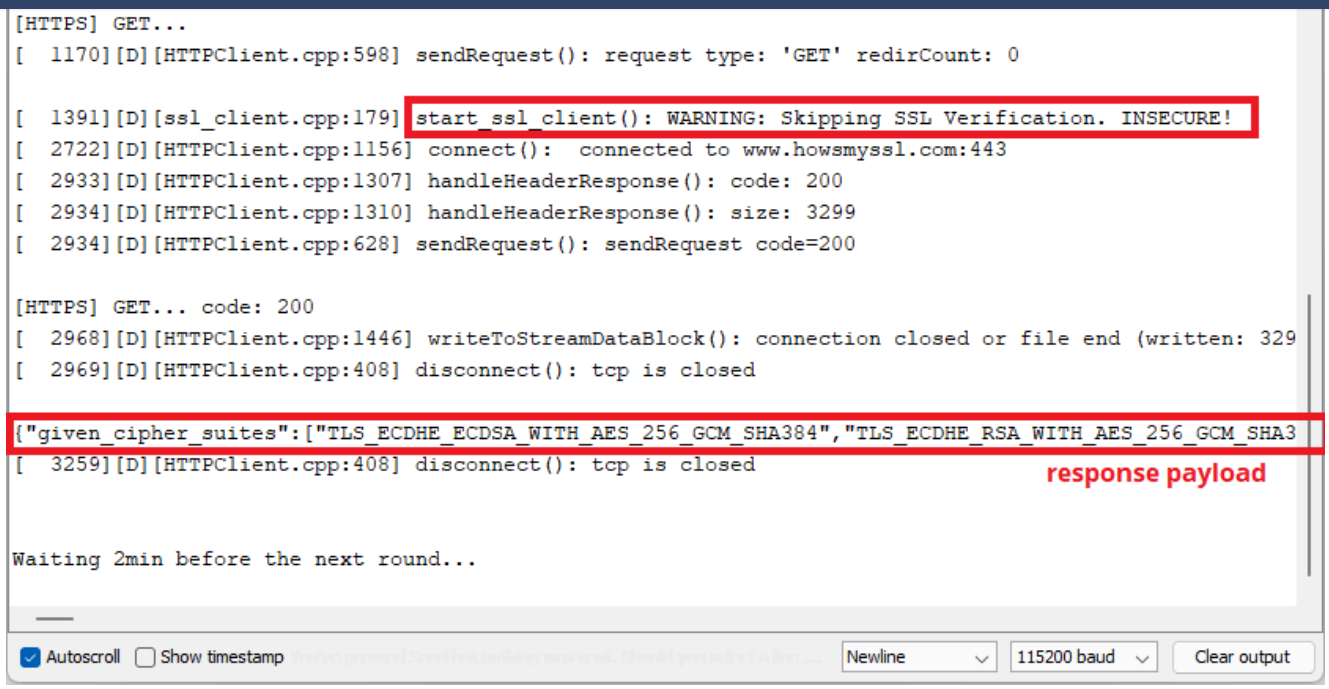
The complete example can be found below.

```
  Complete project details: https://RandomNerdTutorials.com/esp
  Based on the BasicHTTPSClient.ino example found at Examples >
*/

#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <HTTPClient.h>

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

void setup() {
  Serial.begin(115200);
  Serial.println();
  // Initialize Wi-Fi
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
  Serial.println(WiFi.localIP());
}
```

View raw code

With this example, your connection is still encrypted, but you won't be sure if you're talking to the right server. This scenario is useful for testing purposes.

After uploading this example, here's what you should get:



Your connection is still encrypted, but it will skip SSL verification.

Wrapping Up

In this tutorial, you learned how to make HTTPS requests with the ESP32. You also learned about the basic concepts of HTTPS protocol and about SSL/TLS certificates.

We've taken a look at examples with the WiFiClientSecure and HTTPClient libraries. The examples presented are as simple as possible so that you can modify them and apply them to your own projects. You learned how to make HTTPS requests with and without verification of the SSL/TLS certificate.

We hope you found this tutorial useful. We intend to create more tutorials about HTTPS and secure communication. Let us know in the comments below what you think.

Learn more about the ESP32 with our resources:

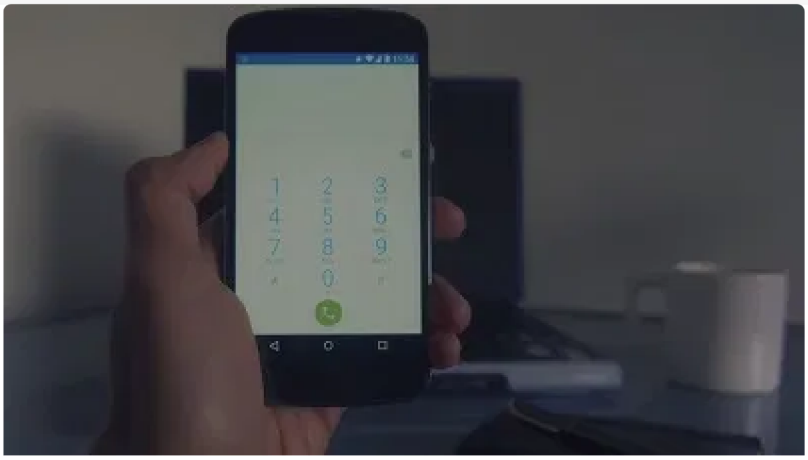
- Learn ESP32 with Arduino IDE
- Build Web Servers with ESP32 and ESP8266
- Firebase Web App with ESP32 and ESP8266
- Free ESP32 Projects and Tutorials

RANDOM NERD TUTORIALS

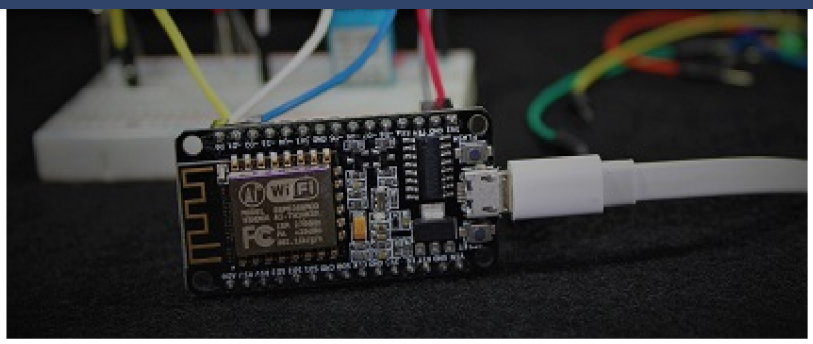
SMART HOME with Raspberry Pi, ESP32, ESP8266 [eBook]

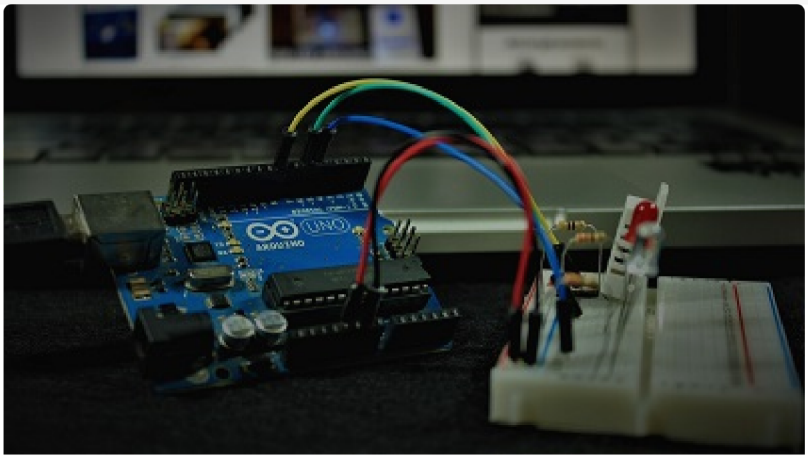
Learn how to build a home automation system and we'll cover the following main subjects: Node-RED, Node-RED Dashboard, Raspberry Pi, ESP32, ESP8266, MQTT, and InfluxDB database **DOWNLOAD »**

Recommended Resources

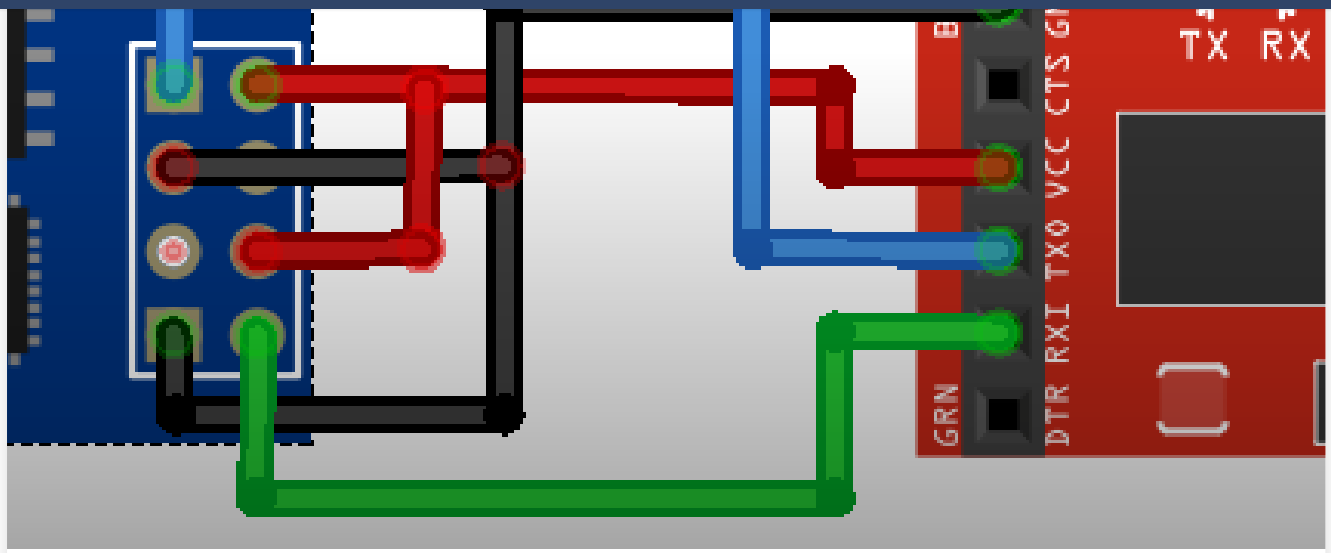


Build a Home Automation System from Scratch » With Raspberry Pi, ESP8266, Arduino, and Node-RED.

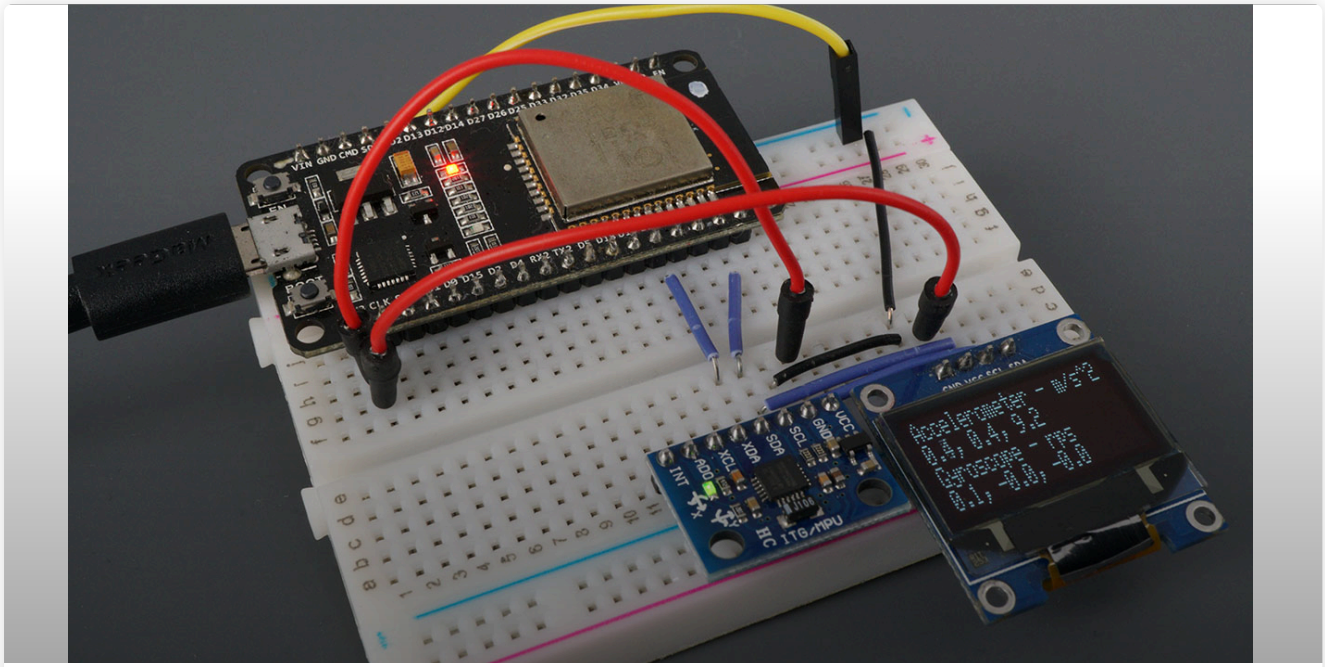




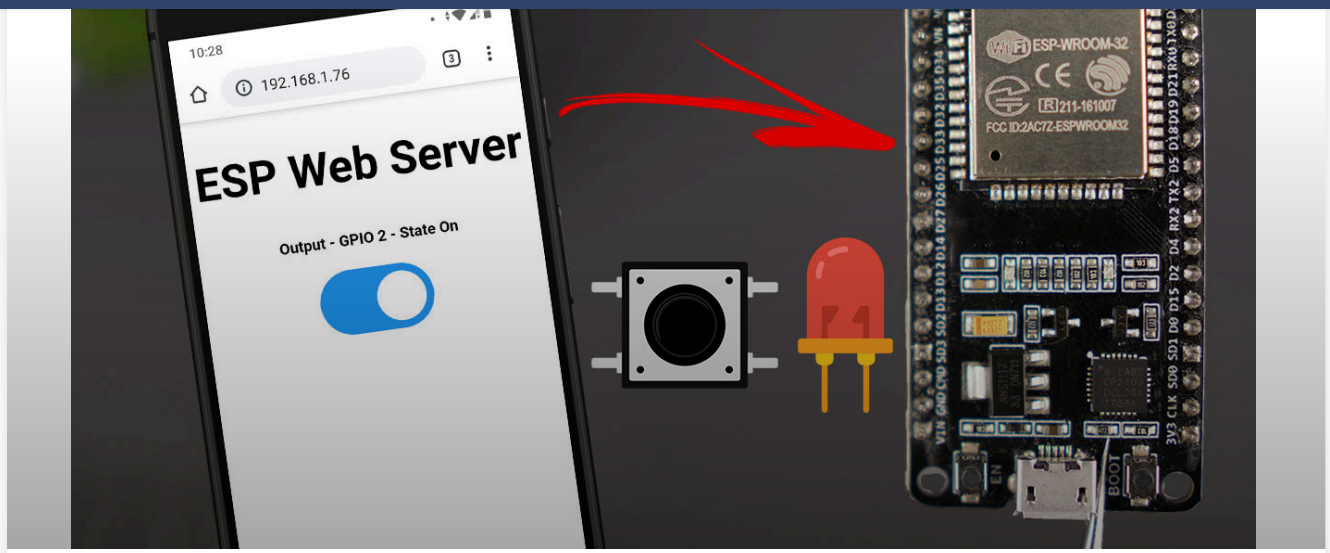
What to Read Next…



Flashing NodeMCU Firmware on the ESP8266 using Windows



ESP32 with MPU-6050 Accelerometer, Gyroscope and Temperature Sensor (Arduino)



ESP32/ESP8266: Control Outputs with Web Server and a Physical Button Simultaneously

Enjoyed this project? Stay updated by subscribing our newsletter!

Your Email Address

SUBSCRIBE

41 thoughts on “ESP32 HTTPS Requests (Arduino IDE)”

ThePartGuy

December 15, 2022 at 7:03 pm

I’m following this blog.

Reply

December 15, 2022 at 9:26 pm

IIUC, I think the “PUBLIC KEY” under the root certificate in your “certificate chain” diagram should be “PRIVATE KEY”.

Reply

Sara Santos

December 16, 2022 at 9:58 pm

Thanks for letting me know.
I’ll check that.
Regards,
Sara

Reply

jhonny

December 15, 2022 at 10:34 pm

👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍👍

Reply

December 15, 2022 at 11:37 pm

Thanks indeed for this wonderful and useful tutorial. We really appreciate your great effort.
Please would you shows us how to implement the secure connection with certificate on esp32 TTGO sim800l using it’s gprs.

Reply

Sara Santos

December 16, 2022 at 9:57 pm

Thanks for the suggestion.
I’ll add that to my list.
Regards,
Sara

Reply

Richard Barker

December 17, 2022 at 9:02 am

This would be very useful. I have been researching this very topic myself as I am using the T-Call v1.3 board myself for a weather station prototype. The SIM800L has proved very reliable however it is a legacy device that uses GPRS and not all revisions of the module support SSL. Even when it does support SSL, it uses an older standard of SSL that a significant number of services don’t support. My server will support the older standard but notably Amazon S3 servers apparently don’t. For

though the SIM800L is a very useful device still and will be good as long as your mobile network still supports GPRS. Thanks Rui and Sara for the tutorial.

Reply

mike

December 16, 2022 at 1:54 am

Thanks, this looks to be a great help & will try it out. I was wondering if you have a version for the ESP8266.

BTW, I've only came across one way (after many searches) to have an ESP talk to/activate an Alexa device vs Alexa activating and ESP device (many of those) but having a tough time (not compatible with new library versions):

https://github.com/Thomptronics/ESP8266-Virtual-Buttons

Reply

Sara Santos

December 16, 2022 at 9:56 pm

Hi.
Yes, we have a similar tutorial for the ESP8266 that will be published soon, so stay tuned!
Regards,
Sara

Andres Sanchez Miranda

December 16, 2022 at 7:13 am

Gracias por tu gran aportación
el formato de los ejemplos y su explicación son perfectos.

Muchas Gracias.

Me faltaría los libros en Español.

Reply

Sara Santos

December 16, 2022 at 9:55 pm

Muchas gracias.
Unfortunately, having the eBooks in multiple languages is beyond our means.
Regards,
Sara

Reply

Me-Chiel

December 16, 2022 at 8:03 am

host a webserver with some switches for homeautomation using the asyncwebserver? So i get a secure connection to turn lights on and off?

Reply

Sara Santos

December 16, 2022 at 9:54 pm

Hi.
Unfortunately not.
This shows how to make requests to a server, so the ESP32 is acting as a client and not as a web server.
Those are two different subjects.
Regards,
Sara

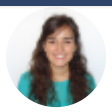
Reply

Dr_Phil

December 16, 2022 at 9:51 am

Excellent article, many thanks. There is actually a simpler way to include the certificate without needing to edit it, use:
static const char *rootCACertificate PROGMEM = R”EOF(
—–BEGIN CERTIFICATE—–
certificate contents
—–END CERTIFICATE—–
)EOF”;

Reply



Sara Santos

December 16, 2022 at 9:53 pm

Thanks for sharing.
I'll try it and edit the post.
Regards,
Sara

Reply

AL

December 16, 2022 at 8:50 pm

Thanks Sara ,Great deliverable
I'm a novice on encryption -would like to generate an example using a public and private key
and illustrate (serial monitor) the M (text just one letter i.e. "D") , the C (encrypted value) and back to M . I believe the tutorial s establish the certificate .However can I communicate with a server or website and readout the M,C, M as the algorithm goes thru it computation -. The computer science application that compliments the math-let the students witness the action -(nut and bolts under the hood)
I understand to a degree the modern algebra in the Cryptography process and now the certification part
Do I have to construct a website (or server) to do this . I would like to use it as a teaching tool for K 12 . Or is there a canned program / tutorial)that already exists .Can email any input off line – thanks

Reply

December 18, 2022 at 8:11 pm

From AI
Located a source pythonpool.com/rsa-encryption-python that nicely compliments this tutorial
Issue closed

Reply

Jose Serena

December 17, 2022 at 9:31 am

Congratulations, one of the best sites to learn ESP. Nearly everyday I check for new articles.

Reply

Sara Santos

December 18, 2022 at 4:29 pm

Thank you for your support.
Regards,
Sara

Reply

December 20, 2022 at 5:51 pm

Thanks for this, would be useful to see a MicroPython implementation also.

Reply

Sara Santos

December 21, 2022 at 11:38 am

Hi.
Thanks for the suggestion.
We will do that.
Regards,
Sara

Reply

Dave K

January 16, 2023 at 10:08 pm

Excellent and useful tutorial! Thank you!

Have you seen any information regarding encryption of ESP-NOW data exchange?

TIA,
Dave K.

Sara Santos

January 17, 2023 at 12:03 am

Hi.
We have a tutorial about that: https://randomnerdtutorials.com/esp32-esp-now-encrypted-messages/
Regards,
Sara

Reply

Dave K

January 17, 2023 at 12:32 am

Great! Thank you. I didn’t remember seeing it.
Best Regards,
Dave K.

Reply

Bilal

February 2, 2023 at 12:39 pm

I need the following libraries in Zip file;
WiFiClientSecure
HTTPClient.h

Reply

Walter

February 18, 2023 at 1:27 pm

ask Google 😉

Reply

Kees

March 24, 2023 at 12:19 pm

Thank you very much for this tutorial.
All sketches in your tutorial worked fine.
The website you used in the examples is http://www.whosmyssl.com
In your last sketch about “ESP32 HTTPS Requests without Certificate” I replaced http://www.whosmyssl.com with a random HTTPS site and got only the html file (source code) of that site in the Serial Monitor.
So how can I succesfully do a ESP32 HTTPS Requests without Certificate to a random website beginning with https://?

Reply

Sara Santos

March 25, 2023 at 11:21 am

I'm sorry but I didn't understand your question.
You just need to use our examples that don't use certificate. The connection will be encrypted, but the board won't check for the validity of the certificate.
Regards,
Sara

Reply

Kees
March 28, 2023 at 2:40 pm

Hi,
Meanwhile it is clear to me that the https:// website had to be prepared to get and post the encrypted data.
So please forget my question for now. I am deeping first in the HTTP matter.

Reply

Kees
March 29, 2023 at 7:32 am

Hi,
A few months ago I had succesfully built the ESP webserver using the BME280 sensor (https://randomnerdtutorials.com/esp32-bme280-arduino-ide-pressure-temperature-humidity/)
In that example every device connected to that wifi network could see the web presentatation showing the data of the sensor.

Reply

Mojeeb

April 15, 2023 at 11:46 am

I'm using the same code (copy, paste), and the following is what appeared:

Starting connection to server…
[6255][E][WiFiGeneric.cpp:1476] hostByName(): DNS Failed for
[6255][E][WiFiClientSecure.cpp:135] connect(): start_ssl_client: -1
Connection failed!

Reply

Mojeeb

April 15, 2023 at 11:52 am

Solved

Reply

Crowley723

April 20, 2023 at 10:41 pm

seem to be a .addHeader() function in WiFiClientSecure.h. Any help would be appreciated.

Reply

Dev

June 14, 2023 at 11:52 am

Hi,

is WiFiClientSecure works with ETH as well.
I am having issues with certificate (X509 – Certificate verification failed, e.g. CRL, CA or signature check failed) but something tells me that real problem is in that i do not use Wifi but ETH instead.

if that's the case (ETH) do you have alternative solution for HTTPS client with ethernet connection?

Best regards

Reply

Sara Santos

June 21, 2023 at 9:41 am

Hi.
No. We don't have such example.
What's exactly the error that you're getting?
Regards,
Sara

Piet Dummie

June 27, 2023 at 1:12 pm

Hi,

I've been fighting with this tutorial for hours now and I can't get it to work, on my own server that is. The sketch (I'm using the version using HttpClient) works fine with the howsmyssl website but if I use my own server it doesn't work.

The rootCACertificate for my home server is the exact same as for how howsmyssl so the only change I made is to the url which I changed from https://www.howsmyssl.com/a/check to:

https://edwinov.com/phptry/aapkut.php

Please tell me what I'm doing wrong here as I can't seem to figure it out by myself.

Reply

Sara Santos

June 28, 2023 at 9:11 am

Hi.
What is exactly the error that you get?
Regards,
Sara

Reply

Mynhardt K

July 11, 2023 at 6:41 pm

Hi,

Thank you very much for this sharing this great information!

i tried the above example code for ESP32 HTTPS Requests without Certificate and changed the 2-minute delay(120000) duration to 5 seconds delay(5000). After a few minutes – I get the below error:

[628782][E][ssl_client.cpp:36] _handle_error(): [start_ssl_client():263]: (-16) BIGNUM – Memory allocation failed
[628782][E][WiFiClientSecure.cpp:135] connect(): start_ssl_client: -16
[HTTPS] GET… failed, error: connection refused

When the memory runs out and the ESP32 board reboots after a panic.

I added the below line before the delay to check whether the memory is released or not.
Serial.println(ESP.getFreeHeap());
I realized the memory is not cleaned up and is getting smaller.

I tried adding client->stop(); at the end, but seems that the stop() is not releasing the internal resources / stopping the connection.

Can you kindly please advise if client->stop(); is the correct way to cleanup the memory?

Best Regards

Mynhardt

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <HTTPClient.h>

// Replace with your network credentials
const char* ssid = “SSID”;
const char* password = “Password”;

void setup() {
Serial.begin(115200);
Serial.println();
// Initialize Wi-Fi
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
Serial.print(“Connecting to WiFi ..”);
while (WiFi.status() != WL_CONNECTED) {
Serial.print(‘.’);
delay(1000);
}
Serial.println(WiFi.localIP());
}

void loop() {
WiFiClientSecure *client = new WiFiClientSecure;
if(client) {
// set secure client without certificate
client->setInsecure();
//create an HTTPClient instance
HTTPClient https;

//Initializing an HTTPS communication using the secure client
Serial.print("[HTTPS] begin...\n");
if (https.begin(*client, "https://www.howsmyssl.com/a/check")) { //
HTTPS
Serial.print("[HTTPS] GET...\n");
// start connection and send HTTP header
int httpCode = https.GET();
```

```
// HTTP header has been send and Server response header has been handled
Serial.printf("[HTTPS] GET... code: %d\n", httpCode);
// file found at server
if (httpCode == HTTP_CODE_OK || httpCode == HTTP_CODE_MOVED_PERMANENTLY)
{
// print server response payload
String payload = https.getString();
Serial.println(payload);
}
}
else {
Serial.printf("[HTTPS] GET... failed, error: %s\n",
https.errorToString(httpCode).c_str());
}
https.end();
}

}
else {
Serial.printf("[HTTPS] Unable to connect\n");
}
Serial.println();
Serial.println("Waiting 5sec before the next round...");
Serial.println(ESP.getFreeHeap()); //Check heap before
client->stop();
Serial.println(ESP.getFreeHeap()); //Check heap after
delay(5000); //5 seconds
}
```

Reply

Mynhardt K

July 11, 2023 at 7:50 pm

I managed to resolve the issue with replacing
client->stop();
with
delete client;

The memory utilization is now stable.

Regards
Mynhardt

Reply

Sara Santos
July 12, 2023 at 9:35 am

Great.
Thanks for sharing your solution.
Regards,
Sara

Reply

Steve Smith
November 18, 2023 at 7:01 am

Sara
I have an implementation of HTTPS POST currently functional using set insecure. I want to secure the connection and use either fingerprint or certificate. This is ok as the certificate can be stored in the SPIFFS with the

about its expiry date when the client is in an unmanned location?
The payload at present is tiny <100 bytes as a json object the application is on an esp8266.

Reply

Leave a Comment

Name *

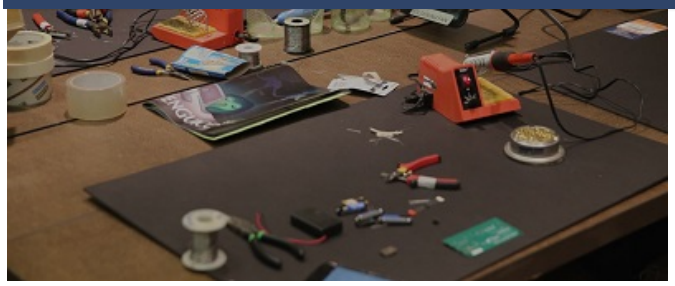
Email *

Website

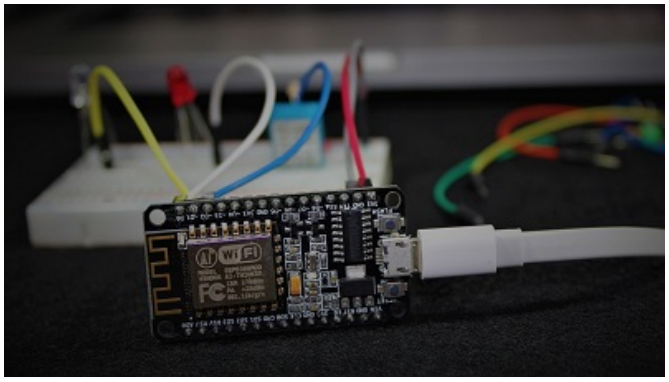
Notify me of follow-up comments by email.

Notify me of new posts by email.

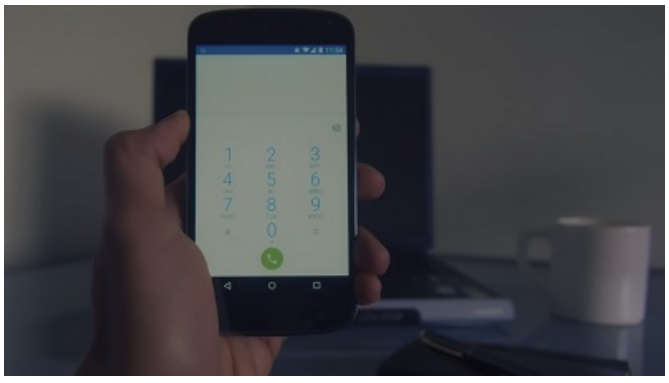
Post Comment



Visit Maker Advisor – Tools and Gear for makers, hobbyists and DIYers »



Home Automation using ESP8266 eBook and video course » Build IoT and home automation projects.



Build Web Servers with ESP32 and ESP8266 » boards to control outputs and monitor sensors remotely.