AN4963
Application note

Security on SPWF04S module

Introduction

The SPWF04S series[a] of Wi-Fi modules feature security functions designed to preserve *confidentiality*, *communication integrity* and *authentication* during wireless communication and Internet connection, on at least two levels.

The first level involves communication between peers as in access to a web server. The communication data must not be read by entities other than the client and the server; the client must ensure that the peer it communicates with is authentic. This feature is provided by the *Transport Layer Security* (TLS) protocol, which allows client and server applications to communication that is confidential and secure.

The second level involves communication between the Wi-Fi device and the Access Point. Radio communication is very easy to intercept: an attacker just needs an antenna to read transmission packets. Encryption is a key instrument in ensuring that packets cannot be read by anything other than the two peers in communication. Security at the level of Wi-Fi network is provided by WPA2-PSK when dealing with personal networks, and WPA2-Enterprise when dealing with enterprise networks. For WPA2-PSK, setting up the module for the network is facilitated by Wi-Fi protected setup (WPS).

Devices that support firmware updating via Wi-Fi or *firmware over-the-air* (FOTA) must be able to assess the authenticity of the firmware source and the integrity of the image file after it has been received.

[a] SPWF04Sxxx Wi-Fi module, www.st.com/wifimodules.

Contents

1 Transport layer security (TLS) protocol overview 6
1.1 TLS sub protocols 7
1.1.1 Handshake protocol 7
1.1.2 Change cipher spec protocol 9
1.1.3 Alert protocol 9
1.1.4 Record protocol 9
1.2 Authentication and certificates 10
2 Wi-Fi Protected Access (WPA) overview 13
2.1 WPA2-PSK 13
2.1.1 Wireless protected setup (WPS) 14
2.2 WPA2-Enterprise 15
2.2.1 EAP-MD5 and EAP-MSCHAPv2 16
2.2.2 EAP-TLS 17
2.2.3 EAP-TTLS and PEAP 18
3 SPWF04S security 20
3.1 TLS on SPWF04S 20
3.1.1 Supported ciphers list 20
3.1.2 Domain name check 21
3.1.3 Authentication in client mode 21
3.1.4 Authentication in server mode 24
3.1.5 Certificates and keys 27
3.1.6 Create a secure socket 29
3.1.7 HTTPS and SMTPS 30
3.2 Wi-Fi protected access on SPWF04S 32
3.2.1 WPA2-PSK 32
3.2.2 WPA2-Enterprise 33
3.2.3 Connection to the network 38
3.3 Secure FOTA (firmware over-the-air) 38
3.3.1 How to generate a FOTA 39
4 SPWF04S TLS tutorial 41
4.1 Example: TLS client with mutual authentication 41
4.2 Example: TLS client with one-way authentication 41
4.3 Example: TLS server with mutual authentication 42
4.4 Example: TLS server with one-way authentication 43

4.5 Example: Amazon with one-way authentication 44
4.6 Example: HTTPS with Amazon 44
4.7 Example: SMTPS with Gmail 45
Appendix A OpenSSL - useful commands 46
Appendix B Extract Root CAs from web browser 49
Appendix C How to set up a WPA2-Enterprise network 54
5 Revision history 61

List of tables

Table 1: Ciphers....8
Table 2: TLS cipher suites....20
Table 3: Commands to obtain the current time in seconds from a PC....29
Table 4: Options for AT+S.SOCKON command....29
Table 5: Options for AT+S.SOCKDON command....30
Table 6: Options for AT+S.HTTPGET and AT+S.HTTPPOST command for the TLS option....31
Table 7: Options for AT+S.SMTP command for TLS option....32
Table 8: Encryption protocols....33
Table 9: WPA2-Enterprise features on SPWF04S....34
Table 10: EAP authentication methods....34
Table 11: Document revision history....61

List of figures

Figure 1: SSL/TLS protocol architecture....6
Figure 2: SSL/TLS full handshake procedure....7
Figure 3: Record protocol operations....9
Figure 4: Certificate chain or chain of trust.....12
Figure 5: WPA2-PSK four-way handshake....14
Figure 6: WPA2-Enterprise architecture....15
Figure 7: EAP-MD5 protocol....16
Figure 8: EAP-TLS protocol....17
Figure 9: EAP-TTLS protocol....18
Figure 10: AT commands for TLS one-way authentication in client mode....22
Figure 11: AT commands for TLS mutual authentication in client mode....23
Figure 12: AT commands for TLS one-way authentication in server mode....25
Figure 13: AT commands for TLS mutual authentication in server mode....26
Figure 14: FOTA packet structure....39
Figure 15: HTTPS in Google Chrome....49
Figure 16: Security Overview window in Google Chrome....50
Figure 17: Certificate window – Certification path....51
Figure 18: Certificate window – Details....52
Figure 19: Certificate Export Wizard....53
Figure 20: IronWifi’s dashboard....56
Figure 21: IronWifi's Networks page....56
Figure 22: 802.1X server information on IronWifi....57
Figure 23: IronWifi's Users page....57
Figure 24: IronWifi's created user page....58
Figure 25: Configuration page for Linksys Access Point....59
Figure 26: WPA2-Enterprise configuration for Linksys Access Point....60

1 Transport layer security (TLS) protocol overview

Originally developed by Netscape in mid 1990s, the Secure Sockets Layer (SSL) is a cryptographic protocol designed to provide communication security over the Internet[2]. Version 1.0 was never publicly released; version 2.0 was released in February 1995 but "contained a number of security flaws which ultimately led to the design of SSL version 3.0"[3]. SSLv3.0 was a complete redesign of the protocol and is still widely supported (since 1996).

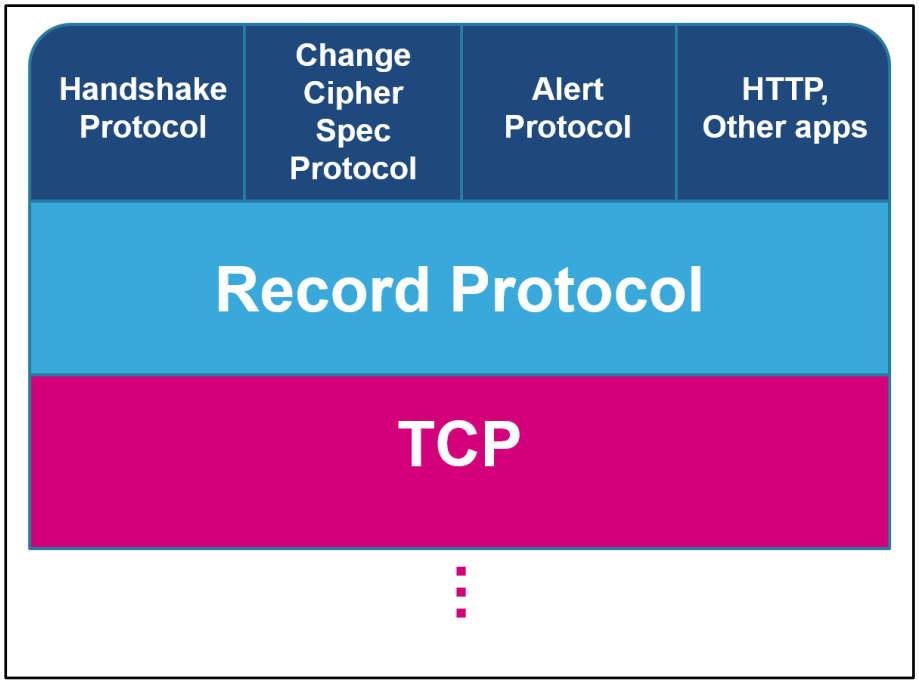
The IETF standard body adopted SSLv3.0 with minor tweaks and published it as *Transport Layer Security* (TLS) version 1.0[4]: the two versions are very similar, but interoperability is precluded. TLSv1.2 is the latest recommended version, offering flexibility and key features that were unavailable in earlier protocol versions.

All TLS versions were further refined [5]removing their backward compatibility with SSL such that TLS sessions will never negotiate the use of SSLv2.0.

SSL/TLS is typically applied into the TCP/IP protocol stack and provides *security services on top of the transport layer*. The protocol is composed of two layers: the *TLS Record layer* and the *TLS Handshake layer*.

At the lowest level, layered on top of some reliable transport protocol, there is the *TLS Record Protocol*. The Record Protocol is used for encapsulation of various higher-level protocols and provides two basic properties for a secure protocol: *confidentiality* and *integrity.*

Figure 1: SSL/TLS protocol architecture



[2] T. Dierks, The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, 2008.

[3] Wikipedia, Transport Layer Security, https://en.wikipedia.org/wiki/Transport_Layer_Security.

[4] T. Dierks and C. Allen, The TLS Protocol Version 1.0. RFC 2246, 1999.

[5] S. Turner and T. Polk, Prohibiting Secure Sockets Layer (SSL) Version 2.0, 2011.

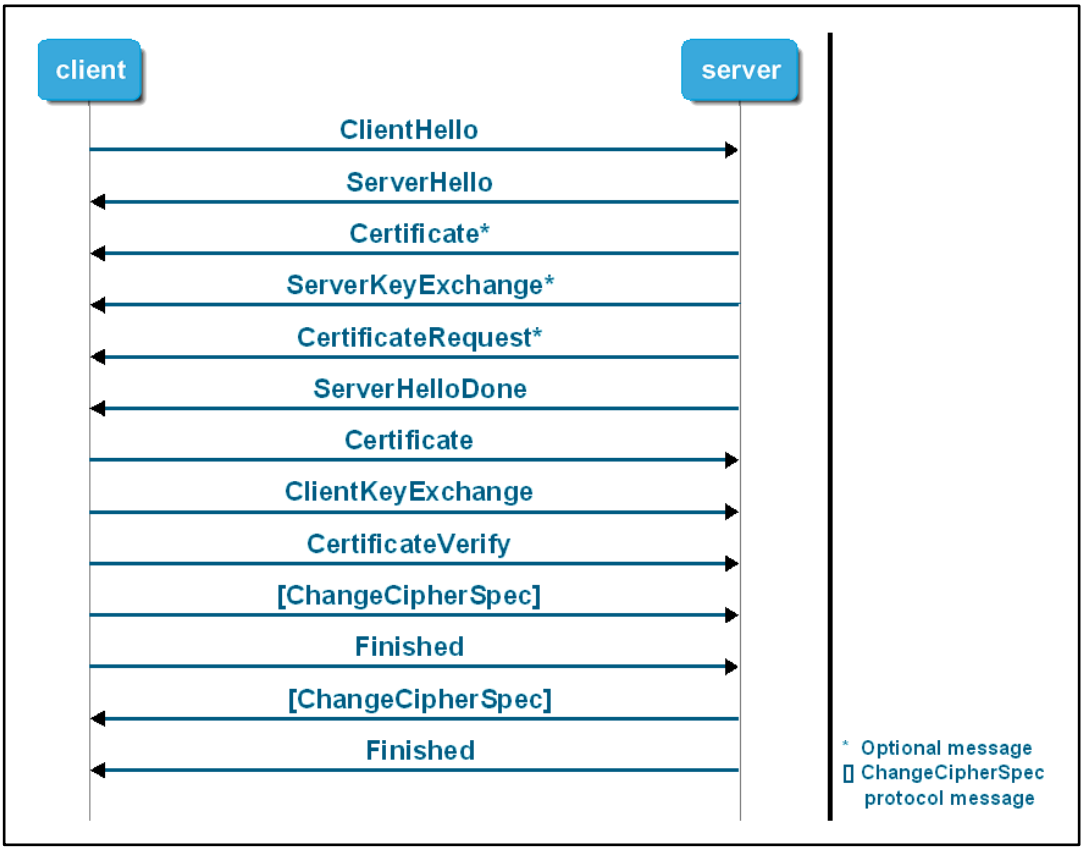
The TLS Handshake layer consists of sub-protocols:

1. Handshake
2. Change Cipher Spec
3. Alert

The Handshake protocol is the most complex part of TLS and provides a number of very important security functions. It allows server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security with the following attributes:

1. Cipher suite negotiation
2. Authentication of the server and, optionally, of the client
3. Session key information exchange.

Figure 2: SSL/TLS full handshake procedure



1.1 TLS sub protocols

1.1.1 Handshake protocol

This sub-protocol is used to negotiate session information between the client and the server. The session information consists of a session ID, peer certificate(s), the cipher suite, the compression algorithm, and a shared secret that is used to generate session key.

Figure 2: "SSL/TLS full handshake procedure" depicts the message flow for a full Handshake process. The *optional* value indicates optional or situation-dependent messages: in mutual authentication, a TLS server has to send its certificate and request a

certificate from the client, while in anonymous negotiation the *optional* messages may be skipped.

Certificate-based authentication is examined more in detail in *Section 1.2: "Authentication and certificates"*.

1. The client sends a `ClientHello` message specifying the highest supported SSL/TLS protocol version (SSLv3.0, TLSv1.0, 1.1 or 1.2), a random number, a list of suggested cipher suites and compression methods.
2. The server responds with a `ServerHello` message, containing the chosen protocol version, another random number, cipher suite and compression method from the choices offered by the client, and the session ID. The chosen protocol version should be the highest that both client and server support. The client and the server have to support at least one common cipher suite, otherwise the Handshake protocol fails. The server generally chooses the strongest common cipher suite they both support.
3. The server sends its digital certificate in an optional `Certificate` message, for example, the server uses X.509 digital certificates.
4. Additionally, a `ServerKeyExchange` message may be sent, if it is required (e.g., if the server has no certificate, or if its certificate is for signing only).
5. If the server requires a digital certificate for client authentication, an optional `CertificateRequest` message is appended.
6. The server sends a `ServerHelloDone` message indicating the end of this phase of negotiation.
7. If the server has sent a `CertificateRequest` message, the client has to send the `Certificate` message. For example the Client uses an X.509 digital certificate.
8. The client sends a `ClientKeyExchange` message. This message contains the premaster secret used in the generation of the symmetric encryption keys and the message authentication code (MAC) keys. The client encrypts pre-master secret with the public key of the server. The public key is sent by the server in the digital certificate or in ServerKeyExchange message.
9. If the client has sent a digital certificate to the server, the client sends a `CertificateVerify` message signed with the client's private key. By verifying the signature of this message, the server can explicitly verify the ownership of the client digital certificate.
10. The client sends a `ChangeCipherSpec` message announcing that the new parameters (cipher method, keys) have been loaded.
11. The client sends a `Finished` message; it is the first message encrypted with the new cipher method and keys.
12. The server responds with a `ChangeCipherSpec` message and a `Finished` message from its end.
13. The TLS Handshake protocol ends and the encrypted exchange of application data can be started.

During the initial handshaking phase, the client and server negotiate **cipher suites,** which **specify a cipher for each of the following functionalities:**

Table 1: Ciphers

Functionality	Cipher
Authentication	RSA, DSA, ECDSA
Key-exchange/agreement	RSA, DH, ECDH, PSK
Symmetric ciphers for encryption	RC4, IDEA, DES, 3DES, AES or Camellia.
Hash	MAC (for SSLv3.0) or HMAC with MD2, MD4, MD5, SHA-1, SHA-256 (after TLSv1.1 and 1.2 standards).

A complete list of SSL/TLS cipher suites can be found in the registry maintained by the Internet Assigned Numbers Authority (IANA)[6].

1.1.2 Change cipher spec protocol

The Change cipher spec protocol is used to change the keying material used for encryption between the client and server. Keying material is raw data that is used to create keys for cryptographic use. The Change Cipher Spec protocol consists of a single message to tell other party in the SSL/TLS session that the sender wants to change to a new set of keys. The key is computed from the information exchanged by the Handshake protocol.

1.1.3 Alert protocol

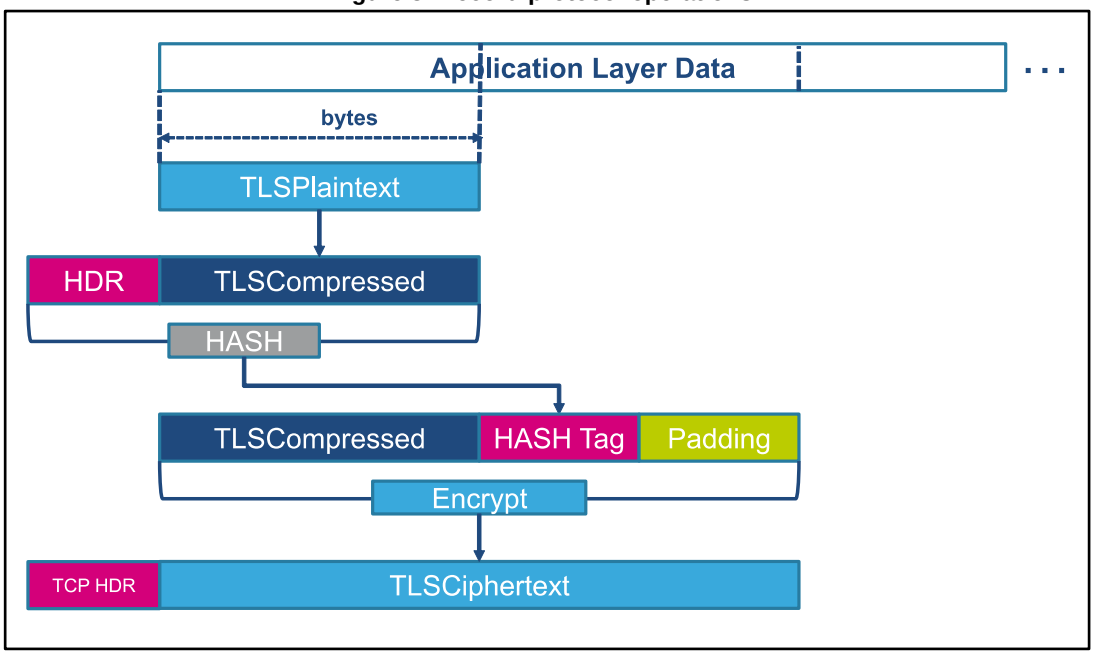
Alert messages are used to indicate a change in status or an error condition to the peer. There is a wide variety of alerts to notify the peer of both normal and error conditions. Alerts are commonly sent when the connection is closed, an invalid message is received, a message cannot be decrypted, or the user cancels an operation.

1.1.4 Record protocol

The Record protocol receives and encrypts data from the higher-layer and delivers it to the Transport Layer. As shown in *Figure 3: "Record protocol operations"*, the Record Protocol takes the data, fragments it into *TLSPlaintext* blocks with a size appropriate to the cryptographic algorithm. Then it optionally compresses (or, for data received, decompresses) the *TLSPlaintext*, applies a MAC or HMAC (HMAC is supported only by TLS) to get the hash tag. Finally the *TLSCompressed* data and hashtag (and some padding eventually) are concatenated and encrypted (or decrypted) using the information negotiated during the Handshake Protocol.

Encryption and hash ensure respectively the confidentiality and the integrity of the plaintext.

Figure 3: Record protocol operations



[6] http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml

1.2 Authentication and certificates

SSL/TLS requires a server certificate and, optionally, a client certificate. The digital certificate certifies the ownership of a public key by the named subject of the certificate, also known as public key certificates. This allows others parties to rely upon signatures or assertions made by the private key that corresponds to the public key that is certified.

Digital certificates used in SSL/TLS comply with the X.509 standard[7], which specifies the information required and the formats for public key certificates. In an X.509 system, the subject of the certificate is identified by a Distinguished Name (DN). A DN is a series of name-value pairs that uniquely identify an entity. The following attribute types are commonly found in the DN:

- **CN**: Common Name
- **T**: Title
- **O**: Organization name
- **OU**: Organization Unit name
- **L**: Locality name
- **ST (or SP or S)**: State or Province name
- **C**: Country

The X.509 standard provides for a DN to be specified in a string format. For example:

- CN=John, O=STM, OU=Test, C=IT

The Common Name (CN) can describe an individual user or any other entity like a Web server. DNs may include a variety of other name-value pairs; indeed, the rules governing the construction of DNs can be complex[8].

In this model of trust relationships, a Certification Authority (CA) is an independent and trusted third party that issues digital certificates to provide you with an assurance that the public key of an entity truly belongs to that entity. The roles of a CA are:

- On receiving a request for a digital certificate, to verify the identity of the requestor before building, signing and returning the personal certificate
- To provide the CA's own public key in its CA certificate
- To publish lists of certificates that are no longer trusted in a Certificate Revocation List (CRL).

An X.509 certificate issued by the CA binds a particular public key to the name of the DN the certificate identifies. Only the public key certified by the certificate will work with the corresponding private key possessed by the DN identified by the certificate.

The contents of a certificate, according to the X.509 version 3 specifications, may include:

- The version number of the X.509 standard supported by the certificate.
- The certificate's serial number. Every certificate issued by a CA has a serial number that is unique among the certificates issued by that CA.
- Information about the user's public key, including the algorithm used and a representation of the key itself.
- The DN of the CA that issued the certificate.
- The period during which the certificate is valid;

[7] D. Cooper, S. Santesson, S. B. S. Farrel, R. Housley and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and CRL profile. RFC 5280," 2008.

[8] S. Kille, Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names. RFC 4514, 2006.

- The DN of the certificate subject, which is also called the subject name; for example, in a TLS client certificate, this is the user's DN.
- Optional certificate extensions.
- The cryptographic algorithm, or cipher, used by the issuing CA to create its own digital signature.
- The CA's digital signature, obtained by hashing all of the data in the certificate together and encrypting it with the CA's private key.

The optional certificate extensions introduced in X.509v3, can be used to provide additional data for the client or server. Examples of X.509v3 extensions are:

- The *Basic Constraints* extension, which is used to indicate a CA certificate and the depth of the subordinate CAs path.
- The *Subject Key Identifier* extension, which contains a 20 bytes hexadecimal value used to identify the certificate. Usually is a function of the certificate's public key (e.g. using a hash function).
- The *Authority Key Identifier* extension, which contains the Subject Key Identifier of the certificate issuer (i.e. defines the public key which has signed the certificate).

The text format certificate is structured thus:

```
-----BEGIN CERTIFICATE-----
```

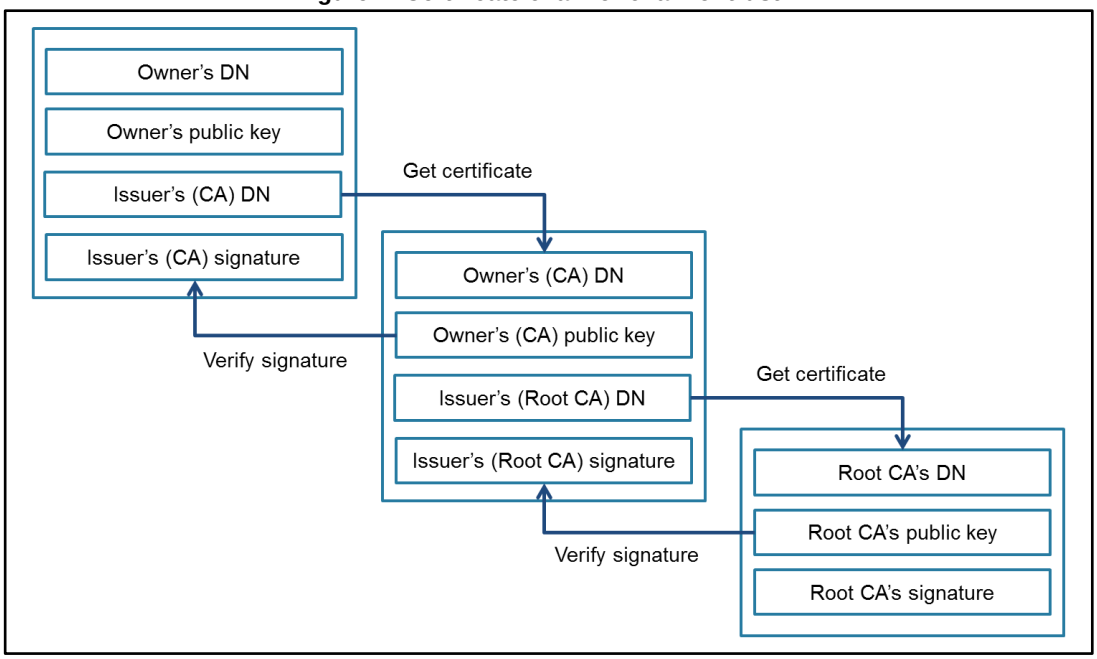
```
…certificate data (base-64 encoded)
```
[9]

```
-----END CERTIFICATE-----
```

Usually the certificate validation isn't made by just one CA, but instead by a **certificate chain**. The chain, or path, begins with the certificate of that entity, and each certificate in the chain is signed by the entity identified by the next certificate in the chain. The chain terminates with a root CA certificate. The root CA certificate is always signed by itself, it must be considered as a trusted CA and must be available in the application (e.g. TLS client, web browser). The signatures of all certificates in the chain have to be verified until the root CA certificate is reached. *Figure 4: "Certificate chain or chain of trust."* illustrates a certification path from the certificate owner to the root CA, where the chain of trust begins. Notice that different chains can have multiple or even none intermediate CAs. For a root CA, the Authority Key Identifier extension value is equal to its own Subject Key Identifier extension.

[9] J. Linn, Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. RFC 1421, 1993.

Figure 4: Certificate chain or chain of trust.



In some cases it would be easier and less expensive using self-signed certificates; e.g., for testing purposes or when the parties know and trust each other. A self-signed certificate is a certificate that is signed by the same entity whose identity it certifies and there is no need for an external CA.

TLS supports these **authentication modes**:

1. Mutual authentication - both parties (client and server) share their signed certificates and authenticate each other. Mutual authentication provides stronger security by assuring that the identity on both sides of the communication are known.
2. One-way authentication - only the server sends its signed certificate and is authenticated by the client. The client is not required to send the server a digital certificate and remains unauthenticated (no certificate).
3. Anonymous – neither entity authenticates the identity of the other party.

Each party is responsible for verifying that the other's certificate is valid and has not expired or been revoked. **In case of one-way or mutual authentication**, because certificate validation requires that root CA keys are distributed independently, **it is assumed that the remote end already possess root CA certificate to accomplish the validation.**

2 Wi-Fi Protected Access (WPA) overview

The Wi-Fi Protected Access (WPA) protocol provides security at level 2 of the ISO/OSI stack in the *radio communication* between a Wi-Fi device (called also *Station*) and a Wi-Fi Access Point (AP). A first version of the protocol (simply called WPA) was released in 2002[10] to overcome the security weaknesses (discovered in 2001) of its ancestor, *Wired Equivalent Privacy* (WEP)[11]. WPA is considered just as a temporary software patch for Wi-Fi devices that already support WEP. In 2004 a completely new protocol was released, called IEEE 802.11i[12]. but usually referred to as WPA2, and became mandatory in 2006 for Wi-Fi certified devices[13].

WPA2 provides two modes. The first mode is designed for small Wi-Fi networks, like home networks, and it is usually referred as *WPA2-Personal* or *WPA2-PSK*. The second mode provides more security and authentication and it is mainly used for big networks, like hospitals, universities or workplaces, and it is usually referred to as *WPA2-Enterprise.*

2.1 WPA2-PSK

In WPA2-PSK the communication between the Station (STA) and the Access Point (AP) is encrypted by AES cipher with 256 bit symmetric key. The key, also called *Pairwise Transient Key* (PTK) is generated at every session starting from the so-called *Pre-Shared Key* (PSK), which is a secret alphanumerical string associated to the AP. This is the one the user has to know and specify into the device in order to join the network.

The protocol used to derive the key is called *four-way handshake.*

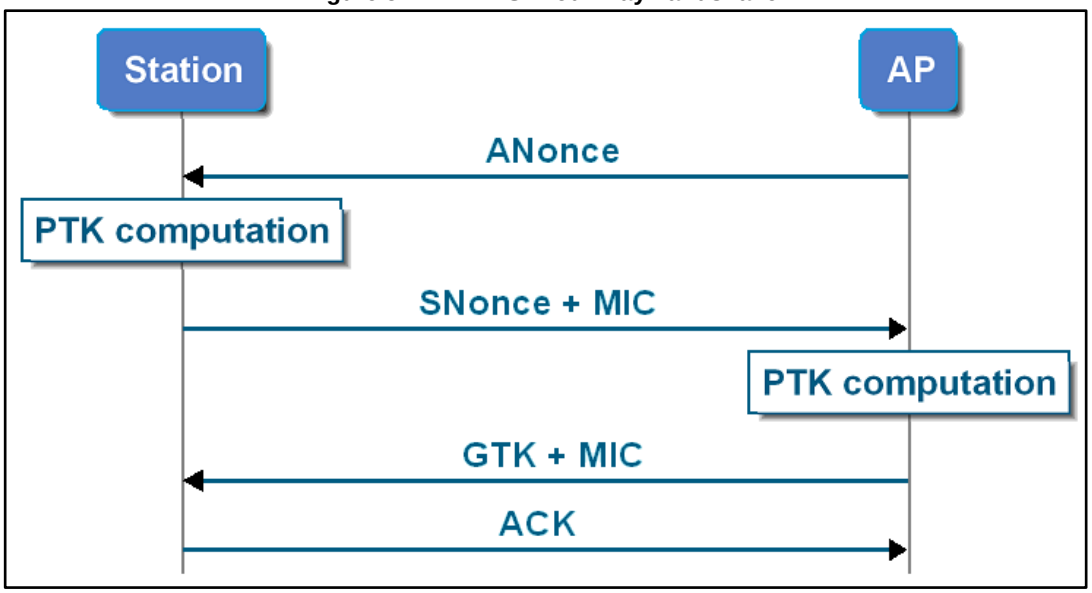
[10] Wi-Fi Alliance, "Wi-Fi Protected Access: Strong, standards-based, interoperable security for today's Wi-Fi networks," White paper, University of Cape Town, pp. 492-495, 2003.

[11] A. Stubblefield, J. Ioannidis and A. D. Rubin, "A key recovery attack on the 802.11 b wired equivalent privacy protocol (WEP)," ACM transactions on information and system security (TISSEC), vol. 7, no. 2, pp. 319-332, 2004.

[12] IEEE Standard for information technology, 802.11i-2004, 2004.

[13] Wi-Fi Alliance, "WPA2™ Security Now Mandatory for Wi-Fi CERTIFIED™ Products," 13 March 2006. [Online]. Available: http://www.wi-fi.org/news-events/newsroom/wpa2-security-now-mandatory-for-wi-fi-certified-products. [Accessed 21 July 2016].

Figure 5: WPA2-PSK four-way handshake



- AP generate a *nonce* (a random number which must be used just one time) called *ANonce,* and sends it to STA.
- STA generates another nonce, called *SNonce*, and computes the PTK, which is a function of the PSK, the two nonces and the MAC addresses of the two entities. Then STA sends the SNonce with a *Message Integrity Code* (MIC) to AP.
- Now also AP is able to compute the PTK. It also computes the *Group Temporal Key* (GTK), which is used to encrypt/decrypt multicast and broadcast messages. AP sends the encrypted GTK and a new MIC to STA.
- STA sends an acknowledgement to AP.

Once the handshake ends correctly, the device has the access to the network and the communication between it and the Access Point is encrypted with the previously computed key.

2.1.1 Wireless protected setup (WPS)

In order to simplify the setup of WPA2-PSK, especially when dealing with devices without advanced input/output as keyboards or monitors, like printers or embedded systems, a network security standard called *Wi-Fi Protected Setup* (WPS)[14] was defined in 2006. WPS permits to the user to easily setup a wireless device in order to establish a connection to a Wi-Fi network using WPA2-PSK.

The standard defines several ways to implement WPS. The most common are two:

- *WPS PIN*: the wireless device provides an 8 digits PIN to the Access Point, which could be read from a sticker on the device or randomly generated by the device if it provides a display. Alternatively the user can insert into the device the 8 digit PIN of the Access Point (reported on a sticker attached to it). This setup method is mandatory for any WPS certified device.

[14] Wi-Fi Alliance , Wi-Fi protected setup specification, 2007.

In 2011, a major vulnerability in PIN mode was discovered [15] , involving the fact that the check of the PIN is made in two steps: on the first four digits, and then on the remaining three (the last digit is a checksum). An attacker is therefore able to guess the PIN in 11000 attempts. Users are encouraged to disable WPS PINs from their networks.

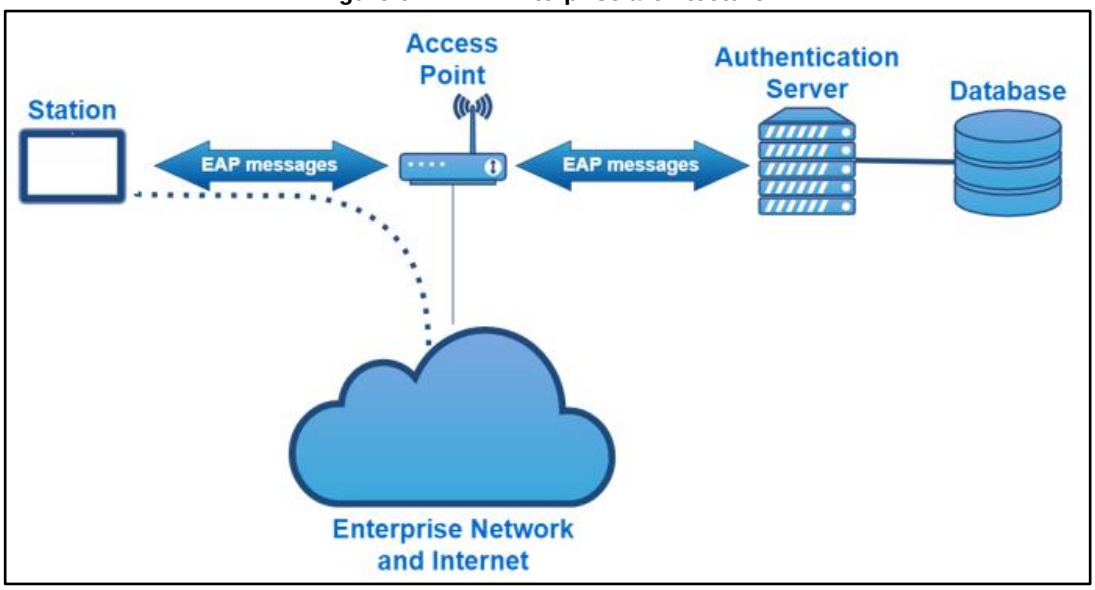
- *WPS PBC (Push Button Configuration)*: the Access Point and the Wireless Station have a push button on their side, which could be either a physical or a virtual button. In order to establish an association, this method requires the user to press the push button on both the Access Point and the Station, within a time interval called *Walk Time*, which is usually 2 minutes. It doesn't matter on which of the two devices the button is pressed first. This method assumes that an attacker can't have physical access to the Access Point, so he can't press the button for getting authenticated to the network.

Once the association is made, using one of the two methods explained above, a messages exchange between the two devices is performed, in order to create a secure tunnel for sending the PSK to the Station. Finally, when the Station gets the PSK, it is able to perform WPA2-PSK in order to be authenticated to the network.

2.2 WPA2-Enterprise

WPA2-Enterprise has been designed in order to provide wireless network access in large environments like enterprises, hospitals or universities, in which several Access Points are used, sharing the same SSID. In WPA2-Enterprise authentication is not made by the Access Points themselves, but from an *Authentication Server* (AS), which have access to a *Database* containing all the data about authorized users (e.g. certification authorities, usernames and passwords, etc.). An example of a WPA2-Enterprise architecture is illustrated in *Figure 6: "WPA2-Enterprise architecture"*.

Figure 6: WPA2-Enterprise architecture



Until the authentication of the Station is made, the only allowed messages between the Access Point and the Station are the ones specified by the *Extensible Authentication*

[15] S. Viehböck, Brute forcing Wi-Fi protected setup, 2011.

Protocol (EAP)[16]. These messages are forwarded to the Authentication Server using the *Remote Authentication Dial in User Service* (RADIUS) protocol[17].

There exist several *authentication methods* which can be used by the Application Server to authenticate the Station. They mainly consist on EAP messages exchange between the Application Server and the Station via the Access Point.

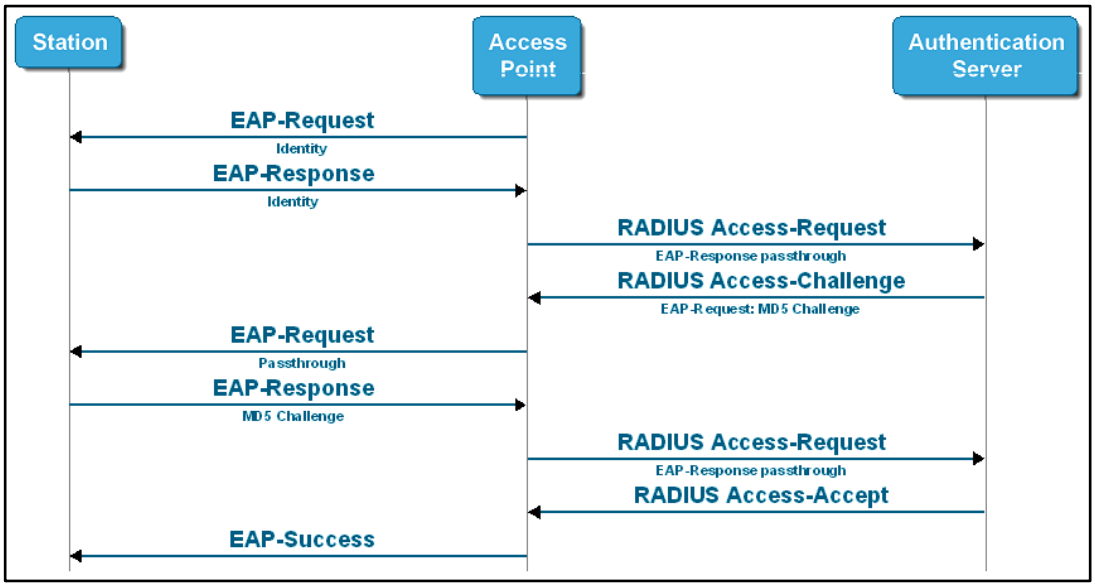
Once the authentication of the Station success, the Authentication Server sends a *Pairwise Master Key* (PMK) to the Station and to the Access Point, using the encryption provided by the authentication method used. Finally, the Station and the Access Point perform a *four-way handshake* in order to construct the PTK starting from the PMK, as described in *Section 2.1: "WPA2-PSK"*.

2.2.1 EAP-MD5 and EAP-MSCHAPv2

EAP-MD5[18] and EAP-MSCHAPv2[19] are two authentication protocol based on *hash functions.* The first one was defined in the original RFC of EAP, while the second was a proprietary solution developed by Microsoft in 2000.

Figure 7: "EAP-MD5 protocol" describes how the EAP-MD5 protocol works. The Station's identity is a string which is stored into the Authentication Server's database, together with a shared secret between the two entities (*password*). The Authentication Server asks the Station for the MD5 hash result of its password (*MD5 challenge*). If the sent challenge is the same as the one stored into the Authentication Server's database, the authentication is completed. EAP-MSCHAPv2 works in a similar way.

Figure 7: EAP-MD5 protocol



[16] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson and H. Levkowetz, Extensible authentication protocol (EAP). RFC 3748, 2004.

[17] S. Willens, A. C. Rubens, C. Rigney and W. A. Simpson, Remote authentication dial in user service (RADIUS). RFC 2865, 2000.

[18] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson and H. Levkowetz, Extensible authentication protocol (EAP). RFC 3748, 2004.

[19] G. Zorn, Microsoft PPP CHAP extensions, version 2. RFC 2759, 2000.

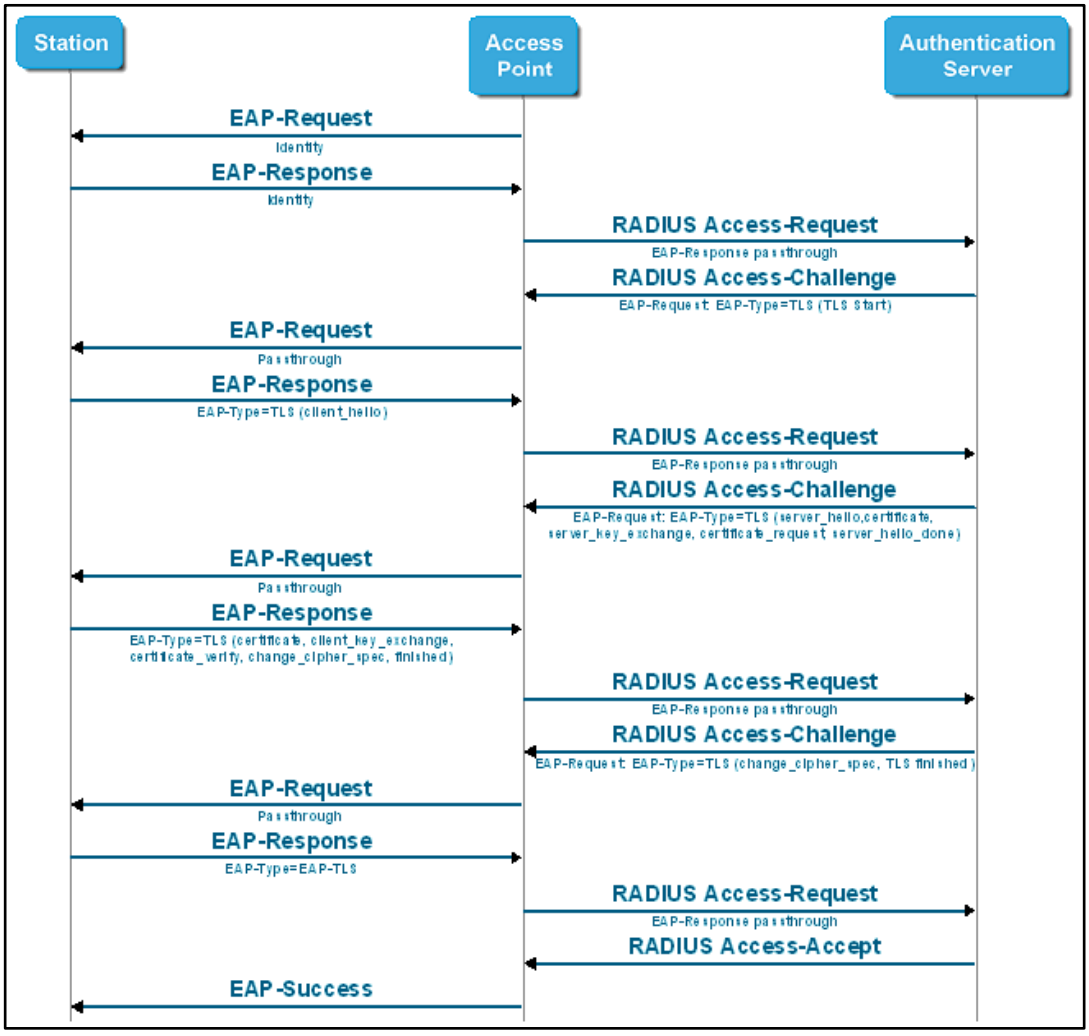
These two methods provide a very easy setup process of the device (the user just need to insert its username and its password), but are vulnerable to *dictionary* and *man-in-the middle attacks*[20].

Since their limitations on security, these two EAP methods are usually used as a *second authentication phase* after that a secure channel is established with *EAP-TTLS* and *PEAP* (see *Section 2.2.3: "EAP-TTLS and PEAP"*).

2.2.2 EAP-TLS

EAP-TLS (Transport Layer Security)[21] is the EAP authentication method which provides the highest security level and it is based on the TLS protocol (see *Section 1: "Transport layer security (TLS) protocol overview"*).

Figure 8: EAP-TLS protocol



TLS handshake messages are exchanged between the Station and the Authentication Server, via the Access Point. These messages are encapsulated into EAP messages

[20] H. Hwang, G. Jung, K. Sohn and S. Park, "A study on MITM (Man in the Middle) vulnerability in wireless network using 802.1 X and EAP," Information Science and Security, 2008. ICISS. International Conference on, pp. 164-170, 2008.

[21] D. Simon, B. Aboba and R. Hurst, The EAP-TLS authentication protocol. RFC 5216, 2008.

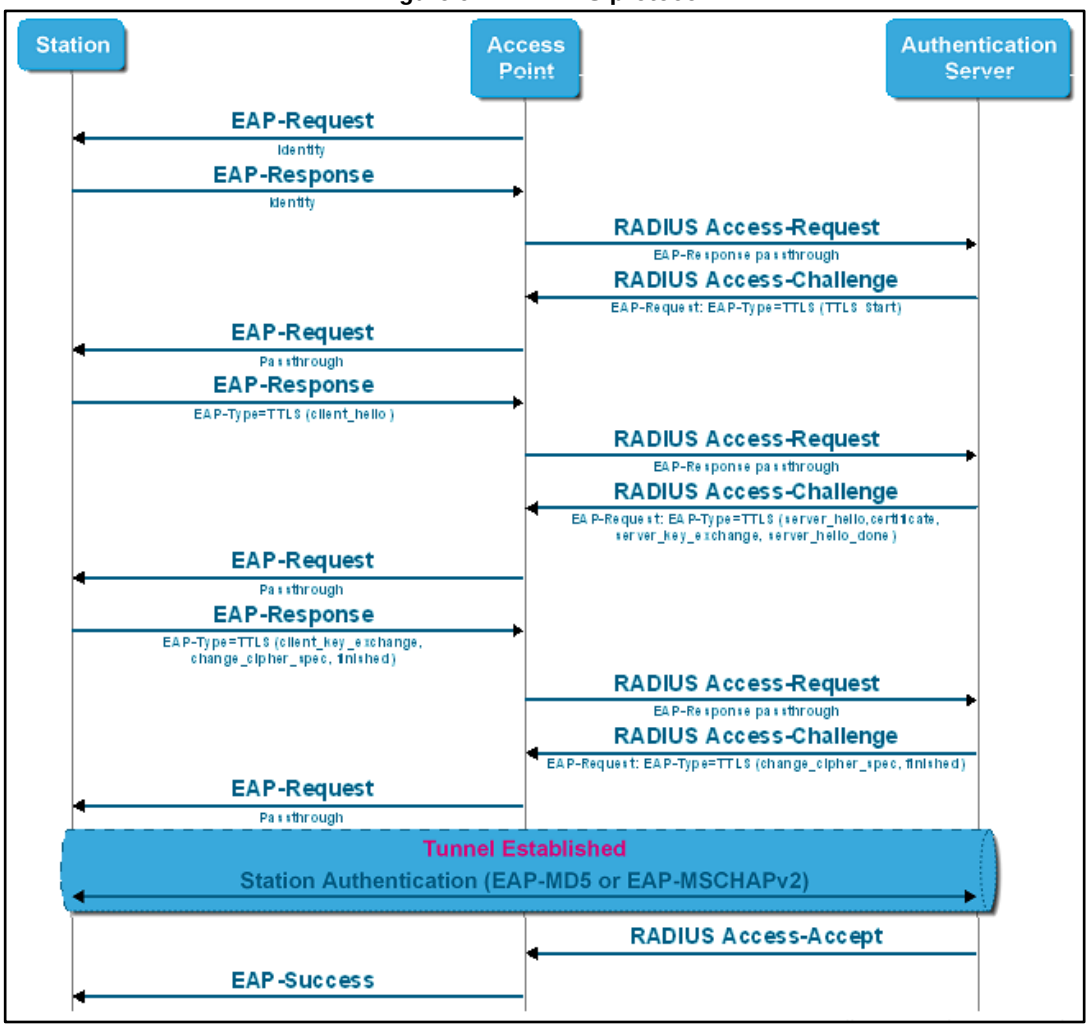
between the Station and the Access Point, and in RADIUS messages between the Access Point and the Authentication Server.

The authentication for both the Station and the Authentication Server is made via *X.509 certificates (mutual authentication).*

2.2.3 EAP-TTLS and PEAP

One of the major limitations of EAP-TLS is its usability: the user has to install on its device a private key and a certificate, and the Authentication Server needs to have installed the CA of the device.

Figure 9: EAP-TTLS protocol



In order to merge the usability of methods like EAP-MD5 or EAP-MSCHAPv2 with the security provided by EAP-TLS, methods based on *two phases* have been developed.

In the first phase a TLS handshake is made, with *one-way authentication* (i.e. only the server is authenticated, as explained in *Section 1.2: "Authentication and certificates"*). Once the key negotiation is done, the device creates a *secure channel* with the Authentication Server in order to start the *second phase* of the protocol, which consists in the authentication of the station via a method like EAP-MD5 or EAP-MSCHAPv2.

The two most known methods used for the first phase are EAP-TTLS (EAP-Tunneled Transport Layer Security)[22] and PEAP (Protected EAP) [23]. The last one is a proprietary solution developed by Microsoft and Cisco. *Figure 9: "EAP-TTLS protocol"* depicts an example of how EAP-TTLS works. PEAP works in a very similar way.

When combined with another method, the name of the resulting authentication method is the combination of the two name (e.g. EAP-TTLS/EAP-MD5 or PEAP/EAP-MSCHAPv2).

[22] P. Funk and S. Blake-Wilson, Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0). RFC 5281, 2008.

[23] V. Kamath, A. Palekar and M. Wodrich, Microsoft's PEAP version 0 (Implementation in Windows XP SP1), The Internet Society, 2002.

3 SPWF04S security

3.1 TLS on SPWF04S

SPWF04S modules integrate a lightweight TLS stack and a cryptographic library. The SPWF04S implements a TLS **client** with the features listed below:

- TLS v1.0, TLS v1.1 and TLS v1.2, with automatic downgrade of protocol version
- Server and client authentication
- Multiple Hashing Functions: SHA-1, SHA-256, SHA-384
- Block, Stream, and Authenticated Ciphers: AES (128 and 256, CBC and GCM), 3DES
- Digital signature algorithms: RSA (1024, 2048), ECDSA
- Key exchange: RSA (1024, 2048), DHE and ECDHE
- X.509 certificate support, either in DER or PEM format
- RSA private key support (PKCS #1), either in DER or PEM format

SPWF04S also implements a TLS **server** with the features listed below:

- TLS v1.0, TLS v1.1 and TLS v1.2, with automatic downgrade of protocol version
- Server and client authentication
- Multiple Hashing Functions: SHA-1, SHA-256, SHA-384
- Block, Stream, and Authenticated Ciphers: AES (128 and 256, CBC and GCM), 3DES
- Digital signature algorithms: RSA (1024, 2048), ECDSA
- Key exchange: RSA (1024, 2048), DHE, ECDHE
- X.509 certificate support, either in DER or PEM format
- RSA private key support (PKCS #1), either in DER or PEM format

The SPWF04S supports X.509v3 certificates with the following restrictions:

- None-Root certificates have to contain the *Authority Key Identifier* extension
- CA certificates have to contain the Basic Constraints and Subject Key Identifier extensions
- The certificates have to be smaller than 4 KBs.

3.1.1 Supported ciphers list

The following table shows supported cipher suites either in client mode than server mode

Table 2: TLS cipher suites

Cipher Suites	2 byte codes (hex format)	Version
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	0xC0, 0x23	TLS v1.2
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	0xC0, 0x09	TLS v1.2
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	0xC0, 0x24	TLS v1.2
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	0xC0, 0x0A	TLS v1.2
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	0xC0, 0x2B	TLS v1.2
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	0xC0, 0x2C	TLS v1.2
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	0xC0, 0x27	TLS v1.2
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	0xC0, 0x27	TLS v1.2
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	0xC0, 0x28	TLS v1.2
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	0xC0, 0x14	TLS v1.2

Cipher Suites	2 byte codes (hex format)	Version
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	0xC0, 0x2F	TLS v1.2
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	0xC0, 0x30	TLS v1.2
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	0x00, 0x67	TLS v1.2
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	0x00, 0x33	TLS v1.0/v1.1/v1.2
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	0x00, 0x9E	TLS v1.2
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	0x00, 0x16	TLS v1.2
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	0x00, 0x6B	TLS v1.2
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	0x00, 0x39	TLS v1.0/v1.1/v1.2
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	0x00, 0x9F	TLS v1.2
TLS_RSA_WITH_AES_128_CBC_SHA256	0x00, 0x3C	TLS v1.2
TLS_RSA_WITH_AES_128_CBC_SHA256	0x00, 0x9C	TLS v1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	0x00, 0x3D	TLS v1.2
TLS_RSA_WITH_AES_256_CBC_SHA	0x00, 0x35	TLS v1.0/v1.1/v1.2
TLS_RSA_WITH_AES_256_GCM_SHA384	0x00, 0x9D	TLS v1.2

3.1.2 Domain name check

When making a TLS connection the client requests a digital certificate from the server; once the server sends the certificate, the client examines it and compares the domain it was trying to connect to with the Common Name (CN) field included in the certificate. If a match is found, the connection proceeds as normal. If a match is not found, the user may be warned of the discrepancy and the connection may be aborted as the mismatch may indicate an attempted man-in-the-middle attack.

3.1.3 Authentication in client mode

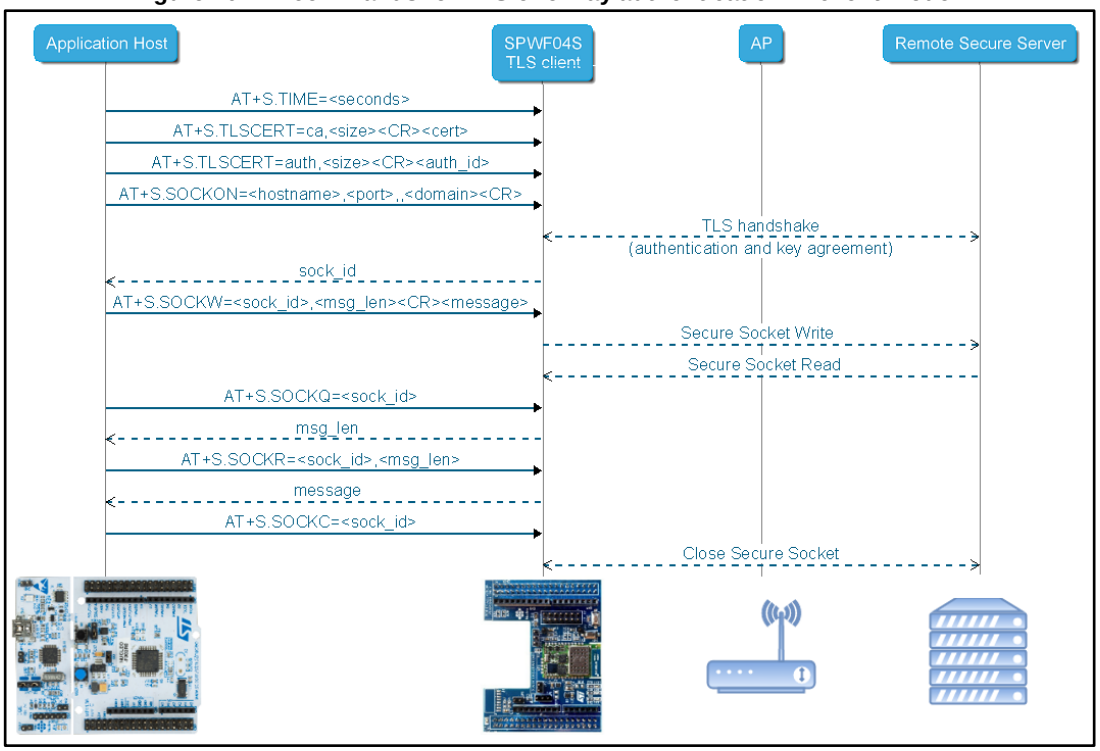
3.1.3.1 One-way authentication in client mode

In one-way authentication mode, the server sends its signed certificate to an unauthenticated client.

To verify the server certificate, the client:

1. **verifies the digital signature**: the issuing Root CA certificate must be preloaded onto the client (with the AT command AT+S.TLSCERT, see *Section 3.1.5: "Certificates and keys"*).
2. **checks that the date of the certificate is in range**: to check the date, the module reference time must be initialized after each module reset (using NTP if SPWF04S is connected to the Internet, or the AT command AT+S.TIME to set it manually, see *Section 3.1.5: "Certificates and keys"*); the time refers to UTC format expressed as the time in seconds since 1970-Jan-01.
3. **verifies the domain**: the domain passed to the client has to match the Common Name (CN) specified in the server certificate.

Figure 10: AT commands for TLS one-way authentication in client mode



If the verification of the server certificate succeeds, the connection proceeds as normal.

If verification fails (either for signature verification error, time or domain mismatch), then the client throws a warning message "AT-S.ERROR:74:Failed to open socket" and the connection is closed.

The maximum allowed size for any file uploaded to the module is 2.5 KB.

After the client is configured as in *Figure 10: "AT commands for TLS one-way authentication in client mode"*, the host can open a secure socket. The SPWF04S is able to manage two secure sockets at a time.

3.1.3.2 Mutual authentication in client mode

In mutual authentication mode, both parties (client and server) share their signed certificates.

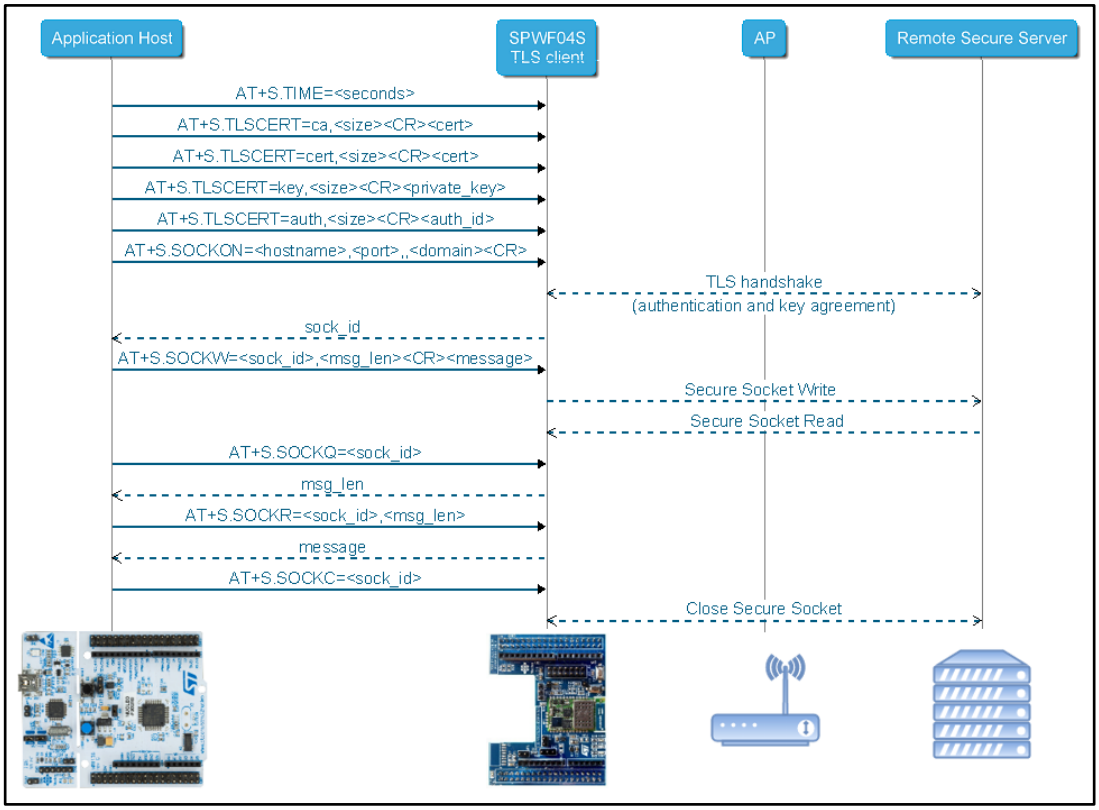
To verify server certificate, the client:

1. **verifies the digital signature**: the issuing Root CA certificate must be preloaded on the client (with the AT command AT+S.TLSCERT, see *Section 3.1.5: "Certificates and keys"*).
2. **checks that the date of the certificate is in range**: to check the date, the module reference time must be initialized after each module reset (using NTP if SPWF04S is connected to the Internet, or the AT command AT+S.TIME to set it manually, see *Section 3.1.5: "Certificates and keys"*); the time refers to UTC format expressed as the time in seconds since 1970-Jan-01.

3. **verifies the domain**: the domain passed to the client must match the name specified in the server certificate (Common Name or URL)
4. **sends its certificate to server**: the certificate and private key of the client must be loaded in advance into client (with the AT command AT+S.TLSCERT, see *Section 3.1.5: "Certificates and keys"*)

To verify client certificate, the server should have access to the issuing CA certificate (public or private).

Figure 11: AT commands for TLS mutual authentication in client mode



The server is in charge of the outcome of the client authentication process:

- **Option 1**: if client authentication succeeds, the handshake is completed and the connection proceeds as normal.
- **Option 2**: if client authentication fails but the server ignores this, then the handshake is completed and the connection proceeds as normal.
- **Option 3**: if client authentication fails and the server must discontinue communication, the handshake is interrupted, the connection is reset by server and the client throws a warning message “AT-S.ERROR:74:Failed to open socket”, and the connection is closed.

To solve connection errors, users therefore need to either change authentication mode to one-way mode or load the correct certificates and key.

The maximum allowed size for any file uploaded to the module is 2.5 KB.

After client is configured as in *Figure 11: "AT commands for TLS mutual authentication in client mode"*, the host can open a secure socket. SPWF04S is able to manage two secure sockets at a time.

3.1.4 Authentication in server mode

3.1.4.1 One-way authentication in server mode

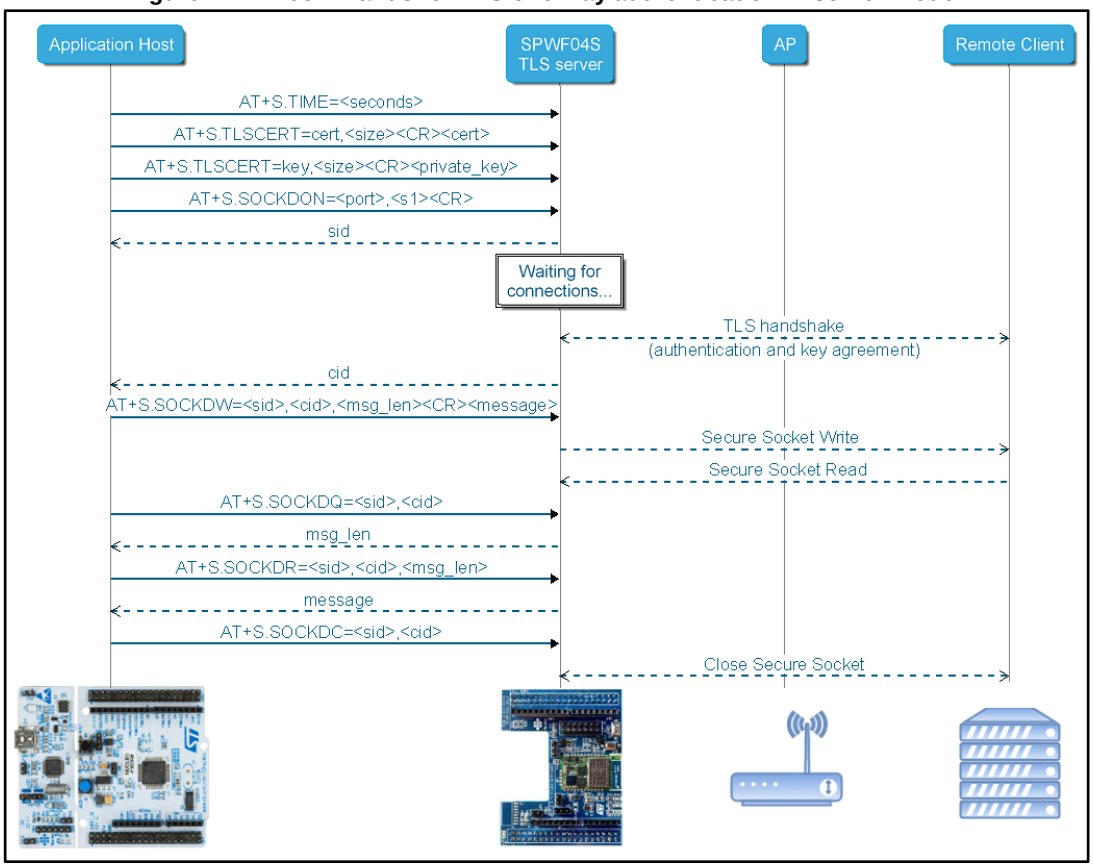
In one-way authentication mode, the server sends its signed certificate to an unauthenticated client. To perform this step, the server's certificate and private key have to be loaded in advance into the SPWF04S module (with the AT command AT+S.TLSCERT, see *Section 3.1.5: "Certificates and keys"*).

To verify the server's certificate, the client should have access to the issuing CA certificate (public or private).

The maximum allowed size for any file uploaded to the module is 2.5 KB.

After the SPWF04S module is configured as shown in *Figure 12: "AT commands for TLS one-way authentication in server mode"*, the host is able to accept requests by clients for open a secure socket with TLS.

Figure 12: AT commands for TLS one-way authentication in server mode



The `<sid>` value returned by the SPWF04S module to the Application Host after the AT+S.SOCKDON command is the identifier of the process accepting the socket request. SPWF04S is able to open up to two instances of these server processes. The `<cid>` value, instead, is returned after the handshake with the client and it is the identifier of the new opened socket between the `<sid>` process and the client. The maximum overall number of opened sockets in server mode is two.

3.1.4.2 Mutual authentication in server mode

In mutual authentication mode, both parties (client and server) share their signed certificates.

First, to be authenticated by the client, the server:

1. **sends its certificate to the client**: the server's certificate and private key must be preloaded on the SPWF04S module (with the AT command AT+S.TLSCERT, see *Section 3.1.5: "Certificates and keys"*)

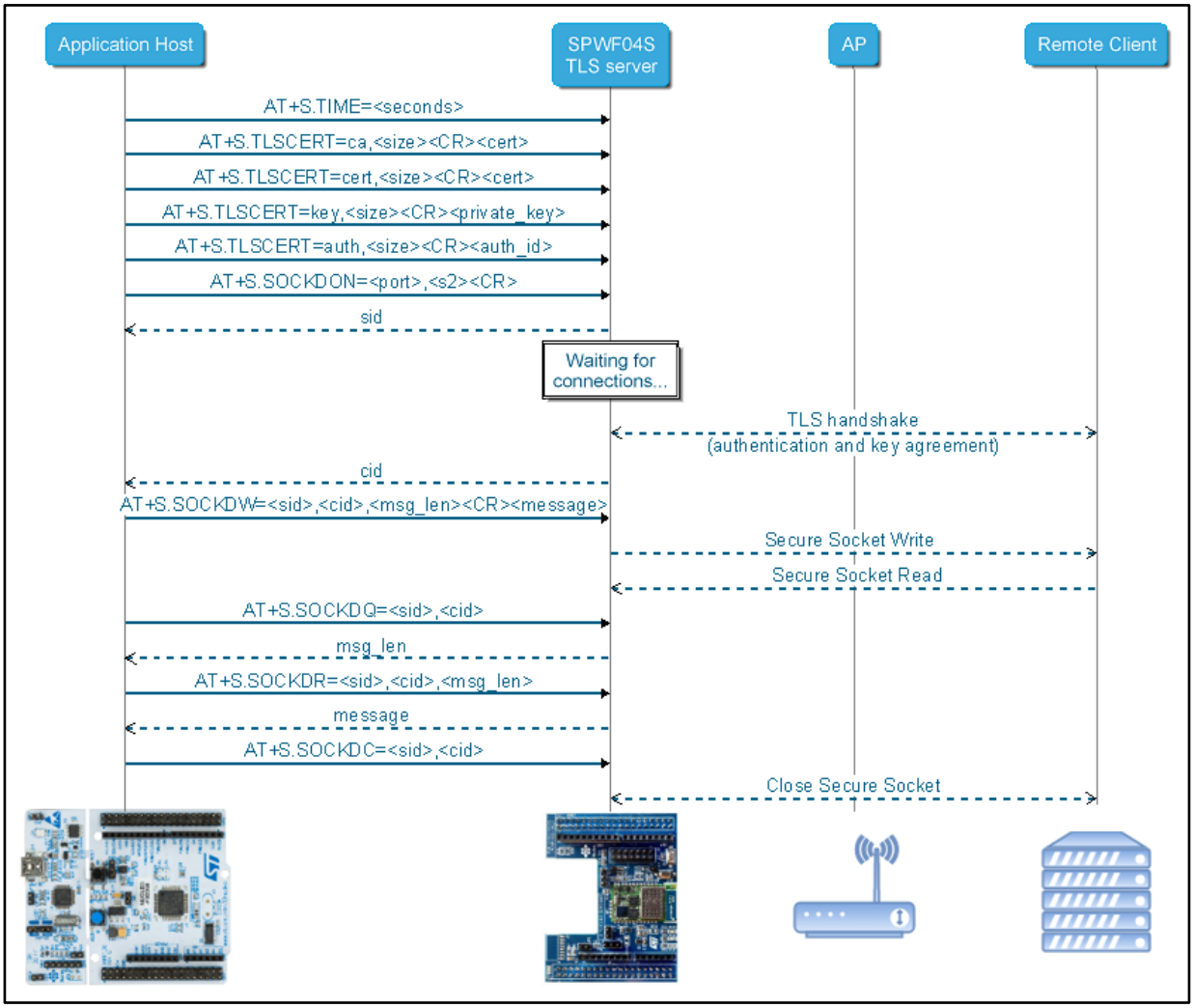
To verify server's certificate, the client should have access to the issuing CA certificate (public or private).

Then, in order to verify client certificate, the server:

1. **verifies the digital signatureof the client's certificate**: the issuing Root CA certificate must be preloaded on the module (with the AT command AT+S.TLSCERT, see *Section 3.1.5: "Certificates and keys"*).
2. **checks that the date of the certificate is in range**: to check the date, the module reference time must be initialized after each module reset (using NTP if SPWF04S is connected to the Internet, or the AT command AT+S.TIME to set it manually, see *Section 3.1.5: "Certificates and keys"*); the time refers to UTC format expressed as the time in seconds since 1970-Jan-01.

To verify client certificate, the server should have access to the issuing CA certificate (public or private).

Figure 13: AT commands for TLS mutual authentication in server mode



The server is in charge of the outcome of the client authentication process:

- **Option 1**: If client authentication succeeds, the handshake is completed and the connection proceeds as normal.

- **Option 2**: if the client authentication fails and the server must discontinue communication, then the handshake is interrupted and the connection is reset by server.

The maximum allowed size for any file uploaded to the module is 2.5 KB.

Once the SPWF04S module is configured as per *Figure 13: "AT commands for TLS mutual authentication in server mode"*, the host is able to accept requests by clients to open a secure socket with TLS.

The `<sid>` value returned by the SPWF04S module to the Application Host after the AT+S.SOCKDON command is the identifier of the process accepting the socket request. SPWF04S is able to open two instances of these server processes. The `<cid>` value, instead, is returned after the handshake with the client and it is the identifier of the new opened socket between the `<sid>` process and the client. The maximum overall number of opened sockets in server mode is two.

3.1.5 Certificates and keys

TLS uses X.509 certificates for authentication. Depending on the authentication method used, at least one of the following files could be needed:

- The Certification Authority (CA) for the server (or client) that the module has to connect with,
- The Authority key Id of the CA,
- The certificate for the SPWF04S module,
- The private key for the module,

Private keys protected with passwords are not supported by SPWF04S (see *Section "Remove password protecting private key"*).

The files can be loaded either in PEM format (text) or in DER format (binary). Depending on the PEM or DER format, the required AT commands for loading these files are the following:

```
AT+S.TLSCERT=content,2

AT+S.TLSCERT=ca,<size><CR><Peer CA>

AT+S.TLSCERT=cert,<size><CR><SPWF04S certificate>

AT+S.TLSCERT=key,<size><CR><SPWF04S private key>

AT+S.TLSCERT=auth,<size><CR><Peer CA Authority Key Id>
```

- `AT+S.TLSCERT=content,<1|2>` allows listing/removing the certificates and key stored into the Flash memory. The command, when the second parameter is set to 1, shows which files are loaded into the Flash memory. When the parameter is set to 2, it removes all the certificates and keys stored in the Flash memory. Since the Flash memory is non-volatile, a deletion of the certificates and keys previously stored is needed.
- `AT+S.TLSCERT=<ca|cert|key|auth>,<size><CR><data>` stores certificates or key files to Flash memory of the module. The first parameter is used to indicate when a root CA (ca), a client certificate (cert), a key file (key) or the Authority Key Id of the root CA (auth) is passed to the module. Please note that when DER format is used,

the Authority Key Id is automatically extracted during the Root CA parsing stage. As a result, the <auth> parameter is no longer needed. All these commands accept data after the <CR> character at the end of the command line. The host is expected to supply <size> of data as last parameter of the command line. The size values must be expressed in bytes.

The Authority Key Id must be loaded in binary format

In order to load certificates via a terminal (like Tera Term or others), the new line character must be set as “CR+LF”

In order to correctly use ECDSA private keys, it is needed to include the elliptic curve parameter together with the key, i.e. the file passed to the command AT+S.TLSCERT=key,<size> has to look like this:

```
-----BEGIN EC PARAMETERS-----
[...]
----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
[...]
-----END EC PRIVATE KEY-----
```

ECDSA private keys can be loaded only in PEM format, and not in PKCS#8, i.e. they have to look like this:

```
-----BEGIN EC PRIVATE KEY-----
[...]
-----END EC PRIVATE KEY-----
```

And not like this:

```
-----BEGIN PRIVATE KEY-----
[...]
-----END PRIVATE KEY----
```

In order to convert a PKCS#8 private key into a PEM one, please refer to Appendix A

Alternatively, certificates and private keys can be stored in the SPWF04S filesystem. In this case the certificates and keys must be saved in either ASN.1 DER format or Base64 encoded version of ASN.1 DER (i.e., PEM) format. The following convention must be used:

- The SPWF04S’s private key must be stored with the filename “tls.key”
- The SPWF04S’s certificate must be stored with the filename “tls.cert”
- Any Root CA must be stored with a filename corresponding to its Subject Key Identifier encoded as lower case hex string (e.g. if the Subject Key Identifier of the Root CA is C0:7A:98:68:8D:89:FB:AB:05:64:0C:11:7D:AA:7D:65:B8:CA:CC:4E, the corresponding file name will be “c07a98688d89fbab05640c117daa7d65b8cacc4e.ca”).

As TLS uses X.509 certificates for authentication, it is important to correctly set the current time in the module to determine whether the certificate is still valid or not. If the module is connected to the Internet, the SPWF04S module is able to automatically set the current time using the NTP protocol[24].

- `AT+S.TIME=<seconds>` is the alternative for setting the current time manually in seconds since 1970-01-01.
- `AT+S.TIME` can be used to check if the current time is correctly set.

Table 3: Commands to obtain the current time in seconds from a PC

Platform	Command
Unix shell	`date +%s`
Microsoft Powershell	`[int][double]::Parse((Get-Date (get-date).touniversaltime() -UFormat %s))`
Perl	`perl -e "print time"`

- `AT+S.SCFG=wifi_eap_skip_datechecks,1` forces the certificate validity check to be skipped
  - `wifi_eap_skip_datechecks` is set to 1 to skip and 0 (default) to not skip certificate validity

Bypassing the date check will provide less security in the authentication process as the SPWF04S module will not know if the Authentication Server certificates have expired.

3.1.6 Create a secure socket

Once the SPWF04S module is set up correctly as client, it is possible to open a secure socket with a server with the AT command:

- `AT+S.SOCKON=<hostname>,<port>,,<t|u|s|server domain>`
  - `<hostname>` is the IP address or the URI of the server
  - `<port>` is the port number where the socket must be open (specified by the server)
  - `<t|u|s|server domain>` specifies options for the type of socket used.
- `AT+S.SOCKL` lists the opened sockets.

Table 4: Options for AT+S.SOCKON command

Option	Result
t	Open a TCP socket (no TLS)
u	Open a UDP socket (no TLS)
s	Open a TLS socket using <hostname> as domain name
<server domain>	Open a TLS socket using <server domain> as domain name

The domain name is the one specified into the CN (Common Name) field of the Server's certificate. The CN usually coincides with the Server's URL, and in this case it's appropriate

[24] D. Mills, J. Martin, J. Burbank and W. Kasch, Network time protocol version 4: Protocol and algorithms specification. RFC 5905, 2010.

to use the letter s as third argument of the command. Otherwise, maybe for test purposes in which the Server doesn't have a URL, it is possible to put explicitly the domain name as third argument of the command.

Once the socket is correctly opened, the command returns the ID of the socket, and it is possible to write and/read from it.

- `AT+S.SOCKW=<id>,<len><CR><data>` writes to the socket
  - `<id>` is the ID of the socket where the data will be written
  - `<len>` is the byte length of the data to send (including the carriage return)
  - `<data>` is the actual data to send
- `AT+S.SOCKR=<id>,<len>` returns the received message from the socket
  - `<id>` is the ID of the socket from which data must be read
  - `<len>` is the byte length of received data (including the carriage return).
- `AT+S.SOCKQ=<id>` returns how many bytes of data is received from the socket
  - `<id>` is the ID of the socket.
- `AT+S.SOCKDON=<port>,<t|u|s1|s2>` is the command to listen for connection requests if SPWF04S acts as a server
  - `<port>` is the port number where the socket must be opened
  - `<t|u|s1|s2>` specifies options for the type of socket used.
- `AT+S.SOCKDL=<sid>` lists the opened sockets from a specific server.

Table 5: Options for AT+S.SOCKDON command

Option	Meaning
t	Open a TCP socket (no TLS)
u	Open a UDP socket (no TLS)
s1	Open a TLS socket with one-way authentication
s2	Open a TLS socket with mutual authentication

Once the socket is correctly opened, the command returns the ID of the socket and it is possible to write and/read from it.

- `AT+S.SOCKDW=<sid>,<cid>,<len><CR><data>` writes to the socket
  - `<sid>` is the ID of the server
  - `<cid>` is the ID of the socket where to write the data
  - `<len>` is the byte length of the data to send (including the carriage return)
  - `<data>` is the data to send.
- `AT+S.SOCKDR=<sid>,<cid>,<len>` returns the received message from the socket
  - `<sid>` is the ID of the server
  - `<cid>` is the ID of the socket from which data must be read
  - `<len>` is the length in byte of received data (including the carriage return).
- `AT+S.SOCKDQ=<sid>,<cid>` returns how many bytes of data is received from the socket
  - `<sid>` is the ID of the server
  - `<cid>` is the ID of the socket.

3.1.7 HTTPS and SMTPS

Once a secure socket is opened, the module is able to perform any application protocol using the AT commands AT+S.SOCKW and AT+S.SOCKR. The major drawback is that the user must insert into the module all the textual messages which the protocol specifies.

For example, in order to access *www.fakeURL.org/path_to_the_file/page.html* with the HTTP protocol, the needed messages are:

```
GET /path_to_the_file/page.html HTTP/1.0
Host: www.fakeURL.org
Connection: close
User-agent: Mozilla/4.0
Accept-language: it
```

The SPWF04S module provides commands for easily usage of two of the most used internet protocols when it acts as a client: HTTP[25] and SMTP[26]. These commands provide also options for using HTTPS[27] and security for SMTP, which rely on TLS. In these cases, the commands take care of opening the secure socket before sending the commands and, after that (and eventually after the server's response) to close the connection.

For HTTP, the two AT commands are AT+S.HTTPGET and AT+S.HTTPPOST for GET and POST messaging, respectively.

- `AT+S.HTTPGET=<host>,[<path>],[<port>],[<TLS>],[<usn>],[<pwd>],[<DownFile>],[<UpFile>]` is the GET synyax
  - `<host>` is the URL or the IP address of the server
  - `<path>` is the path of the requested html page inside the server
  - `<port>` is the port number in which the connection with the server is established
  - `<TLS>` is a variable specifying which level of security is provided (see *Table 6: "Options for AT+S.HTTPGET and AT+S.HTTPPOST command for the TLS option"*)
  - `<usn>,<pwd>` are the username and password in case the page needs authentication.
  - `<DownFile>` specifies a file where to save the HTTP response from the server.
  - `<UpFile>` is used for sending a file in order to make a custom http requests.

Table 6: Options for AT+S.HTTPGET and AT+S.HTTPPOST command for the TLS option

<TLS>	Meaning
0	HTTP without security
1	Automatically detect if to use HTTP or HTTPS
2	HTTPS (HTTP with TLS)

- `AT+S.HTTPPOST=<host>,[<path>],[<port>],[<TLS>],[<usn>],[<pwd>],[<DownFile>],[<UpFile>]` is the POST synyax
  - `<host>` is the URL or the IP address of the server
  - `<path>` is the path of the requested html page inside the server
  - `<port>` is the port number in which the connection with the server is established
  - `<TLS>` is a variable specifying which level of security is provided (see *Table 6: "Options for AT+S.HTTPGET and AT+S.HTTPPOST command for the TLS option"*)
  - `<usn>,<pwd>` are the username and password in case the page needs authentication.

[25] T. Berners Lee, R. Fielding and H. Frystyk, "Hypertext transfer protocol - HTTP/1.0. RFC 1945," 1996.

[26] J. Postel, "Simple Mail Transfer Protocol. RFC 821," 1982.

[27] E. Rescorla, "HTTP over TLS. RFC 2818," 2000.

  - `<DownFile>` specifies a file where to save the HTTP response from the server.
  - `<UpFile>` is used for sending a file in order to make a custom http requests.
- `AT+S.SMTP=<host>,[<port>],[<TLS>],[<usn>],[<pwd>],[<id>],<from>,<to>,,,<subj>,,<len><CR><body>` allows sending emails using the SMTP protocol
  - `<host>` is the URL or the IP address of the SMTP server.
  - `<port>` is the port number in which the connection with the server is established.
  - `<TLS>` is a variable specifying which level of security is provided (see *Table 7: "Options for AT+S.SMTP command for TLS option"*)
  - `<usn>,<pwd>` are the username and password for authentication to the SMTP server
  - `<id>` is the identification number used during starting the SMTP session.
  - `<from>` is the sender's email address.
  - `<to>` is the receiver's email address.
  - `<subj>` is the subject of the email.
  - `<len>` is the length of the body email.
  - `<body>` is the content of the email.

Table 7: Options for AT+S.SMTP command for TLS option

<TLS>	Meaning
0	No security
8	SMTP + STARTTLS if available, otherwise the mail is not sent
5	SMTPS on port 465 if available, otherwise SMTP + STARTTLS if available, otherwise no security
9	SMTPS on port 465 if available, otherwise SMTP + STARTTLS if available, otherwise the mail is not sent

The difference between SMTPS and SMTP+STARTTLS[28] is that in the first case the TLS session is opened before the SMTP protocol starts, while in the second case the TLS socket is opened inside the SMTP session. SMTPS has the major priority among the other options. The choice is based on the support or not of SMTPS and STARTTLS by the server.

The AT commands described here are usable when the SPWF04S acts as a client. Furthermore, an HTTP server runs in background into the module, in order to access to an html page for the remote configuration of the module. The server is also able to use TLS security.

In order to use AT+S.HTTPGET, AT+S.HTTPPOST and AT+S.SMTP with TLS, the Common Name (CN) reported in the server certificate must be exactly the same as that passed to the <host> parameter.

3.2 Wi-Fi protected access on SPWF04S

3.2.1 WPA2-PSK

The AT commands needed to establish a WPA2-PSK network connection are:

```
AT+S.WIFI=0
```

[28] P. Hoffman, "SMTP service extension for secure SMTP over Transport Layer Security," 2002.

```
AT+S.SCFG=wifi_mode,1
AT+S.SCFG=wifi_priv_mode,2
AT+S.SSIDTXT=<SSID>
AT+S.SCFG=wifi_wpa_psk_text,<WPA PSK password>
AT+S.WIFI=1
AT+S.WCFG
AT+S.RESET
```

- `AT+S.WIFI=<0|1>` disables and enables the Wi-Fi radio of the module. Disabling the radio is when modifying the internal parameters of SPWF04S.
- `AT+S.SCFG=<param>,<value>` sets the internal parameters of SPWF04S.
- `AT+S.SCFG=wifi_priv_mode,<type>` specifies the encryption protocol used by the Wi-Fi network (see *Table 8: "Encryption protocols"*).
- `AT+S.SSIDTXT=<SSID>` sets the SSID of the network in the module.
- `AT+S.SCFG=wifi_wpa_psk_text,<WPA PSK password>` sets the Pre-Shared Key of the Wi-Fi network in the module.
- `AT+S.SCFG=wifi_mode,1` configures the module in Station mode
- `AT+S.WCFG` saves the current configuration of the module
- `AT+S.RESET` and resets the module.

Table 8: Encryption protocols

<type>	Encryption protocol
0	None
1	WEP
2	WPA2-PSK
3	WPA2-Enterprise

3.2.1.1 WPS

- `AT+S.WPS=<0|1|2>` sets WPS-PIN or WPS-PBC configuration modes
- `0` is WPS-PBC mode
- `1` is WPS-PIN with PIN taken from variable `wifi_wps_pin` (7 digits; the 8-th digit is computed before the WPS procedure.
- `2` is WPS-PIN with random generated PIN

When using the WPS-PIN, the AT command returns the PIN which must be inserted into the AP configuration page.

3.2.2 WPA2-Enterprise

Refer to *Section "How to set up a WPA2-Enterprise network"* for how to configure a little WPA2-Enterprise network in order to test the features of the SPWF04S module.

The supported EAP authentication algorithms are:

- EAP-TLS
- EAP-TTLS/EAP-MD5
- EAP-TTLS/EAP-MSCHAPv2
- PEAP/EAP-MD5
- PEAP/EAP-MSCHAPv2

See *Section 2.2: "WPA2-Enterprise"* for further information.

Table 9: WPA2-Enterprise features on SPWF04S

	EAP-TLS	TTLS	PEAP
Client Authentication	Client certificate	EAP-MD5 or EAP-MSCHAPv2	EAP-MD5 or EAP-MSCHAPv2
Protocol Operations			
Basic protocol structure	Establish TLS session and validate certificates on both client and server	Two phases: (1) Establish TLS between client and TTLS server (2) Exchange attribute-value pairs between client and server	Two parts: (1) Establish TLS between client and PEAP server (2) Run EAP exchange over TLS tunnel
PKI and Certificate Processing			
Server Certificate	Required	Optional	Optional
Client Certificate	Required	Optional	Optional
Client and User Authentication			
Authentication direction	Mutual: Uses digital certificates both ways	Mutual: Certificate for server authentication, and tunneled method for client	Mutual: Certificate for server authentication, and protected EAP method for client
Protection of user identity exchange	No (Client certificate is exchanged in clear)	Yes; protected by TLS	Yes; protected by TLS

The AT commands designed to set up the SPWF04S module to establish a connection with a WPA2-Enterprise network are:

- `AT+S.SCFG=wifi_priv_mode,3`, where the parameter set to `3` means that the encryption protocol used by the network is WPA2-Enterprise, as described in *Table 8: "Encryption protocols"*
- `AT+S.SCFG=wifi_eap_type,<type>` specifies which authentication method is used to establish the WPA2-Enterprise connection.
  - `<type>` (see *Table 10: "EAP authentication methods"*).
- `AT+S.SCFG=wifi_eap_identity,<identity>` specifies the username by which the SPWF04S authenticates itself to the network.
- `AT+S.SCFG=wifi_eap_anon_identity,<anonymous_identity>` specifies the anonymous identity string used in the first phase of most of the EAP authentication methods.
- `AT+S.SCFG=wifi_eap_passwd,<password>` specifies the password by which the SPWF04S authenticates itself to the network.

Table 10: EAP authentication methods

<type>	EAP authentication method
0	EAP-TLS
1	TTLS-MD5
2	TTLS-MSCHAPv2
3	PEAP-MD5
4	PEAP-MSCHAPv2

Because WPA2-Enterprise authentication methods use X.509 certificates, it is important to correctly set the current time into the module.

- `AT+S.TIME=<seconds>` sets the current time (mandatory as NTP is not available before the connection to the Internet)
  - `<seconds>` represents the current time expressed in seconds since 1970-01-01.
- `AT+S.TIME` can be used to check if the current time is correctly set.

Table 3: "Commands to obtain the current time in seconds from a PC" lists some useful commands which can be used to get the current time from a PC.

3.2.2.1 Certificates and keys

SPWF04S can handle X.509v3 certificates and RSA public key algorithms with key length up to 4096 bits for the authentication phase in WPA2-Enterprise.

Depending on the EAP authentication method used, at least one of the following files may be required:

- The Certification Authority (CA) for the RADIUS server,
- The certificate for the SPWF04S module,
- The private key for the module.

Private keys protected with passwords are not supported by SPWF04S (see *Section "Remove password protecting private key"*).

The files can be loaded either in PEM format (text) or in DER format (binary). The required AT commands for loading these files are:

```
AT+S.WPAECERT=content,2

AT+S.WPAECERT=ca,<size><CR><RADIUS CA>

AT+S.WPAECERT=cert,<size><CR><SPWF04S certificate>

AT+S.WPAECERT=key,<size><CR><SPWF04S private key>
```

- `AT+S.WPAECERT=content,<1|2>` lists/removes the certificates and key stored in the Flash memory.
  - `<1|2>`: `1` - shows which files are loaded into the Flash memory `2` - removes all the certificates and keys stored in the Flash memory. As the Flash memory is non-volatile, the certificates and keys already stored must be deleted.

`AT+S.WPAECERT=<ca|cert|key>,<size><CR><data>` stores certificates or key files to Flash memory of the module. The first parameter is used to indicate when a root CA (ca), a client certificate (cert) or a key file (key) is passed to the module. This command accepts data after the <CR> character at the end of the command line. The host is expected to supply <size> of data as last parameter of the command line. The size values must be expressed in bytes.

3.2.2.2 EAP-TLS authentication

EAP-TLS authentication method requires that both the SPWF04S module and the RADIUS server authenticate themselves via X.509 certificates. To configure the module:

1. Use `AT+S.TIME` to set the current time

2. Load the RADIUS server CA, SPWF04S certificate and private key onto the SPWF04S Flash
3. Establish connection with the WPA2-Enterprise network using the AT commands below:

```
AT+S.WIFI=0
AT+S.SCFG=wifi_mode,1
AT+S.SCFG=wifi_priv_mode,3
AT+S.SCFG=wifi_eap_type,0
AT+S.SSIDTXT=<SSID>
AT+S.SCFG=wifi_eap_identity,<identity>
AT+S.WIFI=1
AT+S.WCFG
AT+S.RESET
```

- `AT+S.WIFI=<0|1>` disables and enables the Wi-Fi radio of the module; disabling the radio is necessary to modify the internal parameters of SPWF04S.
- `AT+S.SCFG=<param>,<value>` sets the internal parameters of SPWF04S.
  - `wifi_priv_mode` specifies the type of security the network provides; setting it to `3` it means that the network is a WPA2-Enterprise network.
  - `wifi_eap_type` specifies the authentication method used by the WPA2-Enterprise network. In this example, setting this value to `0` specifies that it used EAP-TLS.
  - `wifi_eap_identity` is used to identify the SPWF04S module on the network.
- `AT+S.SSIDTXT=<SSID>` sets the SSID of the network into the module.
- `AT+S.WCFG` and `AT+S.RESET` respectively save the current configuration of the module and reset the module.

3.2.2.3 EAP-TTLS authentication

In EAP-TTLS authentication methods, only the RADIUS server is authenticated on the module via X.509 certificate, while the client provides its credentials to the Enterprise network, which are stored in a database internal to the RADIUS server.

SPWF04S provides two authentication methods based on EAP-TTLS: TTLS-MD5 and TTLS-MSCHAPv2.

To configure the module:

1. Use `AT+S.TIME` to set the current time
2. Load the RADIUS server CA onto the SPWF04S Flash (optional but recommended)
3. Establish the connection to the WPA2-Enterprise network

EAP-*TTLS/EAP-MD5* authentication methods require:

```
AT+S.WIFI=0
AT+S.SCFG=wifi_mode,1
AT+S.SCFG=wifi_priv_mode,3
AT+S.SCFG=wifi_eap_type,1
AT+S.SSIDTXT=<SSID>
AT+S.SCFG=wifi_eap_identity,<identity>
AT+S.SCFG=wifi_eap_anon_identity,<anonymous_identity>
AT+S.SCFG=wifi_eap_passwd,<password>
AT+S.WIFI=1
```

```
AT+S.WCFG
AT+S.RESET
```

EAP-*TTLS/EAP-MSCHAPv2* authentication methods require:

```
AT+S.WIFI=0
AT+S.SCFG=wifi_mode,1
AT+S.SCFG=wifi_priv_mode,3
AT+S.SCFG=wifi_eap_type,2
AT+S.SSIDTXT=<SSID>
AT+S.SCFG=wifi_eap_identity,<identity>
AT+S.SCFG=wifi_eap_anon_identity,<anonymous_identity>
AT+S.SCFG=wifi_eap_passwd,<password>
AT+S.WIFI=1
AT+S.WCFG
AT+S.RESET
```

3.2.2.4 PEAP authentication

In PEAP authentication methods, only the RADIUS server is authenticated on the module via X.509 certificate, while the client provides its credentials to the Enterprise network, which are stored in a database internal to the RADIUS server.

SPWF04S provides two authentication methods based on PEAP: PEAP/EAP-MD5 and PEAP/EAP-MSCHAPv2.

To configure the module:

- Use `AT+S.TIME` to set the current time
- Load the RADIUS server CA onto the SPWF04S Flash (optional but recommended)
- Establish connection with the WPA2-Enterprise network

PEAP/EAP-MD5 authentication methods require:

```
AT+S.WIFI=0
AT+S.SCFG=wifi_mode,1
AT+S.SCFG=wifi_priv_mode,3
AT+S.SCFG=wifi_eap_type,3
AT+S.SSIDTXT=<SSID>
AT+S.SCFG=wifi_eap_identity,<identity>
AT+S.SCFG=wifi_eap_anon_identity,<anonymous_identity>
AT+S.SCFG=wifi_eap_passwd,<password>
AT+S.WIFI=1
AT+S.WCFG
AT+S.RESET
```

PEAP/EAP-MSCHAPv2 authentication methods require:

```
AT+S.WIFI=0
AT+S.SCFG=wifi_mode,1
AT+S.SCFG=wifi_priv_mode,3
AT+S.SCFG=wifi_eap_type,4
AT+S.SSIDTXT=<SSID>
```

```
AT+S.SCFG=wifi_eap_identity,<identity>
AT+S.SCFG=wifi_eap_anon_identity,<anonymous_identity>
AT+S.SCFG=wifi_eap_passwd,<password>
AT+S.WIFI=1
AT+S.WCFG
AT+S.RESET
```

3.2.3 Connection to the network

If the module is correctly configured, the following output is printed to the serial console after the reset command:

```
+WIND:1:Poweron:150722-5277210-SPWF04S
+WIND:13:Copyright (c) 2012-2016 STMicroelectronics, Inc.  All rights
Reserved:SPWF04SA/C
+WIND:0:Console active
+WIND:3:Watchdog Running:20
+WIND:32:WiFi Hardware Started
+WIND:21:WiFi Scanning
+WIND:35:WiFi Scan Complete:00
+WIND:19:WiFi Join:<AP MAC address>
+WIND:25:WiFi Association successful:<SSID>
+WIND:51:WPA Handshake Complete
+WIND:24:WiFi Up:0:<IPv4 address>
+WIND:24:WiFi Up::<IPv6 address>
```

The last two messages indicate the IPv4 and IPv6 addresses assigned to the SPWF04S module.

3.3 Secure FOTA (firmware over-the-air)

SPWF04S module can upgrade its software using Wi-Fi to download a new firmware image. This procedure is referred as *firmware over-the-air* (FOTA).

The Firmware's image is downloaded by the module from a server using the following AT command:

- `AT+S.FWUPDATE=e,<host>,<path>,<port>,<TLS>,<usn>,<pwd>`
  - `<host>` is the IP address of the server from which the module downloads the new firmware image.
  - `<path>` is the path inside the server used to download the new firmware image
  - `<port>` is the port number to connect with the server
  - `<TLS>` is a variable specifying which level of security is provided for the download of the firmware (see *Table 6: "Options for AT+S.HTTPGET and AT+S.HTTPPOST command for the TLS option"*).
  - `<usn>,<pwd>` are the username and password if the server requires client authentication.

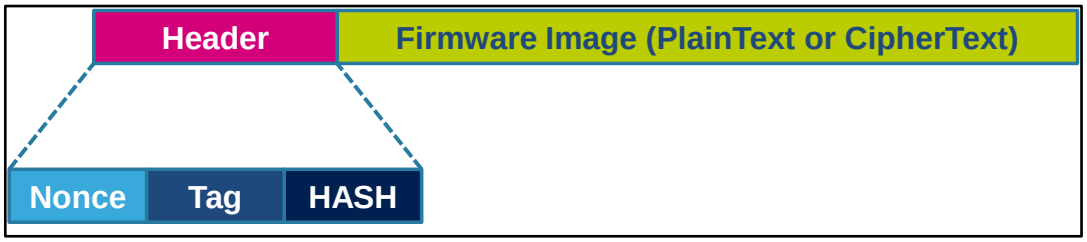
The downloaded file from the Server is a packet containing the firmware image and a *header,* as shown in *Figure 14: "FOTA packet structure"*. The Firmware can be

downloaded in *plaintext* or encrypted with *AES-128-GCM* symmetric cipher[29]. When encryption is used, the process is called *secure FOTA.*

The header contains the following fields:

- **Nonce and Tag**: parameters used for the encryption with AES-128-GCM. If the firmware image is not encrypted, these values are set to 0.
- **HASH**: is the result of the SHA-256[30] hash function on the Firmware image in plaintext. This field is used to check for correct firmware installation (integrity check).

Figure 14: FOTA packet structure



In order to perform Secure FOTA, the SPWF04S module needs a *symmetric key* stored in the `aes128_key` variable in hexadecimal notation, which must also be on the server to perform the decryption of the received firmware image.

Once the download has finished, the SPWF04S module resets to install the new firmware. Before installing the firmware, there is an integrity/authenticity check of the downloaded file:

- for FOTA without security, the SHA-256 hash of the firmware image is computed and compared with the HASH field of the header.
- for a secure FOTA, the firmware image is decrypted in RAM, then the SHA-256 hash of the decrypted image is performed and compared with the HASH field of the header.

If the integrity/authenticity check succeeds, then the installation is performed. A subsequent integrity check then verifies that the installation is completed successfully.

3.3.1 How to generate a FOTA

Starting from a firmware image, a utility for the generation of the FOTA packet to send to the module is provided[31]. The package provides the source files to be compiled and it needs the OpenSSL library in order to work.

In order to compile the utility under Unix, open a terminal, go into the directory containing the source code and type make. This will generate the executable named `create_ota`.

- `./create_ota infile outfile [keyfile]` is the syntax
  - `infile` is the Firmware image in the Intel HEX format[32]. The file size must be a multiple of 4 bytes, otherwise an error is returned.
  - `outfile` is the FOTA packet generated by the utility.
  - `keyfile` is the 128 bit AES key used for the encryption of the firmware image. If this is omitted, the resulting FOTA packet is not encrypted and authenticated.

[29] D. Mc Grew and J. Viega, "The Galois/counter mode of operation (GCM)," Submission to NIST. http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf, 2004.

[30] National Institute of Standards and Technology, "Secure Hash Standard (SHS). FIPS PUB 180-4," FEDERAL INFORMATION PROCESSING STANDARDS, 2012.

[31] The package is available from ST on request

[32] https://en.wikipedia.org/wiki/Intel_HEX

- `openssl rand -out key.bin 16` generates the 128-bit AES key with OpenSSL

The same key must also be loaded in the SPWF04S module or it won't be able to decrypt the received packet; the key must be loaded in HEX.

4 SPWF04S TLS tutorial

4.1 Example: TLS client with mutual authentication

The first example implements a TLS client that seeks to open a secure socket with a server using mutual authentication.

Before running the following examples, the SPWF04S (client) must be connected to the Wi-Fi LAN as described in 3.2 and OpenSSL must be installed on the PC (acting as a server).

For testing purposes, OpenSSL1.0.2f was used as the TLS server.

Documentation and installation instructions are available on the OpenSSL website.

The IP addresses of the client and the server are automatically assigned by the network router.

- `openssl s_server -cert <path/of/the/server/cert.pem> -key <path/of/the/server/private_key.pem> -CAfile <path/of/the/ca.pem> -Verify 2 -verify_return_error -accept <port>` starts the server from a terminal window

The accept parameter specifies the TCP port to listen to for connections, the default is 4433.

1. `AT+S.TLSCERT=content,2` should be run first to clean the Flash memory.
2. Then use:

```
AT+S.TIME=<seconds>
AT+S.TLSCERT=ca,<size><CR><data>
AT+S.TLSCERT=cert,<size><CR><data>
AT+S.TLSCERT=key,<size><CR><data>
AT+S.TLSCERT=auth,<size><CR><data>
AT+S.SOCKON=<hostname>,<port>,,<server domain>
```

`<seconds>` is the current time expressed in seconds since 1970-01-01; if an Internet connection is available, the current time is automatically set by NTP and AT command is not needed.

1. If the TLS handshake is successful:
   - `AT+S.SOCKON` returns the ID of the socket
2. Then it will be possible to write data to and read from the secure socket using:
   - `AT+S.SOCKW`
   - `AT+S.SOCKQ`
   - `AT+S.SOCKR`

4.2 Example: TLS client with one-way authentication

This example implements a TLS client that seeks to open a secure socket with a server using one-way authentication.

Before running the following examples, the SPWF04S (client) must be connected to the Wi-Fi LAN as described in 3.2 and OpenSSL must be installed on the PC (acting as a server).

For testing purposes, OpenSSL1.0.2f was used as the TLS server.

Documentation and installation instructions are available on the OpenSSL website.

The IP addresses of the client and the server are automatically assigned by the network router.

- `openssl s_server -cert <path/of/the/server/cert.pem> -key <path/of/the/server/private_key.pem> -accept <port>` starts the server from a terminal window.

The accept parameter specifies the TCP port to listen to for connections, the default is 4433.

1. `AT+S.TLSCERT=content,2` should be run first to clean the Flash memory.
2. Then use:

```
AT+S.TIME=<seconds>

AT+S.TLSCERT=ca,<size><CR><data>

AT+S.TLSCERT=auth,<size><CR><data>

AT+S.SOCKON=<hostname>,<port>,,<server domain>
```

`<seconds>` is the current time expressed in seconds since 1970-01-01; if an Internet connection is available, the current time is automatically set by NTP and AT command is not needed.

1. If the TLS handshake is successful:
   – `AT+S.SOCKON` returns the ID of the socket
2. Then it will be possible to write data to and read from the secure socket using:
   – `AT+S.SOCKW`
   – `AT+S.SOCKQ`
   – `AT+S.SOCKR`

4.3 Example: TLS server with mutual authentication

This example implements a TLS server which is able to accept connections from clients in mutual authentication.

Before running the following examples, the SPWF04S (client) must be connected to the Wi-Fi LAN as described in 3.2 and OpenSSL must be installed on the PC (acting as a server).

For testing purposes, OpenSSL1.0.2f was used as the TLS server.

Documentation and installation instructions are available on the OpenSSL website.

The IP addresses of the client and the server are automatically assigned by the network router.

1. `AT+S.TLSCERT=content,2` should be run first to clean the Flash memory.
2. Then use:

```
AT+S.TIME=<seconds>

AT+S.TLSCERT=ca,<size><CR><data>
```

```
AT+S.TLSCERT=cert,<size><CR><data>
AT+S.TLSCERT=key,<size><CR><data>
AT+S.TLSCERT=auth,<size><CR><data>
AT+S.SOCKDON=<port>,<s2>
```

`<seconds>` is the current time expressed in seconds since 1970-01-01; if an Internet connection is available, the current time is automatically set by NTP and AT command is not needed.

- `AT+S.SOCKDON` returns the `<sid>` identifier of the started server process, which will be waiting for connection requests by clients.
- `openssl s_client -connect <server IP/URL>:<port> -CAfile <path/of/the/server/CA.pem> -cert <path/of/the/client/cert.pem> -key <path/of/the/client/private_key.pem>` starts the server from a terminal window.

1. This command starts a TLS handshake with the SPWF04S module (specified by its IP address or URL) in the port specified previously.
2. If the TLS handshake ends successfully, the process ends by returning the ID of the socket `<cid>`
3. Then it will be possible to write data to and read from the secure socket using:

- `AT+S.SOCKDW`
- `AT+S.SOCKDQ`
- `AT+S.SOCKDR`

4.4 Example: TLS server with one-way authentication

This example implements a TLS server which is able to accept connections from clients in one-way authentication.

Before running the following examples, the SPWF04S (client) must be connected to the Wi-Fi LAN as described in 3.2 and OpenSSL must be installed on the PC (acting as a server).

For testing purposes, OpenSSL1.0.2f was used as the TLS server.

Documentation and installation instructions are available on the OpenSSL website.

The IP addresses of the client and the server are automatically assigned by the network router.

1. `AT+S.TLSCERT=content,2` should be run first to clean the Flash memory.
2. Then use:

```
AT+S.TLSCERT=cert,<size><CR><data>
AT+S.TLSCERT=key,<size><CR><data>
AT+S.SOCKDON=<port>,<s1>
```

- `AT+S.SOCKDON` returns the `<sid>` identifier of the started server process, which will be waiting for connection requests by clients.
- `openssl s_client -connect <server IP/URL>:<port> -CAfile <path/of/the/server/CA.pem>` starts the server from a terminal window.

1. This command starts a TLS handshake with the SPWF04S module (specified by its IP address or URL) in the port specified previously.
2. If the TLS handshake ends successfully, the process ends by returning the ID of the socket `<cid>`
3. Then it will be possible to write data to and read from the secure socket using:

- `AT+S.SOCDW`
- `AT+S.SOCDQ`
- `AT+S.SOCDR`

4.5 Example: Amazon with one-way authentication

This example shows how to set up the SPWF04S module in order to open a secure socket with a real world web server: *www.amazon.com*.

Before running this example, the SPWF04S (server) must be connected to the Wi-Fi LAN as described in 3.2 and the network must have access to the Internet.

1. First obtain the root CA certificate of the web server (see *Section "Extract Root CAs from web browser"*)
2. Then use:

```
AT+S.TLSCERT=ca,<size><CR><data>

AT+S.TLSCERT=auth,<size><CR><data>

AT+S.SOCKON=www.amazon.com,443,,s
```

As the SPWF04S is attached to the Internet, the current time is automatically set by NTP and there is no need to manually set it with the AT command.

1. If the TLS handshake is successful:
   - `AT+S.SOCKON` returns the ID of the socket
2. Then it will be possible to write data to and read from the secure socket using:
   - `AT+S.SOCKW`
   - `AT+S.SOCKQ`
   - `AT+S.SOCKR`

4.6 Example: HTTPS with Amazon

This example illustrates how to use `AT+S.HTTPGET` and `AT+S.HTTPPOST` to exchange HTTPS messages with a secure web server (*www.amazon.com*).

1. First obtain the root CA certificate of the web server (see *Section "Extract Root CAs from web browser"*)

2. To send a GET command, use:

```
AT+S.TLSCERT=ca,<size><CR><data>
AT+S.TLSCERT=auth,<size><CR><data>
AT+S.HTTPGET=www.amazon.com,,443,2,,,,
```

3. To send a POST command, use:

```
AT+S.TLSCERT=ca,<size><CR><data>
AT+S.TLSCERT=auth,<size><CR><data
AT+S.HTTPPOST=www.amazon.com,,443,2,,,[<DownFile>],[<UpFile>]
```

Once the files are loaded into the Flash of the module, there is no need to reload them again. So it is possible to just use `AT+S.HTTPGET` or `AT+S.HTTPPOST` .

When the <TLS> parameter is set with the value 2, as in this example, the TLS handshake is done before the data is sent to the server.

If the socket creation is successful, the two commands return the response of the server (e.g., a requested HTML page)

4.7 Example: SMTPS with Gmail

This example describes how to send an email with Gmail using the SPWF04S module; a Gmail account is needed.

1. First obtain the root CA certificate of the web server (see *Section "Extract Root CAs from web browser"*)
2. Then use:

```
AT+S.TLSCERT=ca,<size><CR><data>

AT+S.TLSCERT=auth,<size><CR><data>

AT+S.SMTP=smtp.gmail.com,465,9,<username>,<password>,<id>,<username>@gmail.com,<rece
iver_email_address>,,,<subject of mail>,,<body_lenght><CR><body_msg>
```

When the <TLS> parameter is set with the value 9, as in this example, the TLS handshake is done before the SMTP protocol is performed.

An email will be sent to the address specified in the `<receiver_email_address>` parameter.

Appendix A OpenSSL - useful commands

OpenSSL is an open-source implementation for PC platforms (Win, *nix, Mac) of the SSL/TLS protocols providing both client and server functionalities. The core library, written in the C programming language, implements the basic cryptographic functions and provides various utility functions.

For testing purposes, OpenSSL1.0.2f was used as the TLS Server.

Find documentation and installation instructions on the OpenSSL website (www.openssl.org).

RSA certificates and private keys generation

Generate key pair and a self-signed certificate for the CA (trusted certificate):

```
openssl genpkey -out ca_key.pem -outform PEM -algorithm rsa –pkeyopt
rsa_keygen_bits:1024

openssl req -new -key ca_key.pem -days 6500 -set_serial 1111 –subj
“/C=IT/ST=Lombardia/L=Milan/O=STM/OU=R&D/CN=CA domain” -out ca_cert.pem -x509
```

Generate server certificate/key pair:

```
openssl genpkey -out server key.pem -outform PEM -algorithm rsa –pkeyopt
rsa_keygen_bits:1024

openssl req -new -key server key.pem -days 6500 -set serial 2222 –subj
“/C=IT/ST=Lombardia/L=Milan/O=STM/OU=R&D/CN=server domain” –out server_cert_req.pem

openssl ca –in server_cert_req.pem -out server_cert.pem -days 6500 –keyfile
ca_key.pem –cert ca_cert.pem -notext -batch
```

Generate client certificate/key pair:

```
openssl genpkey -out client_key.pem -outform PEM -algorithm rsa –pkeyopt
rsa_keygen_bits:1024

openssl req -new -key client key.pem -days 6500 -set serial 3333 –subj
“/C=IT/ST=Lombardia/L=Milan/O=STM/OU=R&D/CN=client domain” –out client_cert_req.pem

openssl ca –in client_cert_req.pem -out client_cert.pem -days 6500 –keyfile
ca_key.pem –cert ca_cert.pem -notext –batch
```

ECDSA certificates and private keys generation

Generate key pair and a self-signed certificate for the CA (trusted certificate):

```
. openssl ecparam -out ca_key.pem -name prime192v1 -genkey
```

Note: the above command will automatically include the elliptic curve parameters as requested in Section 3.1.5

```
· openssl req -new -key ca_key.pem -days 6500 -set_serial 1111 -subj
“/C=IT/ST=Lombardia/L=Milan/O=STM/OU=R&D/CN=CA domain” -out ca_cert.pem -x509
```

Generate server certificate/key pair

```
· openssl ecparam -out server key.pem -name prime192v1 -genkey
· openssl req -new -key server key.pem -days 6500 -set serial 2222 -subj
“/C=IT/ST=Lombardia/L=Milan/O=STM/OU=R&D/CN=server domain” -out server cert req.pem
· openssl ca -in server cert req.pem -out server cert.pem -days 6500 -keyfile
ca_key.pem -cert ca_cert.pem -notext -batch
```

Generate client certificate/key pair:

```
· openssl ecparam -out client_key.pem -name prime192v1 -genkey
· openssl req -new -key client_key.pem -days 6500 -set_serial 3333 -subj
“/C=IT/ST=Lombardia/L=Milan/O=STM/OU=R&D/CN=client domain” -out client_cert_req.pem
```

```
· openssl ca -in client cert req.pem -out client cert.pem -days 6500 -keyfile
ca_key.pem -cert ca_cert.pem -notext -batch
**************************grey end**************************
```

If there is the need to convert a private key from PKCS#8 to PEM encoding, use the following command:

```
· openssl ec -in old_key.pem -out new_key.pem
```

Certificates and keys inspection

OpenSSL provides some commands useful for certificates and keys inspection, in order to find, for instance, the DN (Distinguished Name) of a certificate or the cryptographic algorithm used for a private key generation.

Following examples consider usage of RSA private keys.

To view the structure of the private key file `key.pem`:

```
openssl rsa –text –in key.pem
```

To view the structure of the Certificate Signing Request (CSR) file `cert_req.pem`:

```
openssl req –text –in cert_req.pem –noout
```

To view the structure of the certificate file `cert.pem`:

```
openssl x509 –text –in cert.pem –noout
```

Remove password protecting private key

When using certificates and private keys generated by third parties, it is possible that the private key is encrypted with a password.

To use the protected private key inside the module, the protection can be removed in OpenSSL (given the password) thus:

```
openssl rsa –in private_key_encrypted.pem –out
client_key_no_encrypted.pem
```

After the command is given, the user will be asked to enter the password.

The new generated file will be the same private key given as input, without the encryption.

Extract private key and certificate from a .p12 file

Sometimes the private key and the certificate are provided in a single file with extension *.p12*, which uses the *archive file format PCKS #12*[a].

To convert this file in two separated files, one with the private key and one with the certificate:

```
openssl pkcs12 -in file.p12 -out newfile_cert.pem -clcerts -nokeys

openssl pkcs12 -in file.p12 -out newfile_key.pem -nocerts -nodes
```

Convert certificates and keys from PEM to DER, and vice-versa

In order to convert a private key from PEM format to DER, it is possible to use the following openssl command:

```
openssl rsa –in key.pem –inform PEM –out key.der –outform DER
```

[a] S. Parkinson, K. Moriarty, M. Scott, A. Rusch and M. Nystrom, PKCS# 12: Personal information exchange syntax v. 1.1. RFC 7292, 2014.

On the contrary, in order to convert a private key from DER format to PEM, it is possible to use the following openssl command:

```
openssl rsa –in key.der –inform DER –out key.pem –outform PEM
```

In order to convert an X509 certificate from PEM format to DER, it is possible to use the following openssl command:

```
openssl x509 –in cert.pem –inform PEM –out cert.der –outform DER
```

On the contrary, in order to convert an X509 certificate from DER format to PEM, it is possible to use the following openssl command:

```
openssl x509 –in cert.der –inform DER –out cert.pem –outform PEM
```

Appendix B Extract Root CAs from web browser

In order to start a TLS session with web servers like Google or Amazon, their root CAs must be installed into SPWF04S module.

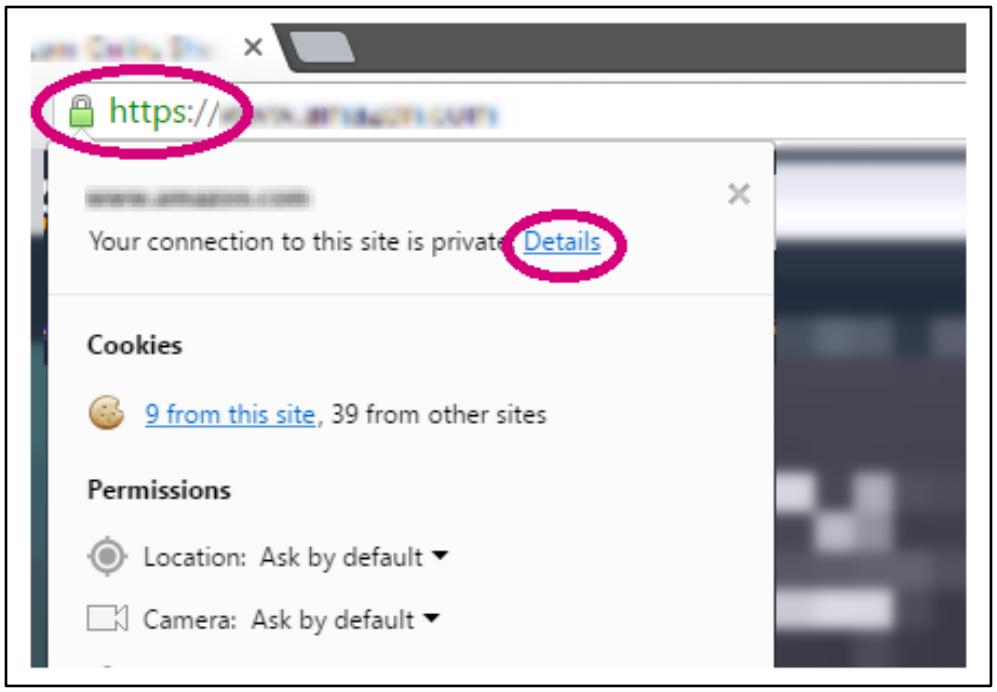
These certificates are included in most common web browsers and it is possible to save them in a file in DER or PEM format.

The following example describes how to extract a root CA from Google Chrome™ on a Microsoft Windows® platform.

When an HTTPS web page is opened, a green lock is displayed next to the URL, as shown in the following figure.

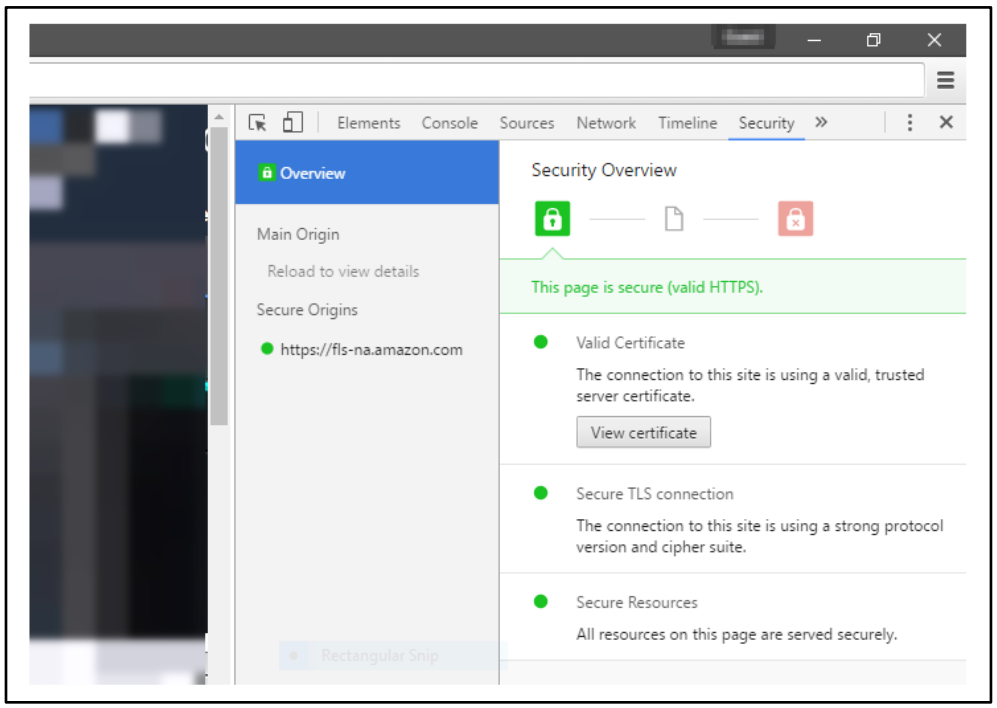
1 Click on the green lock to open a pop-up window showing some information regarding the webpage.

Figure 15: HTTPS in Google Chrome



2 Click *Details* to show the window regarding the security of the webpage.

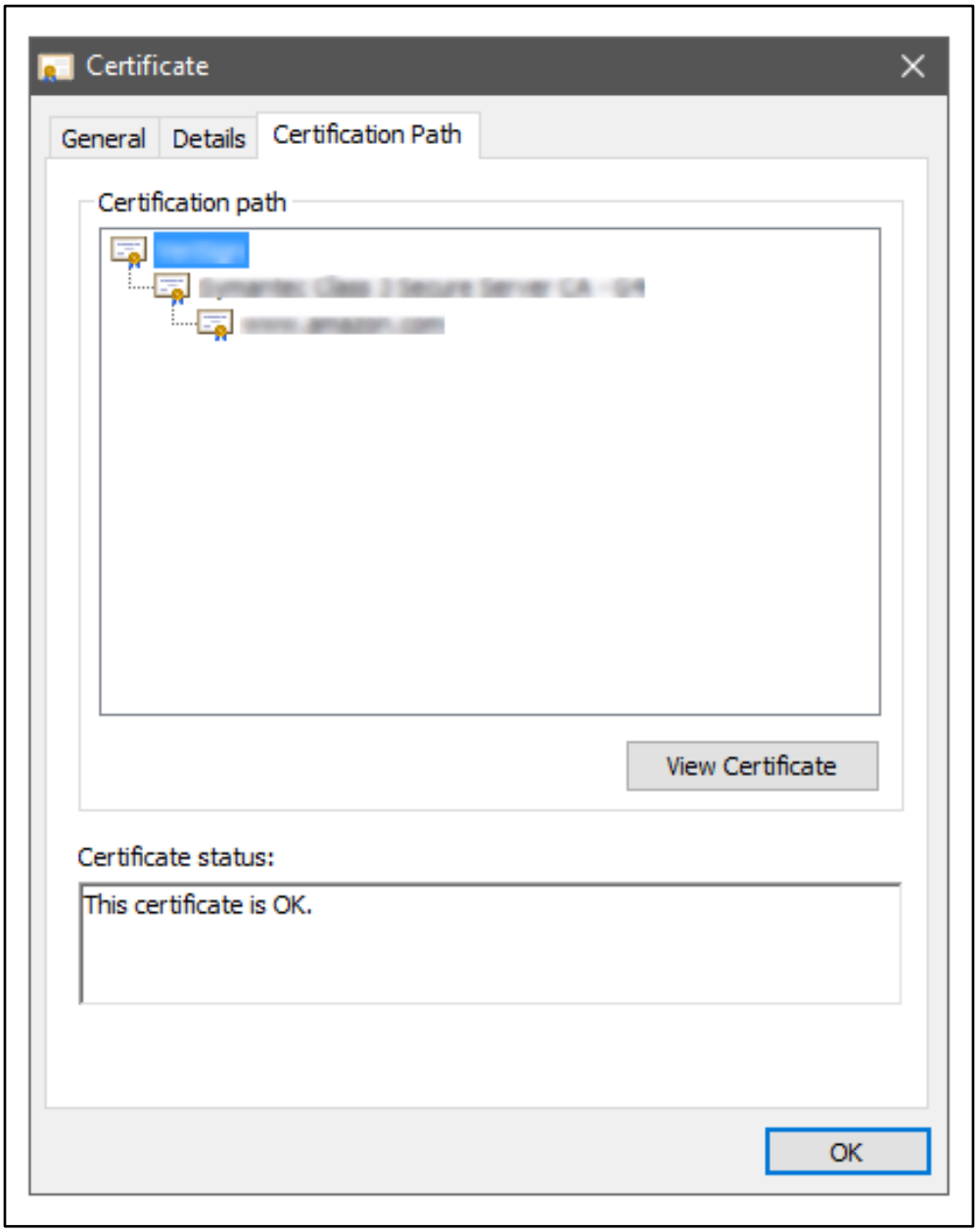
Figure 16: Security Overview window in Google Chrome



3 Click *View Certificate* to see the webpage certificate content.

4 Click on *Certification Path* to view the entire path of validation, up to the Root CA, which is the highest icon on the chain.

Figure 17: Certificate window – Certification path

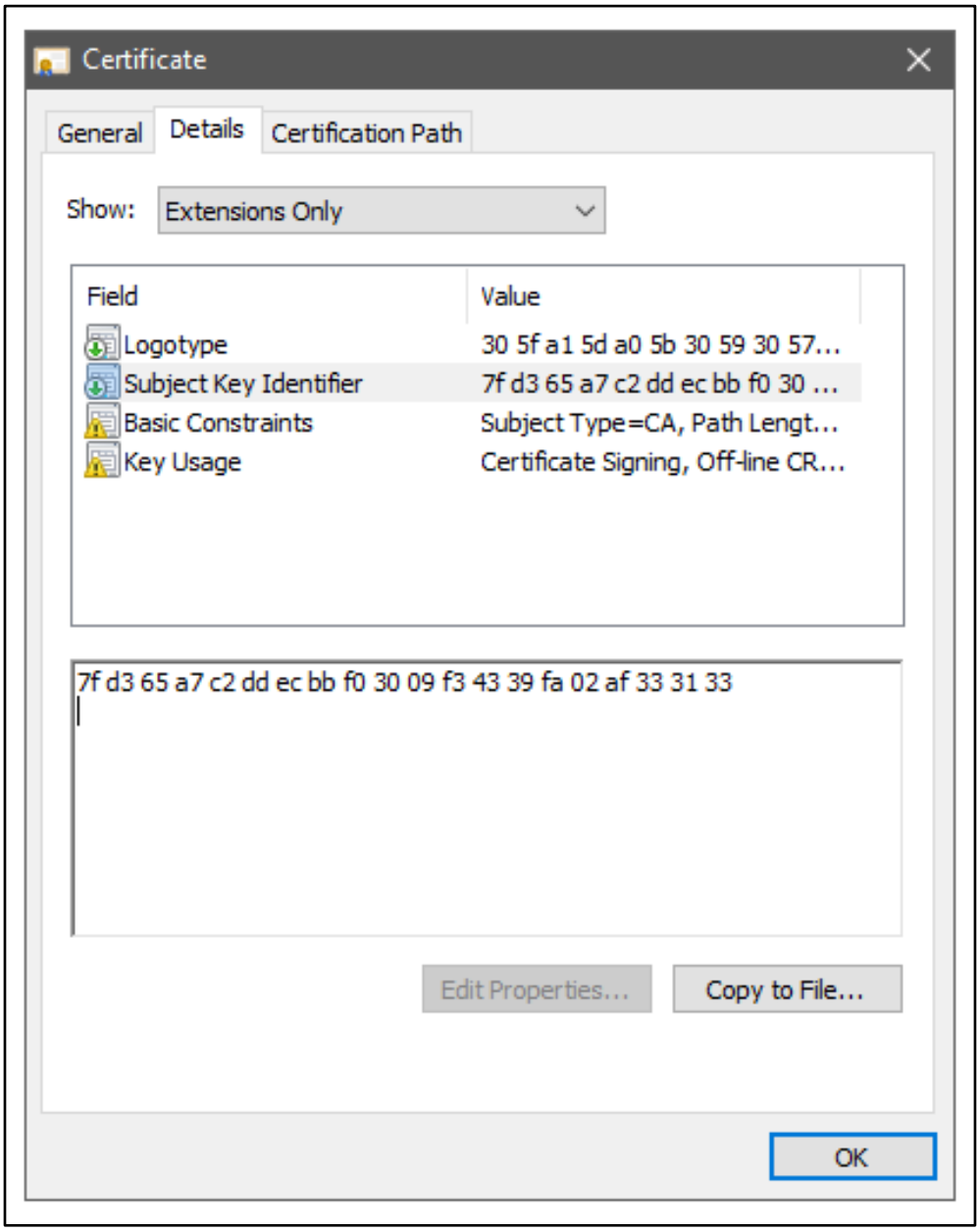


5 Click *View Certificate* to view the Root CA certificate.

6 Click the *Details* tab to view certificate details

The Details include the *Subject Key Identifier*, which can be saved into a binary file (using a dedicated tool like, for example, the *Hex Edithttp://www.hexedit.com/* software) in order to store it into the SPWF04S module.

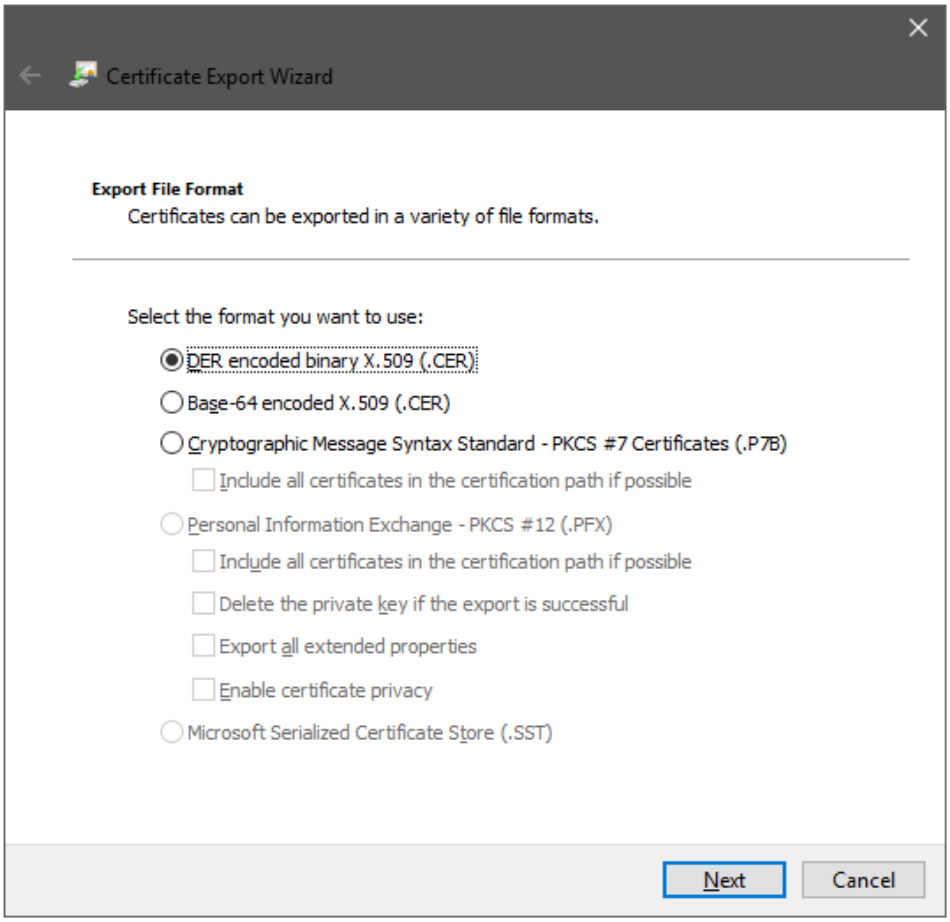
Figure 18: Certificate window – Details



7 Click *Copy to File…* to export the certificate.

A *Certificate Export Wizard* window will guide the whole export process

Figure 19: Certificate Export Wizard



8 Select:*DER encoded binary X.509 (.CER)* to save the certificate as a binary file or*Base-64 encoded X.509 (.CER)* to save the file as a PEM file.

Once the certificate is correctly exported, it is possible to load into the SPWF04S module.

Appendix C How to set up a WPA2-Enterprise network

802.1X server implementation with Raspberry Pi

The Raspberry Pi 3 model B is a credit card-sized single board computer able to run several operating systems like Windows or Linux.

For the implementation of the 802.1X server, Raspberry Pi 3 model B is used with the Linux based *Raspbianhttps://www.raspbian.org*/operating system. The setup process of the Raspberry Pi requires an Internet connection, the possibility to make an SSH connection with the device and root privileges for the device.

For network settings, modify the file /etc/network/interfaces.

For example, to use the static address 192.168.1.100 on the eth0 interface, change the line

```
iface eth0 inet dhcp
```

to:

```
auto eth0  iface eth0 inet static  address 192.168.1.100  netmask 255.255.255.0
gateway 192.168.1.1
```

In order to use the Raspberry Pi as an 802.1X server, dedicated software is needed. This example uses FreeRADIUS *https://www.raspbian.org/,* an open source 802.1X server widely used in enterprise networks.

To install FreeRADIUS on the Raspberry Pi just type in the console:

```
apt-get install freeradius
```

Once the installation is completed, the FreeRADIUS server will be running in background. If needed, for example for testing purposes, it is possible to stop the service in background and open FreeRADIUS in debug mode. In order to stop the service, use the following command:

```
service freeradius stop
```

In order to start freeradius in debug mode use the following command:

```
freeradius –X
```

In order to make FreeRADIUS running again in background use the following command:

service freeradius start

FreeRADIUS is configured by default to use ports 1812 and 1813 respectively for authentication and accounting. These values are given in the /etc/services file.

To use different port numbers, edit the file /`etc/freeradius/radius.conf`.

In this file, two sections named “listen” are present. The first one is used to configure the authentication in FreeRADIUS and looks like this (for readability, comments are removed):

```
listen {
   type = auth
   ipaddr = *
   port = 0
}
```

The value 0 in the port field means that FreeRADIUS will use the port specified in /etc/services. Changing this value will change the port used for authentication.

In the same way, it is possible to modify the port used by FreeRADIUS for accounting editing the following field:

```
listen {
   type = acct
   ipaddr = *
   port = 0
}
```

In the default configuration of FreeRADIUS, the users of the enterprise network are registered into the file `/etc/freeradius/users` and the needed X.509 certificates (own certificate and private key, clients CA etc.) have to be put into the directory `/etc/freeradius/certs.`

In order to add a new user to the network, e.g. a user with username 'test_user' and password 'test_password', add the following lines into `/etc/freeradius/users:`

```
test_user Cleartext-Password := "test_password"
```

The needed certificates and private key can be generated using OpenSSL as specified in Appendix A. It is important to check that the names of the generated certificates are the same as the ones specified in `/etc/freeradius/eap.conf` in the tls section.

For example, if the server's certificate is called server.pem, in the tls section of the `/etc/freeradius/eap.conf` file, there will be the line:

```
certificate_file = {certdir}/server.pem
```

The Access Point and the 802.1X server have to share a secret text string, in order to encrypt passwords and exchange responses during the RADIUS protocol.

To configure this secret text string into the FreeRADIUS, open the /`etc/freeradius/clients.conf` file with a text editor and add the following lines:

```
client <Access Point Identifier string> {
   ipaddr = <Access Point IP address>
   secret = <secret password>
}
```

- `<Access Point Identifier string>` is an arbitrary string used to identify the Access Point
- `<Access Point IP address>` must be changed with the actual IP address of the used Access Point
- `<secret password>` is an arbitrary string to be used as a secret shared with the Access Point.

IronWifi Cloud based 802.1X server configuration

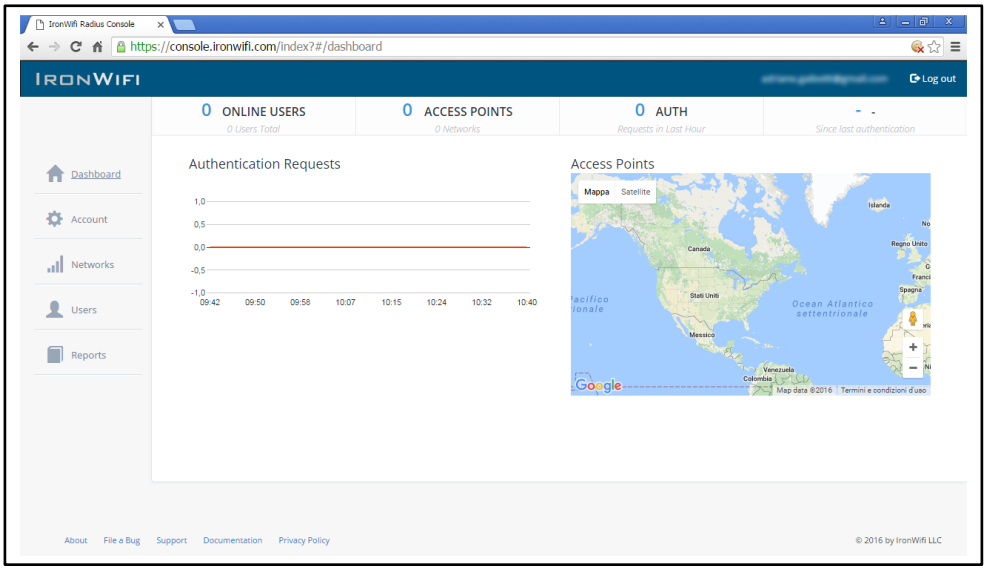
IronWifi is a cloud-based RADIUS authentication and accounting service that provides resizable capacity in the cloud. It provides a user friendly web interface and gives the possibility of certificates, users and network management.

The *basic* plan is free and supports up to ten users and one Access Point.

To use IronWifi as 802.1X, the Access Point needs Internet access

1 Log in or open an account at *https://www.ironwifi.com* to use the IronWifi service:

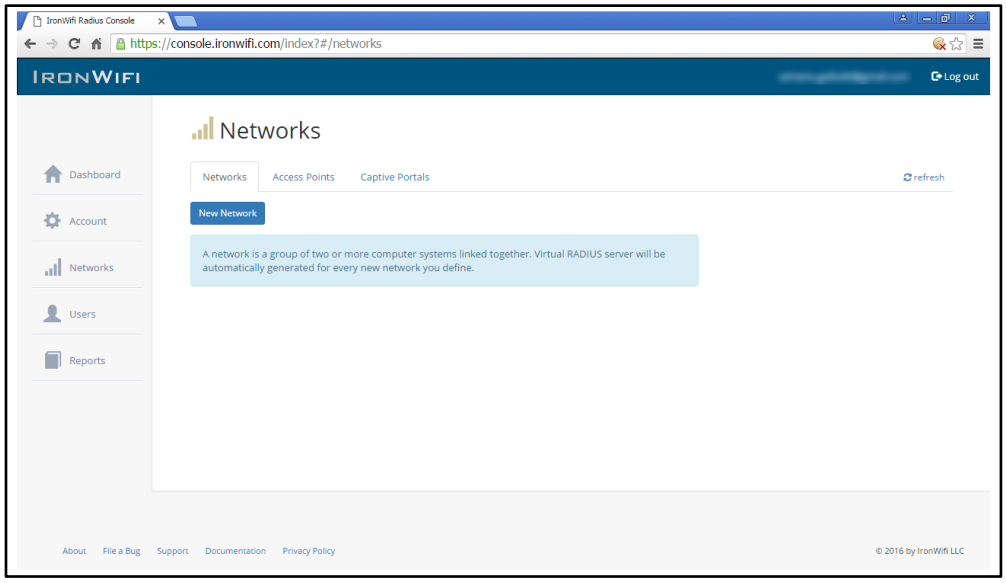
Figure 20: IronWifi's dashboard



2 Click on *Account* to view your account settings and download the CA certificate (in DER format) for 802.1X server authentication.

3 Enter the *Networks* section and select the *Networks* tab

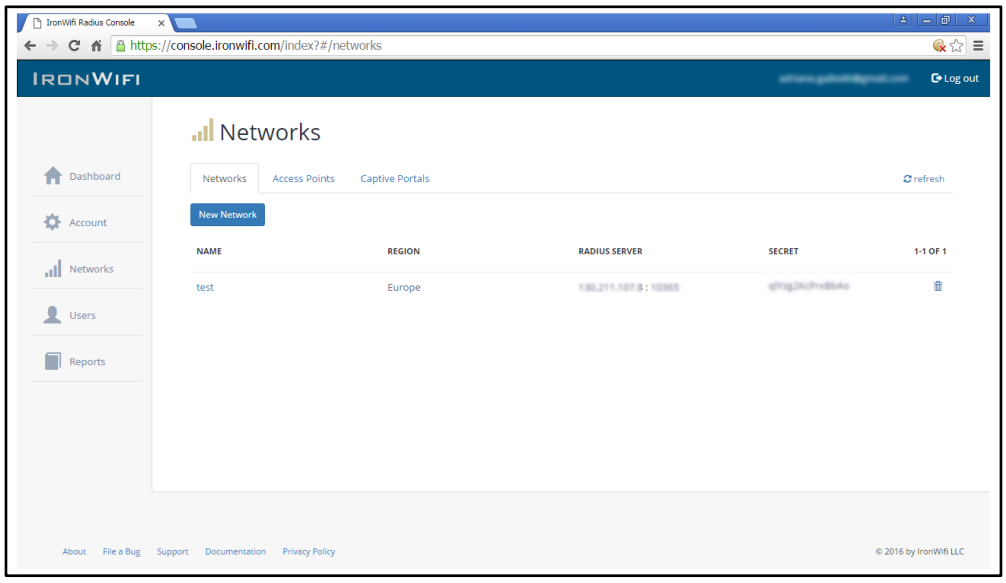
Figure 21: IronWifi's Networks page



4 Click on *New Network*.

The created network will be associated with a new 802.1X server, an IP address, a port for the authentication protocol and a secret string to share with the Access Point.

Figure 22: 802.1X server information on IronWifi



5 Select the *Access Points* tab in the *Networks* section and click on *New Access Point*.

6 Name the Access Point and provide other information like the MAC address.

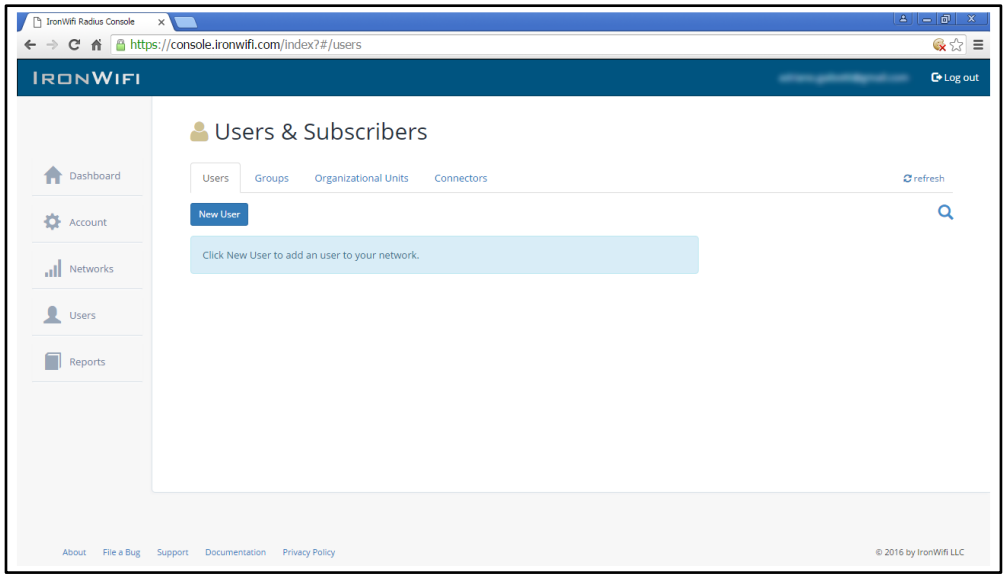
7 Open the *Users* section to create users, groups and organizational units.

It is also possible to import databases through the *Connectors* tab.

8 Create a new user and edit the relevant user details

A user must be associated with an organizational unit

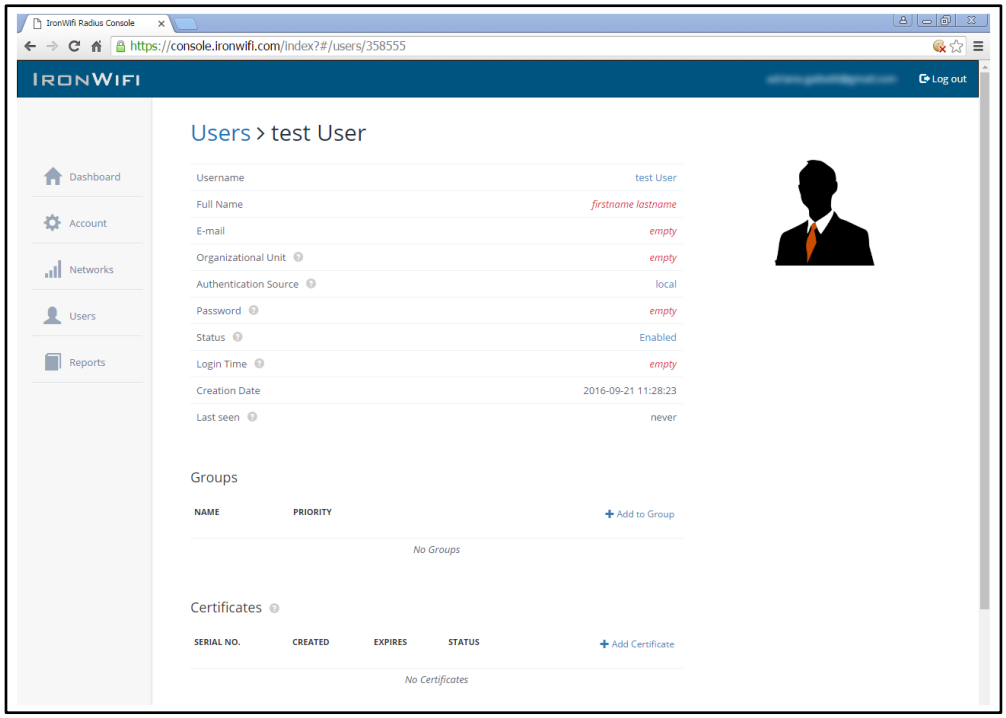
Figure 23: IronWifi's Users page



9 Select + *Add Certificate* export its X.509 certificate and private key.

The certificate and the private key are provided as a single file with extension *.p12*; see *Section "Extract private key and certificate from a .p12 file"*.*Certificates and private keys provided by the basic version of IronWiFi are RSA-512 based.*

Figure 24: IronWifi's created user page



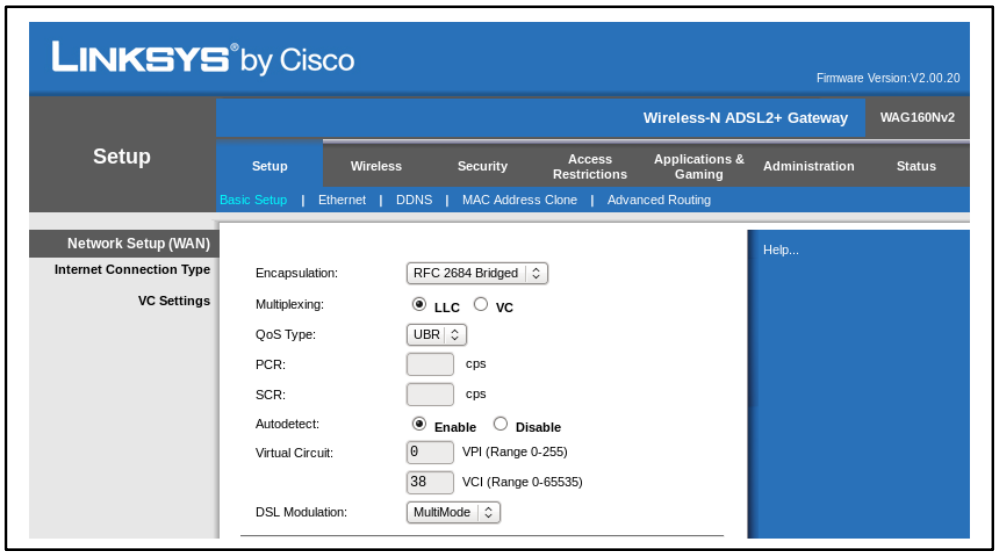
WPA2-Enterprise network setup

The last step in getting the enterprise network working is configuring the Access Point.

1 From a web browser running on a PC connected to the network, open the configuration page of the Access Point.

The figure below shows an example configuration page for the Linksys WAG160N Access Point.

Figure 25: Configuration page for Linksys Access Point



2 To configure the Access Point for a WPA2-Enterprise network, go to the section Wireless → Wireless Security and check the following options:

- Security Mode = WPA-Enterprise or WPA2-Enterprise
- Encryption = TKIP or AES
- RADIUS Server = <IP address of the 802.1X server>
- RADIUS Port = 1812
- Share Key = <secret password>

where <IP address of the 802.1X server> and <secret password> are the values previously specified during the setup of the 802.1X server.

Once the setup of the Access Point is done, the enterprise network is ready.

Figure 26: WPA2-Enterprise configuration for Linksys Access Point

5 Revision history

Table 11: Document revision history

Date	Version	Changes
02-May-2017	1	Initial release.
15-Nov-2017	2	Updated *Section 3: "SPWF04S security"*. Updated *Section 3.2.2.2: "EAP-TLS authentication"*. Updated *Section 4.3: "Example: TLS server with mutual authentication"*. Updated *Section 4.4: "Example: TLS server with one-way authentication"*. Updated *Section 4.6: "Example: HTTPS with Amazon"*. Minor text changes

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.