



## APPLICABILITY TABLE

PRODUCT
GC864-QUAD
GC864-QUAD V2
GC864-DUAL V2
GE864-QUAD
GE864-QUAD AUTOMOTIVE V2
GE864-QUAD ATEX
GE864-QUAD V2
GE864-DUAL V2
GE864-GPS
GE865-QUAD
GL865-DUAL
GL865-QUAD
GL868-DUAL













## 1.4. Document Organization

This document contains the following chapters:

[Chapter 1: “Introduction”](#) provides a scope for this document, target audience, contact and support information, and text conventions.

[Chapter 2: “Preliminary Operations”](#) is about context setting, activation, data states. Furthermore it gives some information about SSL server requirements.

[Chapter 3: “Configuring SSL”](#) describes the steps to be followed in order to configure security settings and data as well as general parameters.

[Chapter 4: “Working with SSL sockets”](#) describes how to connect the module to an SSL server and how to perform data exchange.

[Chapter 5: “SSL Error Codes”](#)

[Chapter 6: “Document history”](#)

## 1.5. Text Conventions



***Danger – This information MUST be followed or catastrophic equipment failure or bodily injury may occur.***



***Caution or Warning – Alerts the user to important points about integrating the module, if these points are not followed, the module and end user equipment may fail or malfunction.***



**Tip or Information – Provides advice and suggestions that may be useful when integrating the module.**

All dates are in ISO 8601 format, i.e. YYYY-MM-DD.

## 1.6. Related Documents

- AT Command Reference Guide, 80000ST10025a [1]
- Telit GSM/GPRS SW User Guide, 1vv0300784 [2]
- IP Easy User Guide, 80000ST10028A [3]





Where:

<Cntx Id> is the context that we want to activate/deactivate.

<Status> is the desired context status (0 means deactivation, 1 activation).



**Warning – SSL sockets can be opened only on <cid>=1, so the user must open it by issuing AT#SGACT=1,1**

### Example:

We want to activate context number one defined with +CGDCONT.

*Command:*

```
AT#SGACT=1 , 1
```

*Answer:*

```
#SGACT: "212.195.45.65"
```

*OK if activation success.*

*ERROR if activation fails.*

The response code to the AT#SGACT=1 command reports the IP address obtained from the network, allowing the user to report it to his server or application.

## 2.2. SSL Server requirements

### 2.2.1. Cipher suites

The user must set up his SSL server according to the cipher suites provided by the Telit module:

- SSL\_RSA\_WITH\_RC4\_128\_MD5
- SSL\_RSA\_WITH\_RC4\_128\_SHA
- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA

If the server doesn't support these cipher suites, the SSL handshake will always fail.



## 2.2.2. Certificate size

The Telit module allows the user the storage of a server certificate. Its maximum size is 2047 bytes; this means that a server with a bigger certificate cannot be authenticated.

A procedure to get the CA certificate to be used in case of a connection to an HTTPS server is explained in paragraph 3.3.1.





**AT#SSLSECCFG= <SSId>,<cipher\_suite>,<auth\_mode>**

Where:

<SSId> is the SSL socket ID that we want to configure. Since at the moment only a single socket can be configured and used, it has to be 1.

<Cipher Suite> represents the set of algorithms which will be used to negotiate the security settings. It includes a key exchange algorithm (used for the authentication during the handshake), an encryption algorithm (used to encrypt the message stream) and the message authentication code (a hash algorithm).

The Telit module supports the following cipher suites:

- 1 SSL\_RSA\_WITH\_RC4\_128\_MD5 (RSA key exchange algorithm, RC4\_128 encryption and MD5 hashing algorithm).
- 2 SSL\_RSA\_WITH\_RC4\_128\_SHA (RSA key exchange algorithm, RC4\_128 encryption and SHA hashing algorithm).
- 3 TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA (RSA key exchange algorithm, AES\_256 encryption and SHA hashing algorithm).

Setting the value 0 the cipher suite is chosen by the remote server (among the supported suites).

<auth\_mode> is the authentication mode. SSL/TLS protocols provide several levels of authentication: anonymous, server authentication and server/client authentication. The choice depends of course on the user's application and on the desired protection against untrusted peers.

Depending on this parameter the user may need to store different security data:

- 0 No authentication → No security data is needed at all
- 1 Server authentication (the most common case) → CACertificate storage is needed
- 2 Client/Server authentication → CACertificate (server), Certificate (client) and private key (client) are needed

If the authentication mode is 1 or 2, the storing of the security data is needed; otherwise next paragraph can be skipped.

### 3.3. Storing security data

In order to perform server or server/client authentication, the user has to store the proper security data (certificate(s) and/or private key) into the module's NVM. This operation is performed by the AT#SSLSECDATA command.

Write, delete and read operations are supported. The command syntax for the write operation is:



**AT#SSLSECDATA=<SSId>,<Action=1>,<DataType>**

Where:

**<SSId>** is the SSL socket ID in which we want store the security data. Since at the moment only a single socket can be configured and used, it has to be 1.

**<Action>** is the action to be performed. In order to store data within NVM it has to be 1.

**<DataType>** is the certificate/key to be stored.

- 0 Certificate (certificate of the client module) → Needed if the client/server authentication has been configured.
- 1 CA Certificate (certificate of the remote server) → Since it's used to authenticate the remote server, it's needed for any non-anonymous authentication.
- 2 RSA private key (private key of the client module) → Needed if the client/server authentication has been configured.

Immediately after the command, the '>' prompt is printed. Then the user can send to the module the certificate to be stored and confirm with the **0x1A** character (CTRL+Z).



**Warning – Certificates MUST be in PEM format. When you store them within the module, remember that after each row only a <LF> character is needed (without <CR>). Furthermore be aware that some serial terminals add an undesired EOF at the end of the certificate; in this case EOF must be removed before to enter CTRL+Z**

### 3.3.1. Obtaining the right CA certificate

During the handshake, the server sends a certificate chain

```
ServerCert -> AuthorityCert1 -> ... -> AuthorityCertN -> RootCACert
```

ServerCert is the certificate of the server we want to connect to; AuhorityCert1...N are CA certificates of intermediate authorities; RootCACert is the certificate of a global recognized Certificate Authority.

Each certificate is signed by the certificate on its right in the chain except the RootCACert which is self-signed.

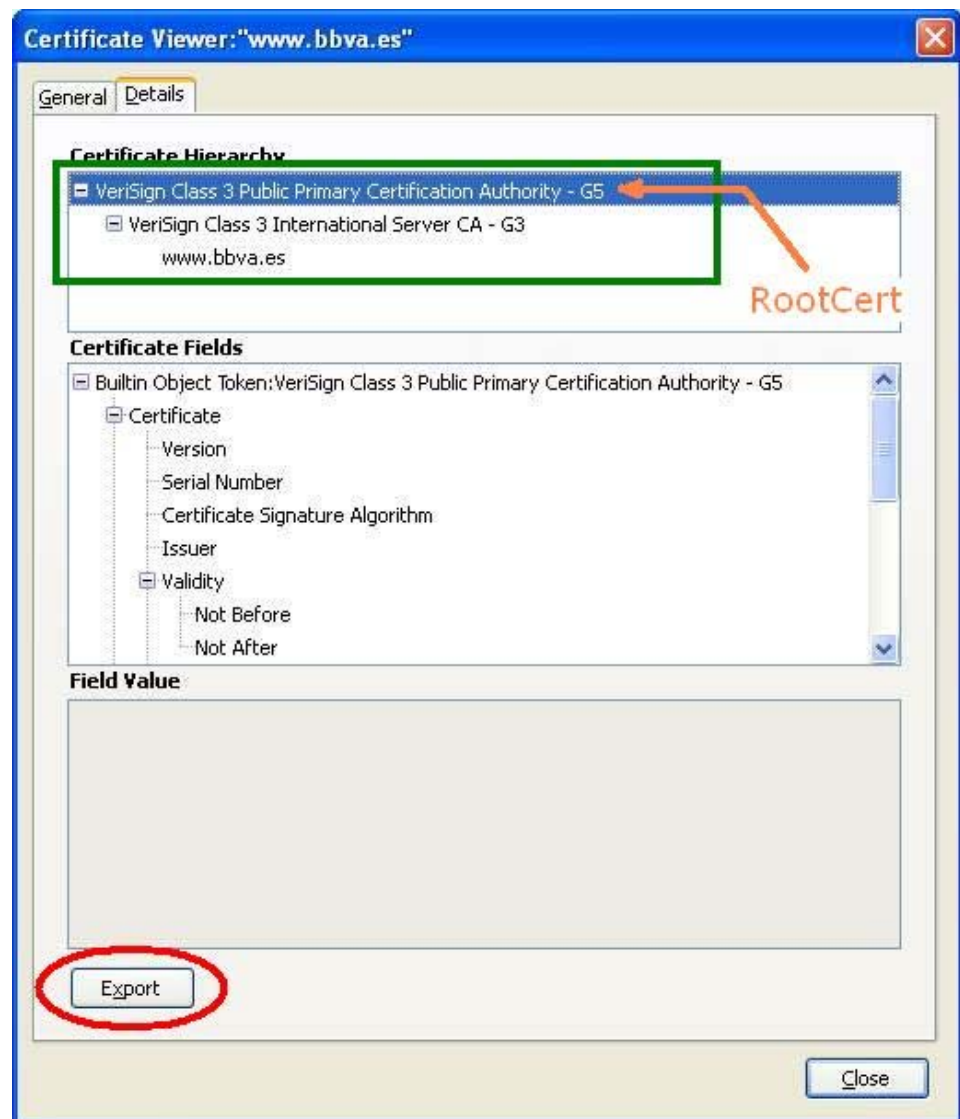
The CA certificate to be stored within the Telit module in order to perform the server authentication is the RootCACert.

If you have to connect to an HTTPS server, you can usually obtain the CA certificate from your browser (though sometimes servers send a different chain). For example, in order to get from Mozilla Firefox the RootCertificate of the Bilbao Bank website, click on the colored certificate area on the right of the address bar:





Then click on “More Information” and in the new window, select the “Security” tab, and click on “View Certificate”.



In the picture shown above, the “Certificate Hierarchy” section contains the certificate chain for the [www.bbva.es](http://www.bbva.es) website. The root CA certificate is the first one so select it and click on the “Export” button: in this way you will be able to save the certificate in PEM format.







```
OK
AT#SSLSECDATA=1,1,1
> -----BEGIN CERTIFICATE-----<LF>
[...]
-----END CERTIFICATE-----<LF>
[CTRL+Z]
OK
AT#SSLSECDATA=1,1,2
> [... private key ...][CTRL+Z]
OK
```

Now the module is ready for SSL socket dial.





## 4.2. Exchange data through a secure socket

### 4.2.1. Online mode

After the `CONNECT` message, the user can send data to the AT port which will be encrypted and sent to the server through the secure socket as soon as the packet size has been reached or the `txTo` expires (see paragraph 3.4 to configure these parameters).

During an online session the user cannot send AT commands on the used serial port/VPort (if CMUX is activated is possible to send AT commands thru the others VPort); anyway it is possible suspend the connection (without closing it) by sending the escape sequence (`+++`). After that, the module prints the `OK` response and is able to parse AT commands again.

Data mode can be restored at any time by sending the `AT#SSLO` command.

Its syntax is:

**`AT#SSLO=<SSId>`**

Where:

`<SSId>` is the SSL socket ID in which we want to restore the online communication. Since at the moment only a single socket can be configured and used, it has to be 1.

After the restore command, the `CONNECT` code is printed, and SSL communication can continue.

If the idle inactivity timeout expires (see paragraph 3.4) or the remote server force the connection closure, the `NO CARRIER` message is printed.

### 4.2.2. Command mode

Data can be exchanged in command mode through an SSL socket by means of the commands `AT#SSLSEND` and `AT#SSLRCV`. The data exchange is performed in blocking mode.

Note: at any moment the user can switch to online mode by sending the `AT#SSLO` command (described in the previous paragraph).

#### 4.2.2.1. Send data

The syntax of the command is:

**`AT#SSLSEND=<SSId>,<Timeout>`**



Where:

<**SSId**> is the SSL socket ID through which we want to send data. Since at the moment only a single socket can be configured and used, it has to be 1.

<**Timeout**> is the maximum blocking timeout. It can be omitted, and in this case the default timeout configurable with AT#SSLCFG will be used.

The data to be sent can be written to the AT port after the '>' prompt; the user can confirm by writing the 0x1A character (ctrl+z) and the data will be forwarded through the secure socket.

Response:

OK on success

ERROR on failure

#### 4.2.2.2. Receive data

The syntax of the command is:

**AT#SSLRCV=<SSId>,<MaxNumByte>,<Timeout>**

Where:

<**SSId**> is the SSL socket ID from which we want to receive data. Since at the moment only a single socket can be configured and used, it has to be 1.

<**MaxNumByte**> is the maximum number of bytes that will be read from socket. The user can set it according to the expected amount of data.

<**Timeout**> is the maximum blocking timeout. It can be omitted, and in this case the default timeout configurable with AT#SSLCFG will be used.

On success, the data is printed in the following format:

#SSLRCV: <numBytesRead>

<DATA>

OK

where <numBytesRead> is the number of bytes actually read (equal or less than <MaxNumBytes>).

If the timeout expires, the module prints the following response

#SSLRCV: 0









```
OK
AT#SSLH=1
DISCONNECTED
```

```
OK
```

The server's response has been the string "Response of the server!".

### 4.5.3. Switch between command and online mode

We open the socket to the IP 123.124.125.126, where we suppose there's an SSL server listening at port 443. After some data exchange, we suspend the connection and exchange data in command mode. At the end we close the SSL socket.

```
AT#SSLD=1,443,123.124.125.126,0,0
CONNECT
..
[bidirectional data exchange]
..
[send +++]
OK
AT#SSLS=1 ← Query status
#SSLS: 1,2,<cipher_suite>

OK
AT#SSLSEND=1
> Send data in command mode![ctrl+z]

OK
AT#SSLRCV=1,100
#SSLRCV: 24
Response in command mode

OK
```



```
AT#SSLH=1
OK
```

#### 4.5.4. Open an SSL socket and restore it later

We open the socket to the IP 123.124.125.126, where we suppose there's an SSL server listening at port 443, setting <closureType>=1. We exchange some data in online mode, and then we close the socket and reopen it using #SSLFASTD. Finally we close definitively the socket after an exchange of data.

```
AT#SSLD=1,443,123.124.125.126,1,0
CONNECT
..
[bidirectional data exchange]
..
[send +++]
OK
AT#SSLH=1
OK
AT#SSLFASTD=1,0
..
[more bidirectional data exchange]
..
[send +++]
OK
AT#SSLH=1,0 ← Force definitive closure
OK
```

#### 4.5.5. Connect to an HTTPS server

In this example we connect to the Bilbao Bank website, and then we get a webpage (in this example we get a simple error page). Previously, we have retrieved the CA certificate (just as explained in paragraph 3.3.1) and stored it within the module.

```
AT#SSLSECCFG=1,0,1
OK
```





```
content-type: text/html
date: Thu, 11 Aug 2011 08:31:33 GMT
last-modified: Tue, 03 Jul 2001 12:58:58 GMT
p3p: CP="NON CUR OTPi OUR NOR UNI"
```

```
<html>                                ← Server response: HTML page
<head>
<title>Recurso no encontrado</title>
</head>

<body bgcolor="ffffff">
<table border="0" cellpadding="0" cellspacing="0"
align="center" width="100%" height="100%">
<tr>
<td align="center" valign="middle">
        <table border="0" cellpadding="0" cellspacing="0"
align="center" width="508">
        <tr>
        <td width="508" height="107" align="left" colspan="2">
                
        </td>
        </tr>
        </table>
</td>
</tr>
</table>
</body>
</html>
```

NO CARRIER ← *Server remote closure. In fact some servers are configured in order to close the socket after a single request.*



## 5. SSL Error Codes

In this chapter we show the list of SSL related errors:

SSL related errors (only if command #SELINT=2 has been issued – see §3.5.2.1.1):	
830	SSL generic error
831	SSL cannot activate
832	SSL socket error
833	SSL not connected
834	SSL already connected
835	SSL already activated
836	SSL not activated
837	SSL certs and keys wrong or not stored
838	SSL error enc/dec data
839	SSL error during handshake
840	SSL disconnected



