

Introduction

The STM32Cube initiative was originated by STMicroelectronics to ease developers life, by reducing development efforts, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows to generate C initialization code, using graphical wizards
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF4 for STM32F4 series)
 - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across the STM32 portfolio
 - A consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
 - All embedded software utilities coming with a full set of examples.

With the increasing number of embedded devices interconnected over the network, hardware-based cryptographic capabilities are required, to ensure secure transactions. The integrated Ethernet MAC and cryptographic processor of the STM32, make it best fits for such applications. The embedded Ethernet features a 10/100 Mbit/s MAC, it supports both the Media Independent Interface (MII) and the Reduced Media Independent Interface (RMII), giving developers the flexibility to connect to the PHY of their choice. Performance is further enhanced through the use of a dedicated DMA controller, and hardware checksums for the IP, UDP, TCP and ICMP protocols.

The hardware cryptographic processor supports AES/128/192/256, Triple DES, DES, SHA-1, SHA-2, MD5 and RNG.

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) cryptographic protocols provide security for communications over networks, such as the Internet, and allow client and server applications to communicate in a way that is private and secure. The purpose of this user manual is to present an SSL Client/Server example, built on top of STM32Cube HAL drivers and the PolarSSL library (a free SSL/TLS library).

Note: This document is applicable to all STM32 Series featuring an Ethernet peripheral and hardware cryptographic processor; for simplicity reason, STM32F4xx and STM32CubeF4 are used as reference platform within all the documents. The same description, file names and screenshot are applicable as well to other Series offering Ethernet connectivity, such as STM32F217xx and STM32F756xx.

To know more about the PolarSSLexample implementation on STM32 Series, refer to the documentation provided within the associated STM32Cube firmware package.



Contents

- 1 SSL / TLS protocol overview 5**
 - 1.1 History of the SSL / TLS protocols 5
 - 1.2 SSL / TLS application layers 5
 - 1.3 SSL / TLS sub-protocols 6
 - 1.3.1 SSL Handshake protocol 7
 - 1.3.2 SSL Record protocol 9
 - 1.3.3 SSL Alert protocol 9
 - 1.3.4 Change Cipher Spec protocol 9

- 2 PolarSSL library 10**
 - 2.1 Overview 10
 - 2.2 License 10

- 3 STM32 hardware cryptography 11**
 - 3.1 Cryptographic processor11
 - 3.2 Random number generator11
 - 3.3 Hash processor11

- 4 Description of the package 12**
 - 4.1 Package directories 12
 - 4.2 Application settings 12
 - 4.2.1 PHY interface configuration 12
 - 4.2.2 MAC and IP address settings 13
 - 4.3 Evaluation boards settings 13

- 5 Using the applications 14**
 - 5.1 SSL client application 14
 - 5.2 SSL server application 17

- 6 Conclusion 21**

- 7 FAQ 22**

- Appendix A Additional information..... 23**

8 Revision history 25

List of figures

Figure 1. SSL application architecture 6
Figure 2. SSL sub-protocols 6
Figure 3. SSL Handshake protocol 7
Figure 4. Handshake protocol to resume an SSL session 9
Figure 5. SSL Record protocol 9
Figure 6. SSL client demonstration architecture 15
Figure 7. SSL client application 16
Figure 8. The ssl_server application window 16
Figure 9. HyperTerminal window 17
Figure 10. SSL server application architecture 18
Figure 11. The SSL server application 19
Figure 12. HTML page displayed on successful connection 19
Figure 13. HyperTerminal SSL server connection status 20
Figure 14. SSL client thread flowchart 23
Figure 15. SSL server thread flowchart 24

1 SSL / TLS protocol overview

The Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols provide communications security over the Internet and allow client/server applications to communicate in a way that is private and reliable. These protocols are layered above a transport protocol such as TCP/IP.

SSL is the standard security technology for creating an encrypted link between server and client. This link ensures that all communication data remains private and secure.

The major objectives of SSL/TLS are:

- Provide data integrity between two communicating applications.
- Protect information transmitted between server and client.
- Authenticate the server to the client.
- Allow the client and server to select the cryptographic algorithms that they both support.
- Optionally authenticate the client to the server.
- Use public-key encryption techniques to generate shared secrets.
- Establish an encrypted SSL connection.

1.1 History of the SSL / TLS protocols

SSL was developed by Netscape in 1994 to secure transactions over the Internet. Soon after, the Internet Engineering Task Force (IETF) began work to develop a standard protocol to provide the same functionality.

- SSL 1.0 (Netscape, 1993): Internal Netscape design
- SSL 2.0 (Netscape, 1994): This version contained a number of security flaws
- SSL 3.0 (Netscape, 1996): All Internet browsers support this version of the protocol
- TLS 1.0 (IETF, 1999): This version was defined in RFC 2246 as an upgrade to SSL 3.0 “The differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that TLS 1.0 and SSL 3.0 do not inter-operate”
- TLS 1.1 (IETF, 2006): This version was defined in RFC 4346. It is an update from TLS version 1.0
- TLS 1.2 (IETF, 2008): This version was defined in RFC 5246. It is based on the earlier TLS 1.1

Note: The “SSL/TLS” protocols are referred by “SSL” throughout this document.

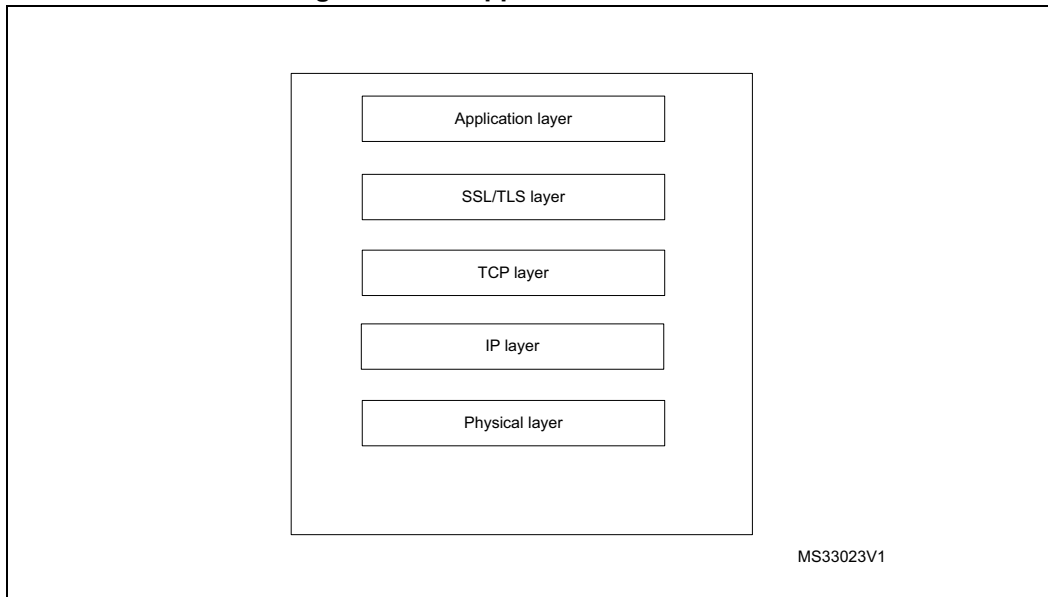
1.2 SSL / TLS application layers

An application using SSL / TLS protocol consists generally of five layers:

- Application layer: the Application Layer refers to the higher-level protocols used by most applications for network communication
- SSL/TLS layer: the SSL/TLS layer provides security communication over the Internet

- TCP layer: the Transport Layer responsibilities include end-to-end message transfer capabilities independent of the underlying network, along with error control, segmentation, flow control, congestion control, and application addressing
- IP layer: the Internet Protocol layer is responsible for addressing hosts and routing packets from a source host to the destination host
- Physical layer: the Physical Layer consists of the basic hardware transmission technologies of a network

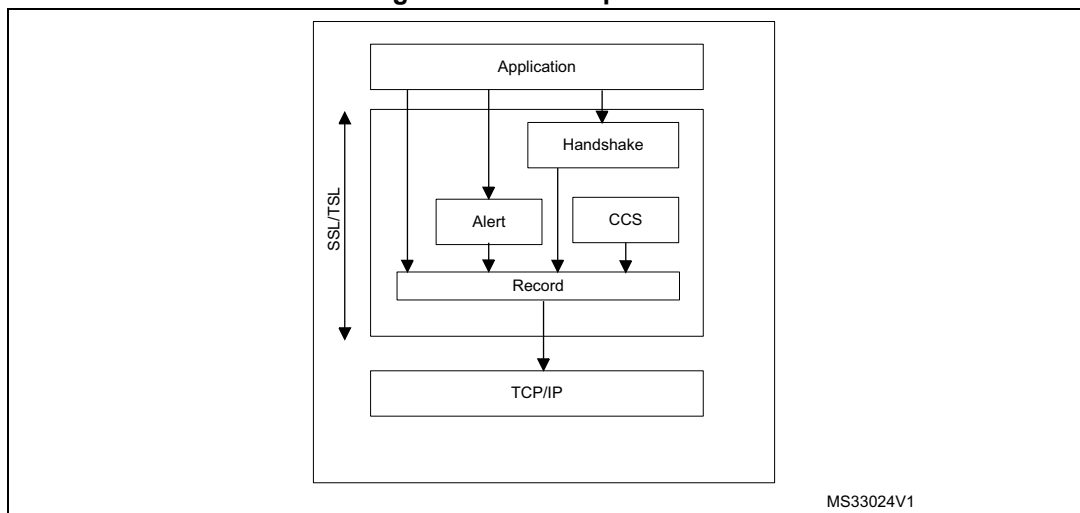
Figure 1. SSL application architecture



1.3 SSL / TLS sub-protocols

The SSL / TLS protocol includes four sub-protocols: the SSL Record protocol, the SSL Handshake protocol, the SSL Alert protocol and the SSL Change Cipher Spec protocol.

Figure 2. SSL sub-protocols

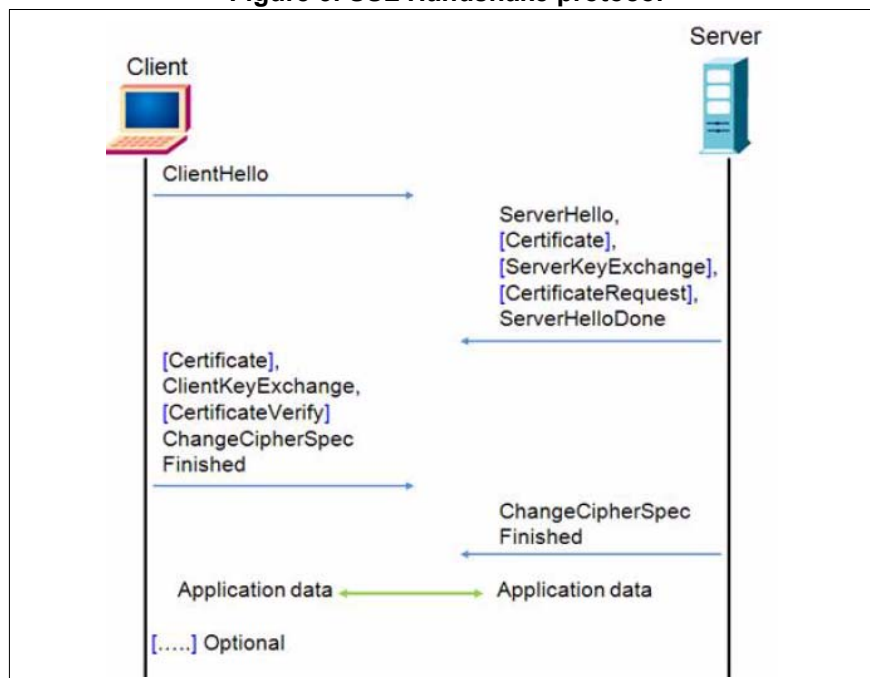


1.3.1 SSL Handshake protocol

The SSL session state is controlled by the SSL Handshake protocol. This protocol uses the SSL Record protocol to exchange a series of messages between SSL server and SSL client, when they first start communicating. This message exchanging is designed to facilitate the following actions:

- The protocol version
- Allow the client and server to select the cryptographic algorithms, or ciphers, that they both support
- Authenticate the server to the client
- Optionally authenticate the client to the server
- Use public-key encryption techniques to generate shared secrets
- Establish an encrypted SSL connection

Figure 3. SSL Handshake protocol



The following description is the procedure for SSL Handshake protocol:

1. The client sends a ClientHello message specifying the highest SSL protocol version it supports, a random number, a list of cipher suites and compression methods
2. Server responds with a ServerHello message that contains the chosen protocol version, another random number, cipher suite and compression method from the choices offered by the client, and the session ID

Note:

The client and the server must support at least one common cipher suite, or else 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. If no certificate is sent, an optional ServerKeyExchange message is sent containing the server public information

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 must send its X.509 client certificate in a Certificate message
8. The client sends a ClientKeyExchange message. This message contains the premaster secret number used in the generation of the symmetric encryption keys and the message authentication code (MAC) keys. The client encrypts pre-master secret number with the public key of the server

Note: The public key is sent by the server in the digital certificate or in ServerKeyExchange message.

9. If the client 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 and a finished message from its end
13. The SSL Handshake protocol ends and the encrypted exchange of application data can be started

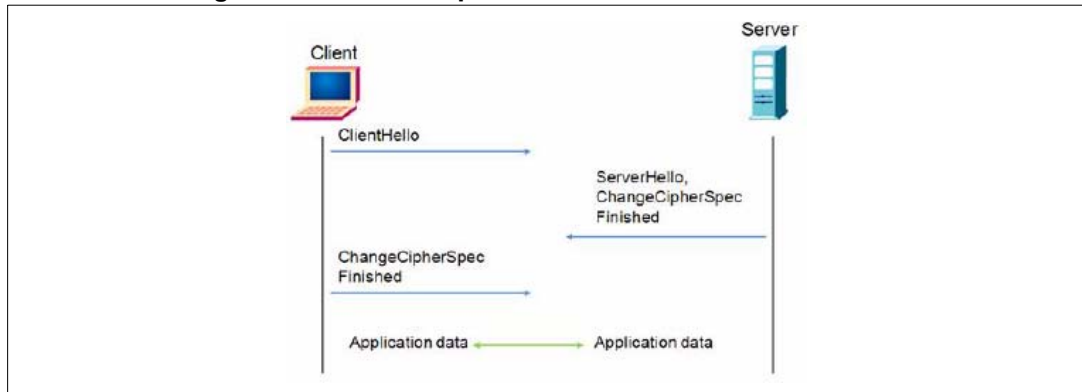
Resuming SSL session

When the client and the server decide to resume a previous session or to duplicate an existing session (instead of negotiating new security parameters), the message flow is as follows:

1. The client sends a ClientHello message using the Session ID of the session to be resumed
2. The server checks its session cache for a match. If a match is found, and the server is willing to re-establish the connection under the specified session state, it sends a ServerHello message with the same Session ID value
3. Both client and server must send ChangeCipherSpec messages and proceed directly to the finished messages
4. Once the re-establishment is complete, the client and server may begin to exchange encrypted application data

Note: If a Session ID match is not found, the server generates a new session ID and the client and server perform a full Handshake protocol [1]: RFC 5246: The TLS protocol version 1.2.

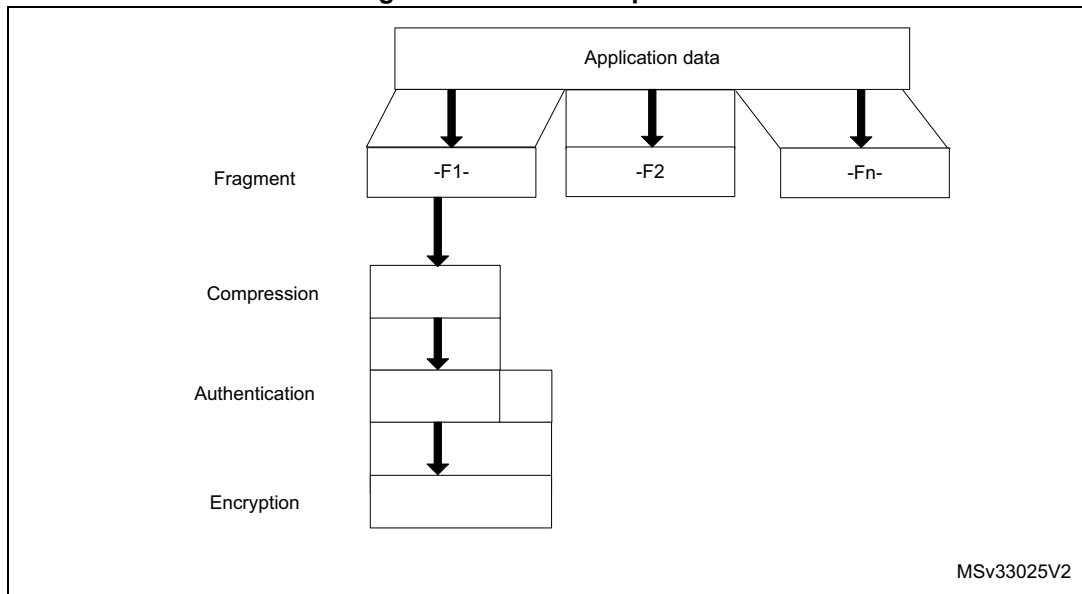
Figure 4. Handshake protocol to resume an SSL session



1.3.2 SSL Record protocol

The Record protocol takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, and transmits the results. The received data is decrypted, verified, decompressed, and reassembled, then delivered to higher level clients.

Figure 5. SSL Record protocol



MSv33025V2

1.3.3 SSL Alert protocol

The SSL Alert protocol signals problems with the SSL session, ranging from simple warnings (unknown certificate, revoked certificate, expired certificate) to fatal error messages that immediately terminate the SSL connection.

1.3.4 Change Cipher Spec protocol

The SSL Change Cipher Spec protocol consists of a single message that indicates the end of the SSL Handshake protocol.

2 PolarSSL library

2.1 Overview

PolarSSL is a light-weight open source cryptographic and SSL/TLS library written in C. This library contains all needed functions to implement an SSL/TLS server or client. It contains also a set of hashing functions and cryptographic algorithms.

Library features:

- SSL 3.0, TLS 1.0 TLS 1.1 and TLS 1.2 client/server support
- Symmetric encryption algorithms: AES, Blowfish, Triple-DES (3DES), DES, ARC4, Camellia, XTEA
- Modes of operation: ECB, CBC, CFB, CTR, GCM
- Hash algorithms: MD2, MD4, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
- Software random number generator: HAVEGE, CTR-DRBG
- X509 certificates, CRLs, Keys and ASN.1
- Public key cryptography: RSA and Diffie-Hellman (DHM) key exchange

The source code of the PolarSSL library can be downloaded from this link:

<http://polarssl.org/>

2.2 License

PolarSSL is licensed according to the dual licensing model. PolarSSL is available under the open source GPL version two license, as well as under a commercial license for closed source projects.

For detailed information about licensing, please refer to this link:

<https://polarssl.org/>

3 STM32 hardware cryptography

As described in [Section 2: PolarSSL library](#), the PolarSSL library contains a set of symmetric encryption algorithms (AES 128/192/256, Triple DES), hashing functions (MD5, SHA-1, SHA-2) and a software random number generator. All these functions and algorithms are needed to implement SSL/TLS applications.

To off-load the CPU from encryption/decryption, hash and RNG (random number generator) tasks, all these functions and algorithms are implemented using the hardware acceleration AES 128/192/256, Triple DES, MD5, SHA-1, SHA-2 and analog RNG through the STM32Cube HAL APIs.

3.1 Cryptographic processor

The cryptographic processor can be used to both encipher and decipher data using the Triple-DES or AES algorithm. It is a fully compliant implementation of the following standards:

- The data encryption standard (DES) and Triple-DES (TDES) as defined by the Federal Information Processing Standards Publication “FIPS PUB 46-3, 1999 October 25”. It follows the American National Standards Institute (ANSI) X9.52 standard
- The advanced encryption standard (AES) as defined by Federal Information Processing Standards Publication “FIPS PUB 197, 2001 November 26”

The CRYPT processor may be used for both encryption and decryption in the Electronic codebook (ECB) mode, the Cipher block chaining (CBC) mode or the Counter (CTR) mode (in AES only).

3.2 Random number generator

The RNG processor is a random number generator based on a continuous analog noise, that provides a random 32-bit value to the host when it is read.

3.3 Hash processor

The hash processor is a fully compliant implementation of the SHA (secure hash algorithm), the (message-digest algorithm 5) hash algorithm and the HMAC (keyed-hash message authentication code) algorithm suitable for a variety of applications. It computes a message digest (160 bits for the SHA-1 algorithm, 256 bits for the SHA-256 algorithm and 224 bits for the SHA-224 algorithm, 128 bits for the MD5 algorithm) for messages of up to $(2^{64} - 1)$ bits, while HMAC algorithms provide a way of authenticating messages by means of hash functions. HMAC algorithms consist in calling the SHA-1, SHA-224, SHA-256 or MD5 hash function twice.

Note: For more detailed information, refer to the CRYPT, HASH and RNG sections of the STM32 device Reference Manual

4 Description of the package

This package contains two applications running on top of the PolarSSL library and LwIP stack in RTOS mode:

- **SSL_Client:** This application proves the ability of the STM32F4xx device to exchange messages with a server over TCP/IP connectivity through a SSL connection. This application allows the user to connect the STM324xx-EVAL board to a secure web server with SSL protocol
- **SSL_Server:** This application is a combination of HTTP with SSL protocol to provide encryption and secure identification of the server. This application allows the user to connect from a web browser to a STM324xx-EVAL evaluation board using SSL protocol

These two applications are located under:

Projects\STM324xx_EVAL\Applications\PolarSSL\, where STM324xx_EVAL refers to STM32F4xx evaluation board such as STM324xG_EVAL for STM32F407xx/417xx devices.

4.1 Package directories

The package contains two applications running on top of PolarSSL, LwIP, FreeRTOS and STM32F4Cube HAL and BSP drivers. The firmware is composed from the following modules:

- **Drivers:** contains the SMT32Cube drivers of the MCU
 - CMSIS
 - BSP drivers
 - HAL drivers
- **Middleware:** contains libraries and protocol components
 - LwIP TCP/IP stack
 - PolarSSL library
 - FreeRTOS
- **Projects:** contains the source file and configurations of each application

4.2 Application settings

4.2.1 PHY interface configuration

The Ethernet peripheral is interfaced with an external PHY to provide physical layer communication. The PHY registers definition are located under the HAL configuration file "stm32f4xx_hal_conf.h".

The PHY operates following two modes, MII and RMII; to select the required mode user has to fill the "MediaInterface" parameter of "Init" structure when initializing the Ethernet peripheral.

Note: Refer to the readme file provided within PolarSSL example of the device, to know about the available PHY interface modes on the supported boards

4.2.2 MAC and IP address settings

The default MAC address is set to: 00:00:00:00:00:02. To change this address, modify the six bytes defined in the *stm32f4xx_hal_conf.h* file.

The default IP address is set to: 192.168.0.10. To change this address, modify the six bytes defined in the *main.h* file.

4.3 Evaluation boards settings

Before running the PolarSSL example, have a look on the associated readme file, to know how to configure the board jumpers for correct operation.

5 Using the applications

5.1 SSL client application

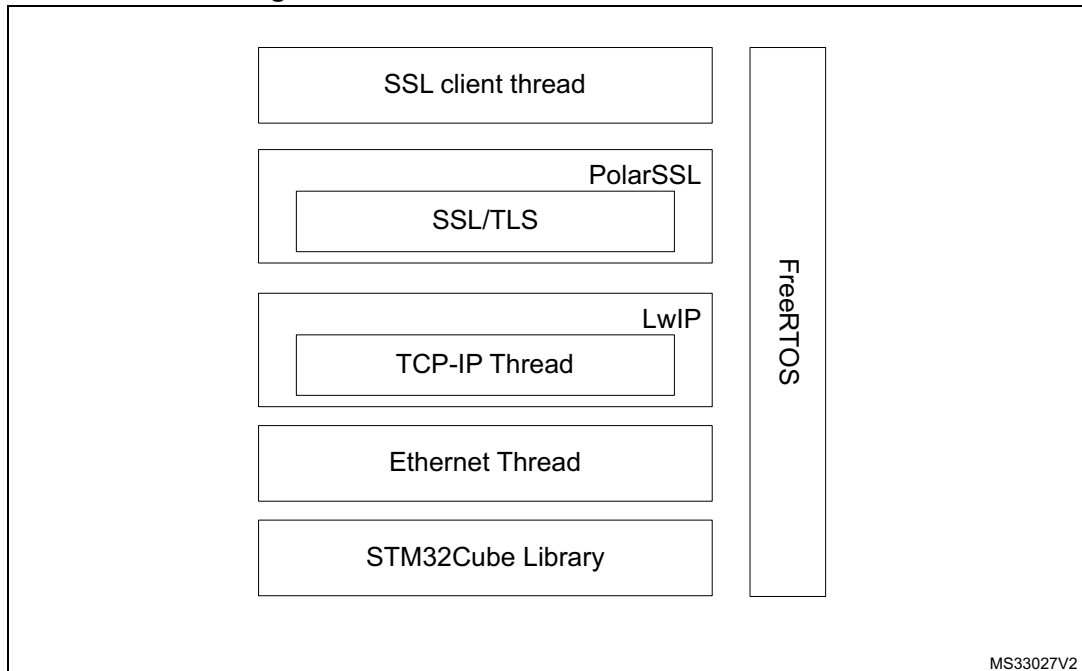
This demonstration consists of using the STM324xx-EVAL evaluation board as a client, that connects to a secure server to provide the SSL Handshake protocol.

Architecture of the application

The SSL client demonstration, as shown in [Figure 6](#) contains five threads:

- **LED task:** blink LED4 every 200 ms
- **Ethernet input thread:** the low-level layer was set to detect the reception of frames by interrupts. So, when the Ethernet controller receives a valid frame, it generates an interrupt. In the handling function of this interrupt, a binary semaphore is created to wake up the Ethernet task. This task transfers the input frames to the TCP/IP stack
- **Ethernet link thread:** handles Ethernet cable connection and disconnection process
- **TCP/IP thread:** all packet processing (input and output) is done inside this thread. The application threads communicate with this thread using message boxes and semaphores.
- **SSL client thread:** this task handles the SSL Handshake protocol. It connects to an SSL server and performs the following:
 - Initializes SSL structures (SSL context, SSL session, SSL RNG)
 - Connects to a SSL server
 - Sets up the SSL session
 - Handles the SSL Handshake protocol
 - Writes a message to the server
 - Reads a message from the server
 - Sends these messages through USART
 - Closes the connection
 - Cleans all SSL structures

Figure 6. SSL client demonstration architecture



How to use the application

First, connect the STM324xx-EVAL evaluation board as follows:

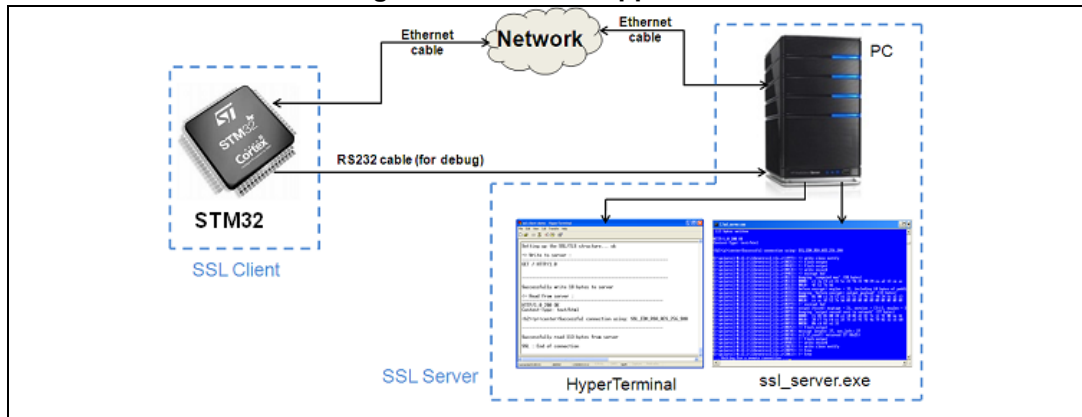
- Ethernet link: connect to a remote PC (through a crossover Ethernet cable) or to the local network (through a straight Ethernet cable)
- RS232 link (used with HyperTerminal like application to display debug messages): connect a null-modem female/female RS232 cable between the USART connector of the STM324xx -EVAL evaluation board and the PC serial port

To run the SSL client example, proceed as follows:

- Build and program the SSL client code in the STM32F4xx Flash
- Run the SSL server application on the remote PC, and run "**ssl_server.exe**" under Utilities\PC_Software\ssl_server. This application then waits for a client connection on https port 443
- Start the STM324xx -EVAL evaluation board
- Monitor the connection status in the SSL server application window and HyperTerminal window

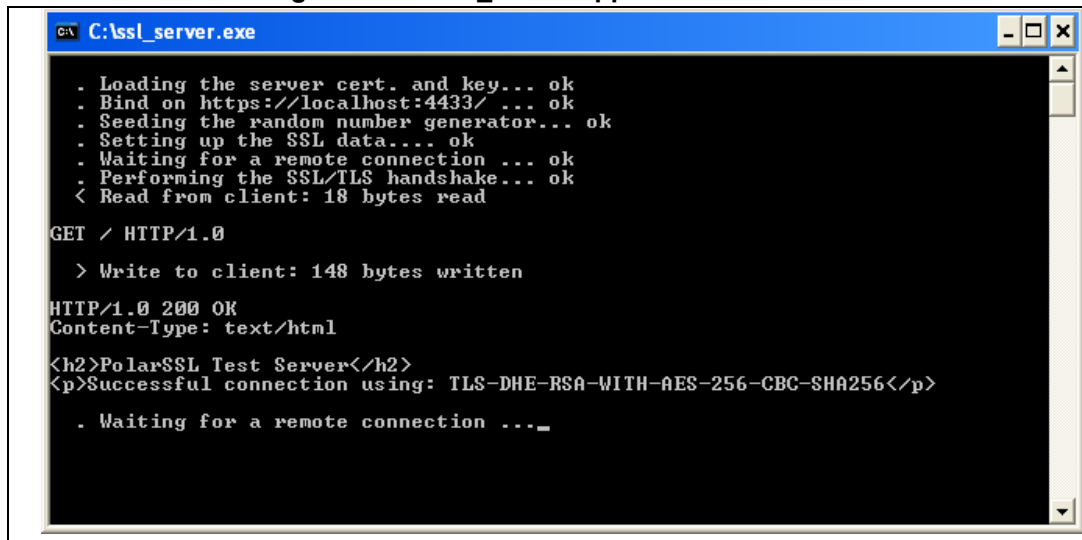
Note: Ensure that the remote PC IP address is the same address as defined in "ssl_client.c" file (`#define SSL_SERVER_NAME "192.168.0.1"`). If a firewall is used, the user must be sure that the "ssl_server" application accepts connection requests. If it does not, the firewall will reject the client requests.

Figure 7. SSL client application



The *ssl_server.exe* application window is shown in *Figure 8*. The SSL server application displays the connection request status; all exchanged messages between the server and the client, are displayed.

Figure 8. The *ssl_server* application window

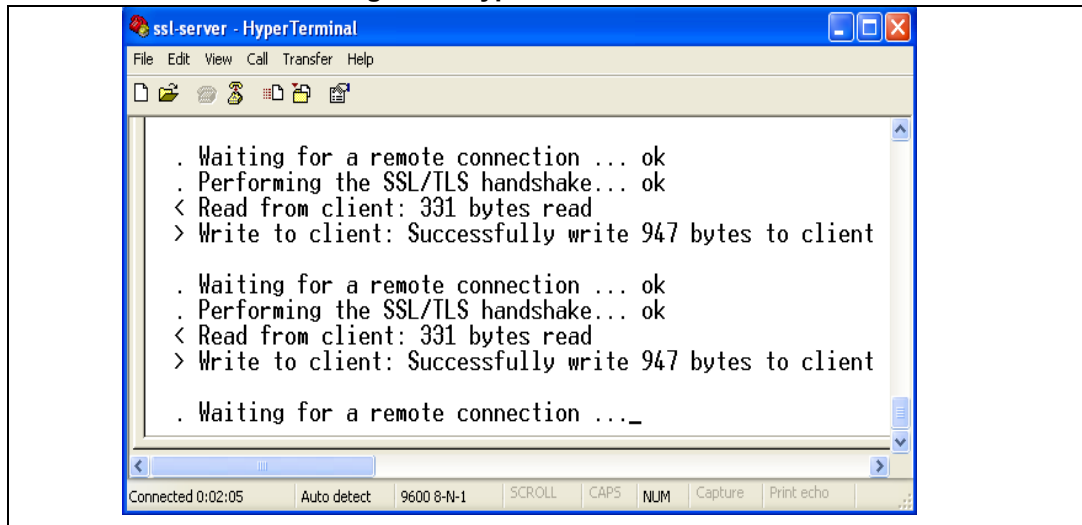


HyperTerminal

HyperTerminal window (*Figure 9*) displays the status of the SSL client application running on the STM32F4xx device (write and read messages):

- Status of SSL structures (SSL context, SSL session, SSL RNG)
- Client request to the server: "GET"
- The received message contains the result of Handshake protocol: for example "Successful connection using: SSL_EDH_RSA_AES_256_SHA".

Figure 9. HyperTerminal window



```
ssl-server - HyperTerminal
File Edit View Call Transfer Help
. Waiting for a remote connection ... ok
. Performing the SSL/TLS handshake... ok
< Read from client: 331 bytes read
> Write to client: Successfully write 947 bytes to client

. Waiting for a remote connection ... ok
. Performing the SSL/TLS handshake... ok
< Read from client: 331 bytes read
> Write to client: Successfully write 947 bytes to client

. Waiting for a remote connection ..._

Connected 0:02:05 Auto detect 9600 8-N-1 SCROLL CAPS NUM Capture Print echo
```

5.2 SSL server application

This demonstration consists of setting up the STM32 device as an SSL server, that waits for a SSL client request to make the connection.

Architecture of the application

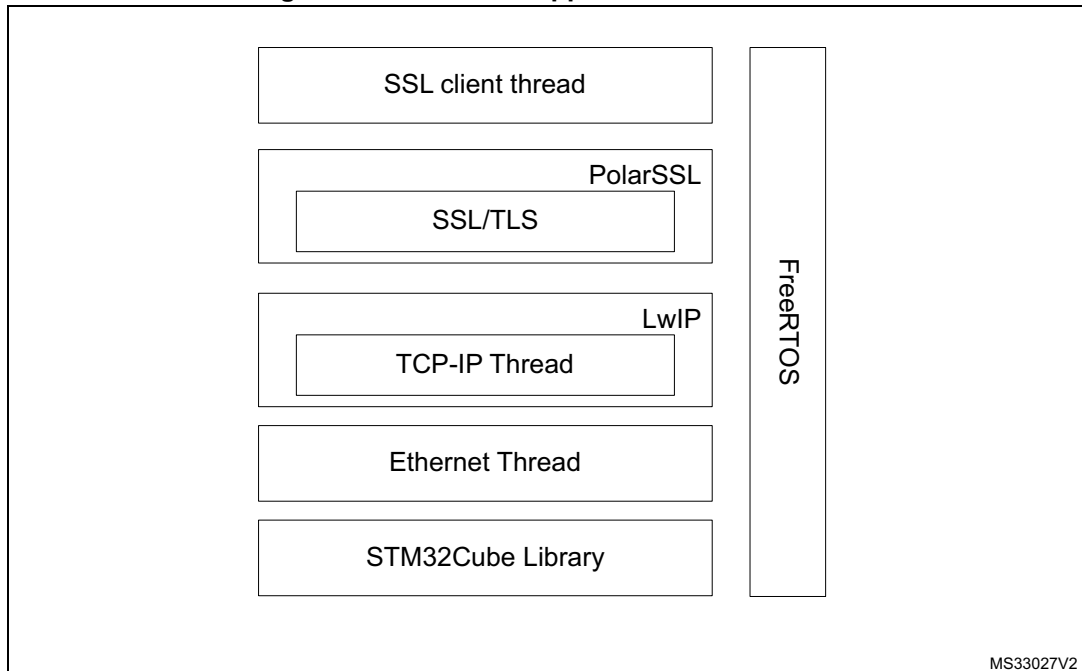
The SSL server demonstration contains six threads:

The **LED**, **Ethernet input**, **Ethernet link**, and **TCP_IP** threads are the same as the SSL client application threads.

SSL server thread: this thread creates an SSL connection and waits for the client request to make the secure connection. When the connection is established, the client sends Get request to load the html page. This page contains information about the tasks running in this demonstration. The SSL server task also sends the status of the connection through the USART.

DHCP_Client thread: This thread is used to configure the IP address by DHCP. To enable the DHCP client, uncomment the define `USE_DHCP` in `main.h` file.

Figure 10. SSL server application architecture



How to use the application

First, connect the STM324xx-EVAL evaluation board as follows:

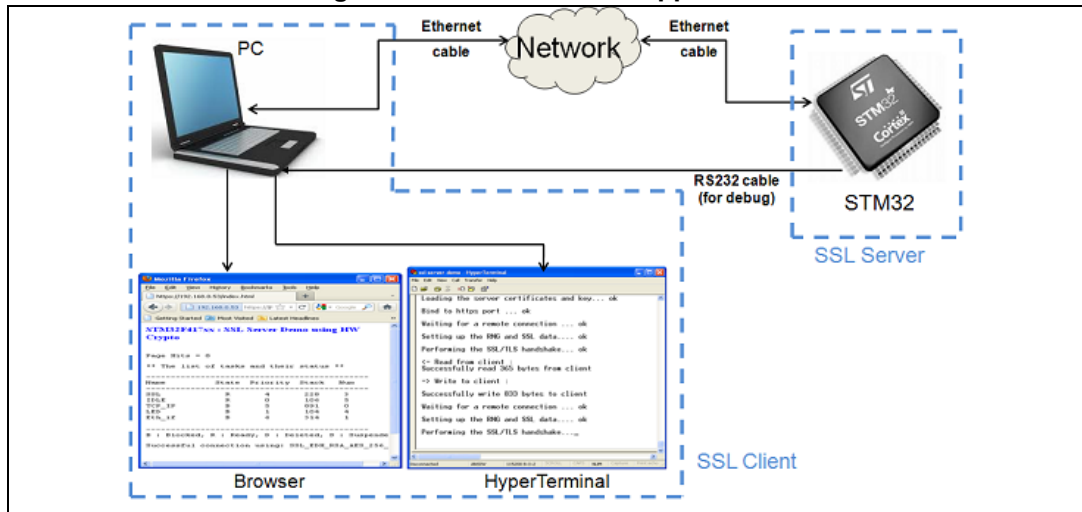
- Ethernet link: connect to a remote PC (through a crossover Ethernet cable) or to the local network (through a straight Ethernet cable)
- RS232 link (used with HyperTerminal-like application to display debug messages):

To run the SSL server demonstration:

- Build and program the SSL server code in the STM32F4xx Flash
- Start the STM324xx-EVAL board
- Open a web browser such as Internet Explorer or Firefox, and type **https://** followed by the IP address of the board in the browser, by default type "**https://192.168.0.10**"

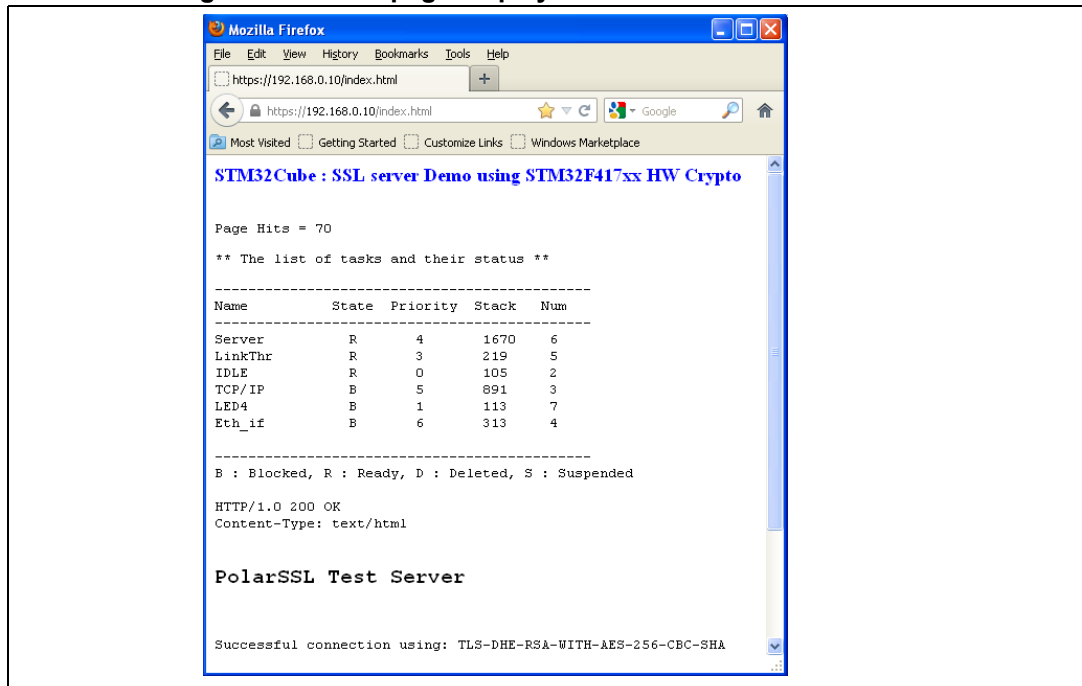
Note: If a firewall is present, user must be sure that the HTTPS port accepts the connection requests. If it does not, the firewall will reject the connection.

Figure 11. The SSL server application



On successful connection, a page is displayed showing the running tasks and their status. This page contains also the number of page hits and the list of cipher suites, used in the connection.

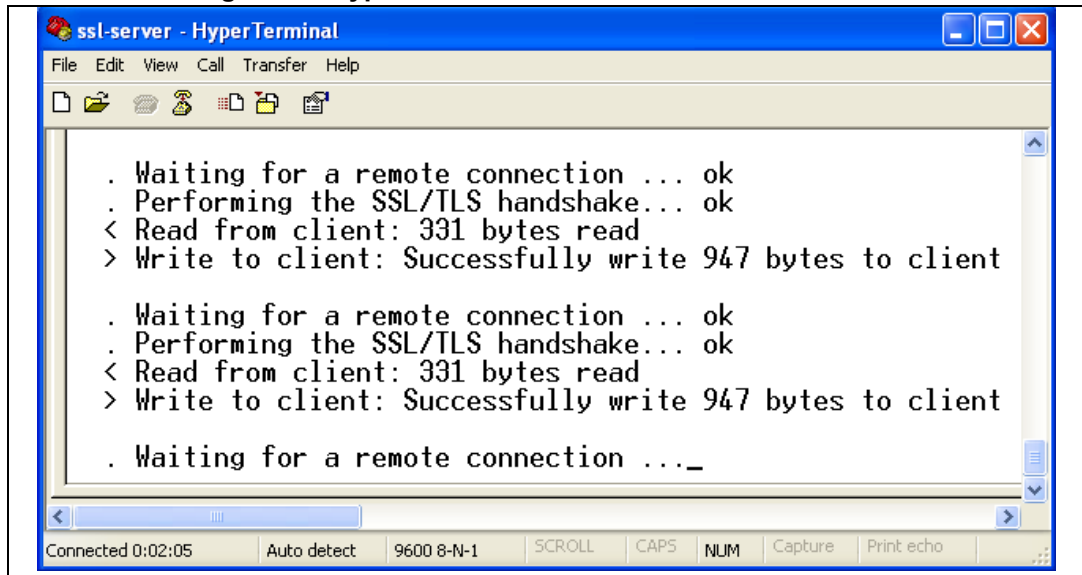
Figure 12. HTML page displayed on successful connection



The user can monitor the connection status of the SSL server application, running on STM32F4xx device, using the HyperTerminal window. This window (Figure 13) shows:

- The status of connection, SSL structures and Handshake protocol
- The size of the client request message
- The size of the server response (html page)

Figure 13. HyperTerminal SSL server connection status



Note: The first time that the user connects to the server, he receives a warning message from the browser about the certificate presented. This warning occurs when the certificate has been issued by a certification authority (CA), that is not recognized by the browser or when the certificate was issued to a different web address.

6 Conclusion

This user manual describes two STM32F4xx applications that implement the PolarSSL library with the STM32Cube drivers.

The first one demonstrates the ability of the STM32F4xx devices to exchange messages with a server through an SSL connection. This application allows the STM32 to connect to a secure web server.

The second one is a combination of HTTP with SSL protocol to provide encryption and secure identification of the server. This application allows the user to connect to an STM32 using the SSL protocol from a web browser.

7 FAQ

How to choose between static or dynamic (DHCP) IP address allocation?

When the macro `#define USE_DHCP` located in “main.h” is commented, a static IP address is assigned to the STM32 microcontroller (by default 192.168.0.10, this value can be modified from “main.h” file).

If the macro `#define USE_DHCP` is uncommented, the DHCP protocol is enabled, and the STM32 will act as a DHCP client

How the application behaves when the Ethernet cable is disconnected?

When the cable is disconnected the Ethernet peripheral stops both transmission and reception traffics, also the network interface will be set down. If an LCD controller is used a message is displayed to inform the user that the cable is not connected, else the Red LED of the evaluation board will turn on.

When the user connects again the cable, the Ethernet traffic will resume and network interface will be set up. If an LCD controller is used, a message is displayed to inform user about the new IP address, either with static or dynamic allocation, otherwise the Yellow LED of the evaluation board will turn on.

How to port the application on a different hardware?

When another hardware platform is used, the user has to check the GPIO configuration into the `HAL_ETH_MspInit()` function for the Ethernet peripheral, also `HAL_PPP_MspInit()` or `HAL_MspInit()` if the application needs more PPP peripheral.

Appendix A Additional information

Figure 14. SSL client thread flowchart

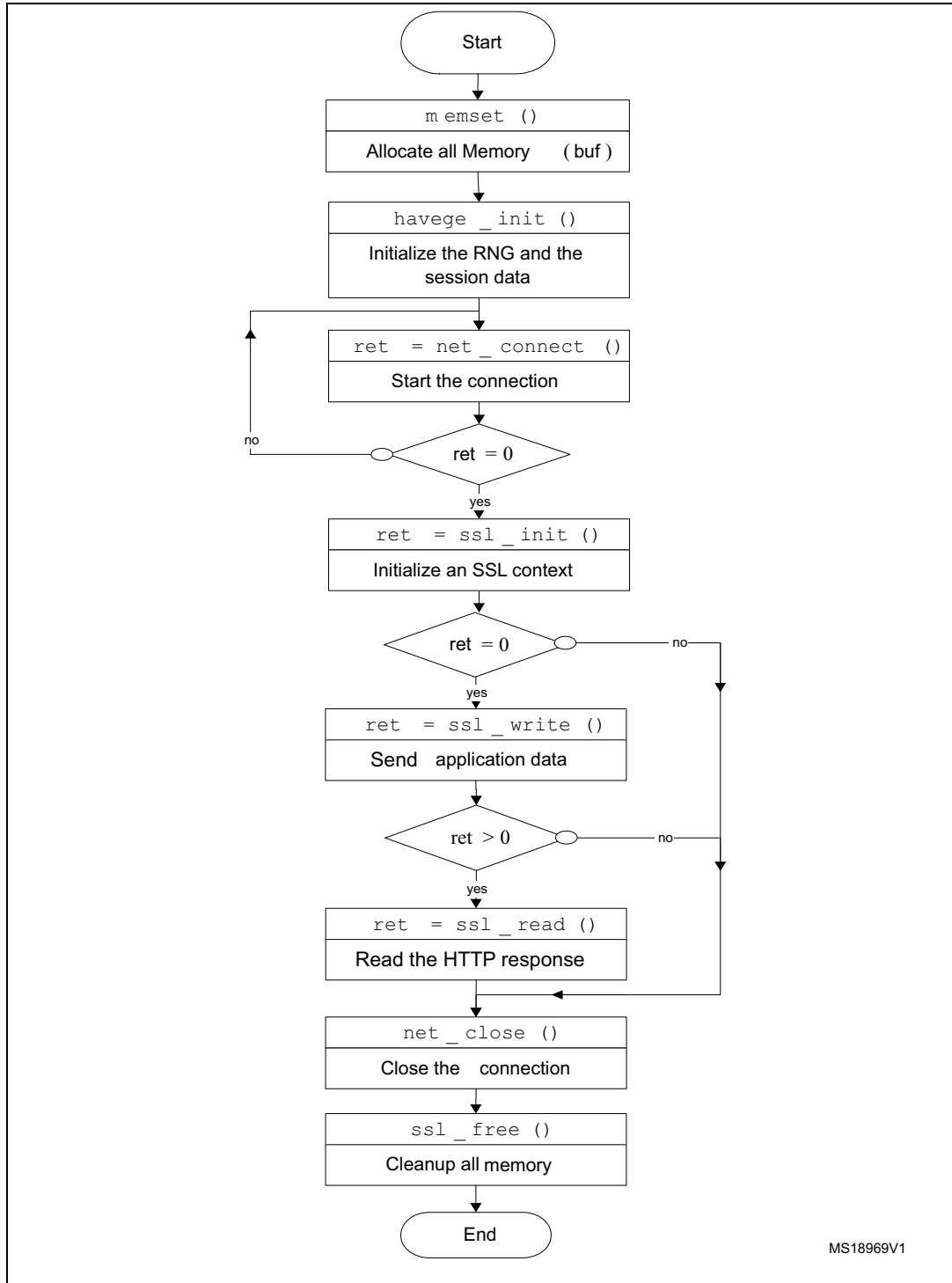
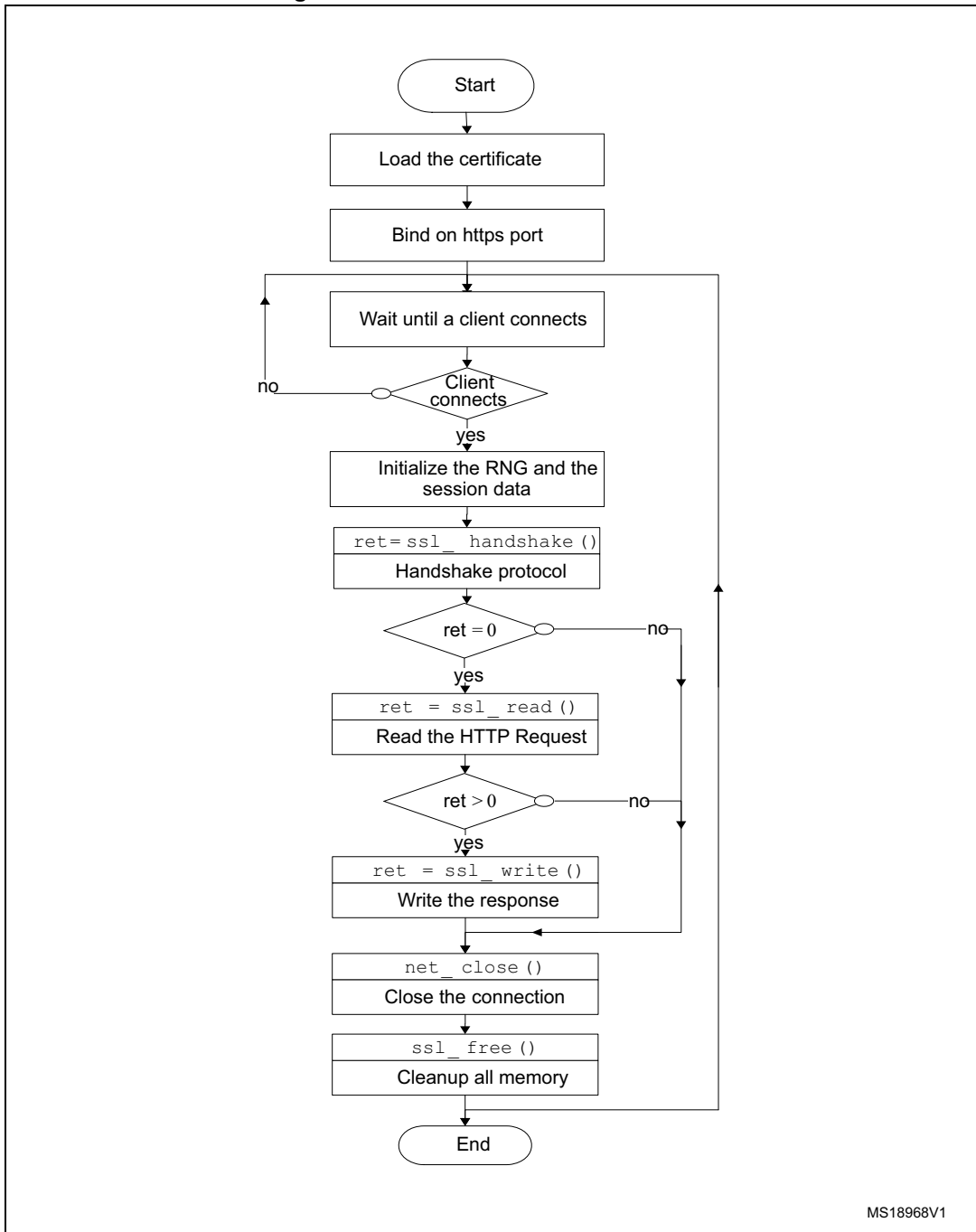


Figure 15. SSL server thread flowchart



MS18968V1

8 Revision history

Table 1. Document revision history

Date	Revision	Changes
27-Mar-2014	1	Initial release.
5-Jun-2015	2	Updated: <i>Section 3: STM32 hardware cryptography,</i> <i>Section 4: Description of the package,</i> <i>Section 4.2.1: PHY interface configuration</i> <i>Section 4.3: Evaluation boards settings,</i> <i>Section 5.1: SSL client application</i>

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.

© 2015 STMicroelectronics – All rights reserved