User's Manual

NetX™ Secure ECC

User Guide

Renesas Synergy™ Platform
Synergy Software
Synergy Software (SSP) Component

Renesas Electronics
www.renesas.com

Mar 4, 2019

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Renesas Synergy Specific Information

If you are using NetX Secure ECC for the Renesas Synergy platform, please use the following information.

Unsupported Features

The following ECC curves are not supported by Renesas SSP:

- P-224
- P-384
- P-521

Installation and Use of NetX Secure ECC

Page 1: If you are using Renesas Synergy SSP and the e^2 studio ISDE, NetX Secure ECC will already be installed. You can ignore the installation section.

Use Elliptic Curve Cryptography (ECC) in NetX Secure

Installation and Use of ECC

This chapter contains information related to installation, setup, and use of ECC crypto algorithms in NetX Secure TLS.

Product Distribution

In NetX Secure 5.11SP1, the ECC-crypto is distributed as a separate package. To install ECC files, user shall unzip the files into the NetX Secure source code directory.

The ECC-related files contain the keyword “ecc” in the file names. The following files are added to provide ECC functionality.

nx_secure_tls_ecc.h Header file for NetX Secure TLS for ECC
nx_secure_tls_ecc_*.c C Source files for NetX Secure TLS for ECC
nx_secure_x509_ecc_*.c C Source files for X.509 digital certificates.

Supported ECC curves

NetX Secure implements parts of the curves as per http://www.secg.org/sec2-v2.pdf. The following curves are supported:

- secp192r1
- secp224r1
- secp256r1
- secp384r1
- secp521r1

If other ECC curves are used, the *nx_secure_tls_session_start()* routine returns NX_SECURE_TLS_NO_SUPPORTED_CIPHERS for non-supported curves.

Note that TLS certificate chain may be encrypted by ECC-algorithms as well. Even though the curves provided by TLS client are supported, it is possible that the ECC curve used in the certificate chain is not supported.

In this case, *nx_secure_tls_session_start* routine returns NX_SECURE_TLS_UNSUPPORTED_PUBLIC_CIPHER.

Crypto Methods for ECC

Crypto methods for Elliptic Curve groups:

- `NX_CRYPTO_METHOD crypto_method_ec_secp192;`
- `NX_CRYPTO_METHOD crypto_method_ec_secp224;`
- `NX_CRYPTO_METHOD crypto_method_ec_secp256;`
- `NX_CRYPTO_METHOD crypto_method_ec_secp384;`
- `NX_CRYPTO_METHOD crypto_method_ec_secp521;`

The crypto methods for ECC curves are defined in nx_crypto_generic_ciphersuites.c

Crypto methods for ECDH and ECDHE:

- `NX_CRYPTO_METHOD crypto_method_ecdh;`
- `NX_CRYPTO_METHOD crypto_method_ecdhe;`

Crypto method for ECDSA:

- `NX_CRYPTO_METHOD crypto_method_ecdsa;`

ECDH and ECDSA crypto methods are defined in *nx_crypto_generic_ciphersuites.c.* ECDHE crypto methods are defined in *nx_crypto_generic_ciphersuites_ecc.c.*

These crypto methods are already defined. Combined with other crypto methods such as RSA, SHA, AES, they can be used as building blocks for the ciphersuite lookup table. The lookup table *_nx_crypto_ciphersuite_lookup_table_ecc* in nx_crypto_generic_ciphersuites_ecc.c can be used for TLS 1.2 connection.

Enable ECC in NetX Secure

By default ECC is not enabled in NetX Secure 5.11SP1. To add ECC support, the following changes must be bade in nx_secure_port.h:

(1) The symbol *NX_SECURE_ENABLE_ECC_CIPHERSUITE* must be defined in nx_secure_port.h.
(2) The header file *nx_secure_tls_ecc.h* must be included in nx_secure_port.h.

For the change to take effect, user shall rebuild NetX Secure Library, and all applications that use the library.

In the application code, the API n*x_secure_tls_ecc_initialize()* must be called after TLS session is created . This API notifies the TLS session of the type of curves used in the system. During the TLS handshake phase, if ECC algorithm is selected, the client and server exchange ECC curve-related parameters, so ECC can be used for the session.

The following code segment illustrates how to use the API. Note that the arguments (*nx_crypto_ecc_supported_groups, nx_crypto_ecc_supported_groups_size, and nx_crypto_ecc_curves)* are all defined in *nx_crypto_generic_ciphersuites_ecc.c*. Therefore these symbols can be used directly.

```
status = nx_secure_tls_ecc_initialize(&tls_session,
                  nx_crypto_ecc_supported_groups,
                  nx_crypto_ecc_supported_groups_size,
                  nx_crypto_ecc_curves);
```

The TLS Crypto table in nx_crypto_generic_ciphersuites_ecc.c contains ECC-class ciphersuites lookup table. Thereforfe TLS session wishing to use ECC shall use `nx_crypto_tls_ciphers_ecc` when creating TLS Sessions. The TLS_Crypto table defined in nx_crypto_generic_ciphersuites.c contains non-ECC ciphersuites.

Known Limitations

The following known limitations are in the scope of ECC for TLS.

- SHA384 or SHA512 are not supported except for verification of certificate signature.
- In ServerKeyExchange of the ECDHE ciphersuites, only SHA1 and SHA256 are supported for the signature hash.
- TLS server does not support dynamic certificate selection when there are multiple certificates in the local store.
- X509 Certificate KeyUsage is not observed.
- ECDSA_fixed_ECDH, RSA_fixed_ECDH or ECDH_anon are not supported.

Configuration Options

There are several configuration options for building NetX Secure. Following is a list of all options, where each is described in detail:

Define	Meaning
NX_SECURE_ENABLE_ECC_CIPHERSUITE	Defined, this option enables the ECC support in TLS. By default this symbol is not defined.

TLS Client Example

The following example uses ECC for TLS client application. This demo is designed to work with the OpenSSL reverse-echo server (openssl s_server -rev). For simplicity, in this example API calls are assumed to be successful, and return values are not checked.

```
#include "tx_api.h"
#include "nx_api.h"
#include "nx_secure_tls_api.h"

/* Define the size of our application stack. */
#define     DEMO_STACK_SIZE             4096

/* Define the remote server IP address using NetX IP_ADDRESS macro. */
#define     REMOTE_SERVER_IP_ADDRESS    IP_ADDRESS(192, 168, 1, 1)

/* Define the remote server port. 443 is the HTTPS default. */
#define     REMOTE_SERVER_PORT          443

/* Define the ThreadX and NetX object control blocks...  */

NX_PACKET_POOL          pool_0;
NX_IP                   ip_0;
NX_TCP_SOCKET tcp_socket;
NX_SECURE_TLS_SESSION tls_session;
NX_SECURE_X509_CERTIFICATE tls_certificate;

/* Define space for remote certificate storage. There must be one certificate
structure and it's associated buffer for each expected certificate from the remote
host. If you expect 3 certificates, you will need 3 structures and 3 buffers. The
buffers must be large enough to hold the incoming certificate data (2KB is usually
sufficient but large RSA keys can push the size beyond that). */
NX_SECURE_X509_CERTIFICATE remote_certificate;
NX_SECURE_X509_CERTIFICATE remote_issuer_certificate;
UCHAR remote_certificate_buffer[2000];
UCHAR remote_issuer_buffer[2000];

/* Define an HTTP request to be sent to the HTTPS web server. */
UCHAR http_request[] = { … };

/* Define the IP thread's stack area.  */
ULONG ip_thread_stack[3 * 1024 / sizeof(ULONG)];

/* Define packet pool for the demonstration.  */
#define NX_PACKET_POOL_SIZE ((1536 + sizeof(NX_PACKET)) * 32)

ULONG packet_pool_area[NX_PACKET_POOL_SIZE/sizeof(ULONG) + 64 / sizeof(ULONG)];

/* Define the ARP cache area.  */
ULONG arp_space_area[512 / sizeof(ULONG)];

/* Define the TLS Client thread.  */
ULONG             tls_client_thread_stack[6 * 1024 / sizeof(ULONG)];
TX_THREAD         tls_client_thread;
void              client_thread_entry(ULONG thread_input);
```

```
/* Define the TLS packet reassembly buffer. */
UCHAR tls_packet_buffer[4000];

/* Define the metadata area for TLS cryptography. The actual size needed can be
   Ascertained by calling nx_secure_tls_metadata_size_calculate.
*/
UCHAR tls_crypto_metadata[18000];

/* Pointer to the TLS ciphersuite table that is included in the platform-specific
   cryptography subdirectory. The table maps the cryptographic routines for the
   platform to function pointers usable by the TLS library.
*/
extern const NX_SECURE_TLS_CRYPTO nx_crypto_tls_ciphers_ecc;
extern const USHORT nx_crypto_ecc_supported_groups[];
extern const NX_CRYPTO_METHOD *nx_crypto_ecc_curves[];
extern const UINT nx_crypto_ecc_supported_groups_size;

/* Binary data for the TLS Client X.509 trusted root CA certificate, ASN.1 DER-
   encoded. A trusted certificate must be provided for TLS Client applications
   (unless X.509 authentication is disabled) or TLS will treat all certificates as
   untrusted and the handshake will fail.
*/
const UCHAR trusted_ca_data[] = { … }; /* DER-encoded binary certificate. */
const UINT trusted_ca_length[] = 0x574;

/* Define the application - initialize drivers and TCP/IP setup.  */
void    tx_application_define(void *first_unused_memory)
{
UINT  status;

    /* Initialize the NetX system.  */
    nx_system_initialize();

    /* Create a packet pool. Check status for errors. */
    status =  nx_packet_pool_create(&pool_0, "NetX Main Packet Pool", 1536,
                                    (ULONG*)(((int)packet_pool_area + 64) & ~63) ,
                                    NX_PACKET_POOL_SIZE);

    /* Create an IP instance for the specific target. Check status for errors. */
    status = nx_ip_create(&ip_0, …);

    /* Enable ARP and supply ARP cache memory for IP Instance 0. Check status for
       errors. */
    status =  nx_arp_enable(&ip_0, (void *)arp_space_area, sizeof(arp_space_area));

    /* Enable TCP traffic. Check status for errors. */
    status =  nx_tcp_enable(&ip_0);

    status =  nx_ip_fragment_enable(&ip_0);

    /* Initialize the NetX Secure TLS system.  */
    nx_secure_tls_initialize();

    /* Create the TLS client thread to start handling incoming requests. */
    tx_thread_create(&tls_client_thread, "TLS Server thread", client_thread_entry, 0,
                     tls_client_thread_stack, sizeof(tls_client_thread_stack),
                     16, 16, 4, TX_AUTO_START);
}

/* Thread to handle the TLS Client instance. */
void client_thread_entry(ULONG thread_input)
{
UINT       status;
NX_PACKET *send_packet;
NX_PACKET *receive_packet;
UCHAR receive_buffer[100];
ULONG bytes;
ULONG server_ipv4_address;

    /* We are not using the thread input parameter so suppress compiler warning. */
    NX_PARAMETER_NOT_USED(thread_input);
```

```
/* Ensure the IP instance has been initialized.  */
status =  nx_ip_status_check(&ip_0, NX_IP_INITIALIZE_DONE, &actual_status,
                             NX_IP_PERIODIC_RATE);

/* Create a TCP socket to use for our TLS session.  */
status =  nx_tcp_socket_create(&ip_0, &tcp_socket, "TLS Client Socket",
                               NX_IP_NORMAL, NX_FRAGMENT_OKAY,
                               NX_IP_TIME_TO_LIVE,
                               8192, NX_NULL, NX_NULL);

/* Create a TLS session for our socket. This sets up the TLS session object for
   later use */
status =  nx_secure_tls_session_create(&tls_session,
                                       &nx_crypto_tls_ciphers,
                                       tls_crypto_metadata,
                                       sizeof(tls_crypto_metadata));

/* Initialize ECC parameters for this session. */
status = nx_secure_tls_ecc_initialize(&tls_session,
                                      nx_crypto_ecc_supported_groups,
                                      nx_crypto_ecc_supported_groups_size,
                                      nx_crypto_ecc_curves);

/* Set the packet reassembly buffer for this TLS session. */
status = nx_secure_tls_session_packet_buffer_set(&tls_session, tls_packet_buffer,
                                                 sizeof(tls_packet_buffer));

/* Initialize an X.509 certificate with our CA root certificate data. */
nx_secure_x509_certificate_initialize(&certificate, trusted_ca_data,
                                      trusted_ca_length, NX_NULL, 0, NX_NULL, 0,
                                      NX_SECURE_X509_KEY_TYPE_NONE);

/* Add the initialized certificate as a trusted root certificate. */
nx_secure_tls_trusted_certificate_add(&tls_session, &certificate);

/* The remote server will be sending one or more certificates so we need to
   allocate space to receive and process them. Assume the server will provide at
   least an identity certificate and an intermediate CA issuer. */
nx_secure_tls_remote_certificate_allocate(&tls_session, &remote_certificate,
  remote_certificate_buffer,
  sizeof(remote_certificate_buffer));
nx_secure_tls_remote_certificate_allocate(&tls_session,
                                          &remote_issuer_certificate,
                                          remote_issuer_buffer,
                                          sizeof(remote_issuer_buffer));

/* Setup this thread to open a connection on the TCP socket to a remote server.
   The IP address can be used directly or it can be obtained via DNS or other
   means.*/
server_ipv4_address = REMOTE_SERVER_IP_ADDRESS;
status = nx_tcp_client_socket_connect(&tcp_socket, server_ipv4_address,
                                      REMOTE_SERVER_PORT, NX_WAIT_FOREVER);

/* Start the TLS Session using the connected TCP socket. This function will
   ascertain from the TCP socket state that this is a TLS Client session. */
status = nx_secure_tls_session_start(&tls_session, &tcp_socket,
                                     NX_WAIT_FOREVER);

/* Allocate a TLS packet to send an HTTP request over TLS (HTTPS). */
status = nx_secure_tls_packet_allocate(&tls_session, &pool_0, &send_packet,
                                       NX_TLS_PACKET, NX_WAIT_FOREVER);

/* Populate the packet with our HTTP request. */
nx_packet_data_append(send_packet, http_request, strlen(http_request), &pool_0,
                      NX_WAIT_FOREVER);
```

```
    /* Send the HTTP request over the TLS Session, turning it into HTTPS. */
    status = nx_secure_tls_session_send(&tls_session, send_packet, NX_WAIT_FOREVER);

    /* Check for errors…  */
    if (status != NX_SUCCESS)
    {
          /* Release the packet since the packet was not sent.  */
          nx_packet_release(send_packet);
    }

    /* Receive the HTTP response and any data from the server. */
    status = nx_secure_tls_session_receive(&tls_session, &receive_packet,
                                           NX_WAIT_FOREVER);

    /* Extract the data we received from the remote server. */
    status = nx_packet_data_extract_offset(receive_packet, 0, receive_buffer, 100,
                                           &bytes);
    /* Display the response data. */
    receive_buffer[bytes] = 0;
    printf("Received data: %s\n", receive_buffer);

    /* End the TLS session now that we have received our HTTPS/HTML response. */
    status = nx_secure_tls_session_end(&tls_session, NX_WAIT_FOREVER);

    /* Check for errors to make sure the session ended cleanly. */

    /* Disconnect the TCP socket. */
    status =  nx_tcp_socket_disconnect(&tcp_socket, NX_WAIT_FOREVER);

}
```

Figure 1.1 Example of using ECC for TLS Client Application

TLS Server Example (HTTPS Web Server)

The following example uses ECC for TLS server application. This example demonostrates a simple TLS Web Server (HTTPS). For simplicity, in this example API calls are assumed to be successful, and return values are not checked.

```
#include "tx_api.h"
#include "nx_api.h"
#include "nx_secure_tls_api.h"

#define     DEMO_STACK_SIZE         4096

/* Define the ThreadX and NetX object control blocks...  */

NX_PACKET_POOL          pool_0;
NX_IP                   ip_0;
NX_TCP_SOCKET tcp_socket;
NX_SECURE_TLS_SESSION tls_session;
NX_SECURE_X509_CERTIFICATE tls_certificate;

/* Define the IP thread's stack area.  */
ULONG ip_thread_stack[3 * 1024 / sizeof(ULONG)];

/* Define packet pool for the demonstration.  */
#define NX_PACKET_POOL_SIZE ((1536 + sizeof(NX_PACKET)) * 32)

ULONG packet_pool_area[NX_PACKET_POOL_SIZE/sizeof(ULONG) + 64 / sizeof(ULONG)];

/* Define the ARP cache area.  */
ULONG arp_space_area[512 / sizeof(ULONG)];
```

```
/* Define the TLS Server thread.  */
ULONG             tls_server_thread_stack[6 * 1024 / sizeof(ULONG)];
TX_THREAD         tls_server_thread;
void              server_thread_entry(ULONG thread_input);

/* Define the TLS packet reassembly buffer. */
UCHAR tls_packet_buffer[4000];

/* Define the metadata area for TLS cryptography. The actual size needed can be
   Ascertained by calling nx_secure_tls_metadata_size_calculate.
*/
UCHAR tls_crypto_metadata[18000];

/* Pointer to the TLS ciphersuite table that is included in the platform-specific
   cryptography subdirectory. The table maps the cryptographic routines for the
   platform to function pointers usable by the TLS library.
*/
extern const NX_SECURE_TLS_CRYPTO nx_crypto_tls_ciphers_ecc;
extern const USHORT nx_crypto_ecc_supported_groups[];
extern const NX_CRYPTO_METHOD *nx_crypto_ecc_curves[];
extern const UINT nx_crypto_ecc_supported_groups_size;

/* Binary data for the TLS Server X.509 certificate, ASN.1 DER-encoded. */
const UCHAR certificate_data[] = { … }; /* DER-encoded binary certificate. */
const UINT certificate_length[] = 0x574;

/* Binary data for the TLS Server RSA Private Key, from private key
   file generated at the time of the X.509 certificate creation. ASN.1 DER-encoded.
*/
const UCHAR private_key[] = { … }; /* DER-encoded RSA private key file (PKCS#1) */
const UINT private_key_length = 0x40;

/* Define some HTML data (web page) with an HTTPS header to serve to connecting
clients. */
const UCHAR html_data[] = { … };

/* Define the application – initialize drivers and TCP/IP setup.  */
void    tx_application_define(void *first_unused_memory)
{
UINT  status;


    /* Initialize the NetX system.  */
    nx_system_initialize();

    /* Create a packet pool. Check status for errors. */
    status =  nx_packet_pool_create(&pool_0, "NetX Main Packet Pool", 1536,
                                    (ULONG*)(((int)packet_pool_area + 64) & ~63) ,
                                    NX_PACKET_POOL_SIZE);

    /* Create an IP instance for the specific target. Check status for errors. */
    status = nx_ip_create(&ip_0, …);

    /* Enable ARP and supply ARP cache memory for IP Instance 0. Check status for
       errors. */
    status =  nx_arp_enable(&ip_0, (void *)arp_space_area, sizeof(arp_space_area));

    /* Enable TCP traffic. Check status for errors. */
    status =  nx_tcp_enable(&ip_0);

    status =  nx_ip_fragment_enable(&ip_0);

    /* Initialize the NetX Secure TLS system.  */
    nx_secure_tls_initialize();

    /* Create the TLS server thread to start handling incoming requests. */
    tx_thread_create(&tls_server_thread, "TLS Server thread", server_thread_entry, 0,
                     tls_server_thread_stack, sizeof(tls_server_thread_stack),
                     16, 16, 4, TX_AUTO_START);
}
```

```
 /* Thread to handle the TLS Server instance. */
void server_thread_entry(ULONG thread_input)
{
UINT       status;
NX_PACKET *send_packet;
NX_PACKET *receive_packet;
UCHAR receive_buffer[100];
ULONG bytes;

    NX_PARAMETER_NOT_USED(thread_input);

    /* Ensure the IP instance has been initialized.  */
    status =  nx_ip_status_check(&ip_0, NX_IP_INITIALIZE_DONE, &actual_status,
                                 NX_IP_PERIODIC_RATE);

    /* Create a TCP socket to use for our TLS session.  */
    status =  nx_tcp_socket_create(&ip_0, &tcp_socket, "TLS Server Socket",
                                   NX_IP_NORMAL, NX_FRAGMENT_OKAY,
NX_IP_TIME_TO_LIVE,
                                   8192, NX_NULL, NX_NULL);

    /* Create a TLS session for our socket.  */
    status =  nx_secure_tls_session_create(&tls_session,
                                           &nx_crypto_tls_ciphers,
                                           tls_crypto_metadata,
                                           sizeof(tls_crypto_metadata));

    status = nx_secure_tls_ecc_initialize(&tls_session,
                                          nx_crypto_ecc_supported_groups,
                                          nx_crypto_ecc_supported_groups_size,
                                          nx_crypto_ecc_curves);

    /* Check status for errors...  */

    /* Set the packet reassembly buffer for this TLS session. */
    status = nx_secure_tls_session_packet_buffer_set(&tls_session, tls_packet_buffer,
                                                     sizeof(tls_packet_buffer));

    /* Initialize an X.509 certificate and private RSA key for our TLS Session. */
    nx_secure_x509_certificate_initialize(&certificate, certificate_data, NX_NULL, 0,
                                          certificate_length, private_key,
                                          private_key_length);

    /* Add the initialized certificate as a local identity certificate. */
    nx_secure_tls_add_local_certificate(&tls_session, &certificate);


    /* Setup this thread to listen on the TCP socket.
       Port 443 is standard for HTTPS. */
    status =  nx_tcp_server_socket_listen(&ip_0, 443, &tcp_socket, 5, NX_NULL);

    while(1) {
        /* Accept a client TCP socket connection.  */
        status =  nx_tcp_server_socket_accept(&tcp_socket, NX_WAIT_FOREVER);

        /* Check for errors… */

        /* Start the TLS Session using the connected TCP socket. */
        status = nx_secure_tls_session_start(&tls_session, &tcp_socket,
                                             NX_WAIT_FOREVER);

        /* Receive the HTTPS request. */
        status = nx_secure_tls_session_receive(&tls_session, &receive_packet,
                                               NX_WAIT_FOREVER);

        /* Extract the HTTP request information from the HTTPS request. */
        status = nx_packet_data_extract_offset(receive_packet, 0, receive_buffer,
        100,
                                               &bytes);
        /* Display the HTTP request data. */
```

```
        receive_buffer[bytes] = 0;
        printf("Received data: %s\n", receive_buffer);

        /* Allocate a TLS packet to send HTML data back to client. */
        status = nx_secure_tls_packet_allocate(&tls_session, &pool_0, &send_packet,
                                               NX_TLS_PACKET, NX_WAIT_FOREVER);

        /* Populate the packet with our HTTP response and HTML web page data. */
        nx_packet_data_append(send_packet, html_data, strlen(html_data), &pool_0,
                              NX_WAIT_FOREVER);

        /* Send the HTTP response over the TLS Session, turning it into HTTPS. */
        status = nx_secure_tls_session_send(&tls_session, send_packet,
                                            NX_WAIT_FOREVER);

        /* Check for errors…  */
        if (status != NX_SUCCESS)
        {
              /* Release the packet since it was not sent.  */
              nx_packet_release(send_packet);
        }

        /* End the TLS session now that we have sent our HTTPS/HTML response. */
        status = nx_secure_tls_session_end(&tls_session, NX_WAIT_FOREVER);

        /* Check for errors to make sure the session ended cleanly. */

        /* Disconnect the TCP socket so we can be ready for the next request. */
        status =  nx_tcp_socket_disconnect(&tcp_socket, NX_WAIT_FOREVER);

        /* Unaccept the server socket.  */
        status =  nx_tcp_server_socket_unaccept(&tcp_socket);

        /* Setup server socket for listening again.  */
        status =  nx_tcp_server_socket_relisten(&ip_0, 443, &tcp_socket);
    }
}
```

Figure 1.2 Example of NetX Secure use with NetX

NetX™ Secure ECC User Guide

Publication Date: Rev.1.00 Mar 4, 2019

Published by: Renesas Electronics Corporation

NetX™ Secure ECC User Guide

Renesas Electronics Corporation

R11UM0125EU0100