AN1083: Creating and Using a Secure CoAP Connection with ARM's mbed TLS

This document describes how to create and use a secure Constrained Application Protocol (CoAP) connection with ARM's mbed TLS library (mbed TLS).

Silicon Labs has implemented Secure CoAP functionality on two different TLS (Transport Layer Security) code bases: the Silicon Labs custom implementation and ARM's mbed TLS. Refer to *UG278: ZCL/IP User's Guide* for more information on the Silicon Labs custom TLS implementation.

This document assumes you have a solid understanding of ARM mbed TLS and its security features. If not, consult https://www.mbed.com/en/technologies/security/.

KEY FEATURES

- Introduces ARM mbed TLS.
- Describes the system requirements, software contents, and documentation.
- Provides detailed instructions for using Secure CoAP with ARM's mbed TLS library

1 Introduction

Beginning with Silicon Labs Thread release 2.3, users who are developing for any device with >256 kB data storage, such as the EFR32xG12, have the option of using ARM mbed TLS for communications security. ARM mbed TLS is a widely-used and highly configurable implementation that supports additional features, including:

- SSL version 3
- Variety of key exchange methods (RSA, DHE-RSA, ECDHE-RSA, ECDH-RSA, PSK, DHE-PSK, and others)
- Variety of cryptographic algorithms (AES, 3DES, DES, XTEA, and others)
- Various modes of operations (CBC, EBC, CFB, and others)
- Various hash algorithms (MD2, MD4, SHA-1, SHA-256, and others)
- ECC support including ECDHE, ECDSA and ECJPAKE, among others
- NIST standardized CTR_DRBG and HMAC_DRBG random number generators
- Supports a variety of underlying technologies – X.509 certificates; RSA and ECC private and public key reading

For more information on ARM mbed TLS, see https://tls.mbed.org/core-features and https://tls.mbed.org/standard-compliance. For more information on using the Silicon Labs custom TLS implementation, see *UG278: ZCL/IP User's Guide.*

2 System Requirements, Software Contents, and Documentation

2.1 Software Requirements

- Silicon Labs Thread release 2.3 and above
- Simplicity Studio v4 and above
- Linux or other UNIX (e.g., Mac OS X) environment
- Plugins control whether Silicon Labs TLS or ARM mbed TLS is used in your application (see *UG278: ZCL/IP User's Guide* for more information)

2.2 mbed TLS Configuration Header File

If you're building a Thread stack with the mbed TLS library, the mbed TLS library is built using the configuration file header file named config.h. You can find this file in this location:

<stack install location>/ip/tls/mbedtls/

Note: If you are building an application using mbed TLS, Silicon Labs recommends that you use the standard config.h shipped with the stack because it has Thread commissioning and dotdot support out-of-the-box. If you want to build a custom mbed TLS library, Silicon Labs recommends that they start with the stack-config.h that is shipped with the stack because it already includes the required #defines for Thread commissioning.

2.3 mbed TLS Sample Programs

You can find the mbed TLS sample programs in these locations:

- <stack install location>/util/third_party/mbedtls/programs
- <stack install location>/util/third_party/mbedtls/tests

2.4 API Documentation

API documentation is generated by Doxygen from the header files of the mbed TLS library. The API Reference clarifies the structures, decisions, and code constructs. It is included in the mbed TLS configuration file header file (config.h).

2.5 README File

The mbed TLS README file describes how to complete some basic tasks using the mbed TLS library. The README file is named README.md. You can find this file in this location:

<stack install location>/util/third_party/mbedtls

3 Using Secure CoAP

3.1 Enabling mbed TLS and Using the DTLS-CLI Plugin

Refer to *UG278: ZCL/IP User's Guide* for the steps to enable mbed TLS and for information on using the DTLS-CLI plugin.

3.2 Using the mbed TLS Configuration Header File

Follow these steps if you are building an application using mbed TLS.

1. Enable the following configuration #defines for Thread commissioning to work properly (see stack-config.h).

For the Wireless Gecko (EFR32™) Portfolio only:

```
#define MBEDTLS_AES_ALT
#define MBEDTLS_ECP_ALT
#define MBEDTLS_ECP_DEVICE_ALT
#define MBEDTLS_ECP_DOUBLE_JAC_ALT
#define MBEDTLS_ECP_DEVICE_ADD_MIXED_ALT
#define MBEDTLS_ECP_NORMALIZE_JAC_ALT
#define MBEDTLS_MPI_MODULAR_DIVISION_ALT
#define MBEDTLS_SHA256_ALT
```

Otherwise:

```
#define MBEDTLS_NO_PLATFORM_ENTROPY
```

General Stack Configuration:

```
#define MBEDTLS_HAVE_HAL_WATCHDOG_TIMER
#define MBEDTLS_HAVE_ASM
#define MBEDTLS_PLATFORM_MEMORY
#define MBEDTLS_AES_ROM_TABLES
#define MBEDTLS_CIPHER_MODE_CBC
#define MBEDTLS_ECP_DP_SECP256R1_ENABLED
#define MBEDTLS_ECP_NIST_OPTIM
#define MBEDTLS_KEY_EXCHANGE_ECJPAKE_ENABLED
#define MBEDTLS_SHA256_SMALLER
#define MBEDTLS_SSL_MAX_FRAGMENT_LENGTH
#define MBEDTLS_SSL_PROTO_TLS1_2
#define MBEDTLS_SSL_PROTO_DTLS
#define MBEDTLS_SSL_DTLS_ANTI_REPLAY
#define MBEDTLS_SSL_DTLS_HELLO_VERIFY
#define MBEDTLS_SSL_EXPORT_KEYS
#define MBEDTLS_AES_C
#define MBEDTLS_ASN1_PARSE_C
#define MBEDTLS_ASN1_WRITE_C
#define MBEDTLS_BASE64_C
#define MBEDTLS_BIGNUM_C
#define MBEDTLS_CCM_C
#define MBEDTLS_CIPHER_C
#define MBEDTLS_CTR_DRBG_C
#define MBEDTLS_ECJPAKE_C
#define MBEDTLS_ECP_C
#define MBEDTLS_ENTROPY_C
#define MBEDTLS_HMAC_DRBG_C
#define MBEDTLS_MD_C
#define MBEDTLS_NET_C
#define MBEDTLS_OID_C
#define MBEDTLS_PK_C
#define MBEDTLS_PLATFORM_C
#define MBEDTLS_SHA256_C
#define MBEDTLS_SSL_COOKIE_C
#define MBEDTLS_SSL_CLI_C
#define MBEDTLS_SSL_SRV_C
#define MBEDTLS_SSL_TLS_C
```

```
#define MBEDTLS_PLATFORM_CALLOC_MACRO      umm_calloc
#define MBEDTLS_PLATFORM_FREE_MACRO          umm_free
```

Recommended:

```
#define MBEDTLS_MPI_MAX_SIZE                32
#define MBEDTLS_ECP_MAX_BITS               256
#define MBEDTLS_ECP_WINDOW_SIZE              2
#define MBEDTLS_SSL_MAX_CONTENT_LEN       1024
#define MBEDTLS_SSL_CIPHERSUITES MBEDTLS_TLS_ECJPAKE_WITH_AES_128_CCM_8
```

2. (Optional) Enable the following configuration #defines for X.509 certificate-based secure sessions on dotdot applications:

```
#define MBEDTLS_KEY_EXCHANGE_ECDHE_ECDSA_ENABLED
#define MBEDTLS_ECDH_C
#define MBEDTLS_ECDSA_C
#define MBEDTLS_CERTS_C
#define MBEDTLS_X509_USE_C
#define MBEDTLS_X509_CRT_PARSE_C
#define MBEDTLS_SSL_CIPHERSUITES          \
   MBEDTLS_TLS_ECJPAKE_WITH_AES_128_CCM_8, \
   MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
```

3.3 Setting a Certificate on the Device

Set a certificate on the device using the following API:

```
void emberSetDtlsDeviceCertificate(const CertificateAuthority **certAuthority, const DeviceCertifi-
cate *deviceCert);
```

The callback for this is:

```
void emberSetDtlsDeviceCertificateReturn(uint32_t result);
```

3.4 Creating a Secure CoAP Session on an Application

1. Create a secure CoAP session using the following API. Pass in the `remoteAddress` of the peer with which you want to establish a connection.

Note: For dotdot, the local and remote ports are `EMBER_COAP_SECURE_PORT` (5684).

```
void emberOpenDtlsConnection(const EmberIpv6Address *remoteAddress,
                             uint16_t localPort,
                             uint16_t remotePort);
```

2. Check whether the request for the secure session succeeded by looking for this callback:

```
void emberOpenDtlsConnectionReturn(uint32_t result,
                                   const EmberIpv6Address *remoteAddress,
                                   uint16_t localPort,
                                   uint16_t remotePort);
```

The following callback should be expected on both devices which confirms that a secure session was successfully established.

```
void emberDtlsSecureSessionEstablished(uint8_t flags,
                                       uint8_t sessionId,
                                       const EmberIpv6Address *localAddress,
                                       const EmberIpv6Address *remoteAddress,
                                       uint16_t localPort,
                                       uint16_t remotePort);
```

The callback communicates the `sessionId` of the valid secure connection. If the session failed to be established in some way, then the `emberOpenDtlsConnectionReturn` callback will return with an error.

3.5 Obtaining the Identifier of the Secure Connection

Use the following API call to obtain the `sessionId` (identifier) of the secure connection to be used for transport support:

```
void emberGetSecureDtlsSessionId(const EmberIpv6Address *remoteAddress,
                                 uint16_t localPort,
                                 uint16_t remotePort);
```

The `sessionId` is returned in:

```
void emberGetSecureDtlsSessionIdReturn(uint8_t sessionId,
                                       const EmberIpv6Address *remoteAddress,
                                       uint16_t localPort,
                                       uint16_t remotePort);
```

3.6 Sending Data with the Secure Connection

The CoAP header file (coap.h) describes the CoAP transport and how the transmit and receive handlers work. Consult the comments under (`*EmberCoapTransmitHandler`) and `emberProcessCoap` in coap.h for the details. You can find the coap.h file in this location:

<stack install location>/include/coap.h

1. Use the following API call to send data over the secure connection.

```
EmberStatus emberCoapSend(const EmberIpv6Address *destination,
                          EmberCoapCode code,
                          const uint8_t *path,
                          const uint8_t *payload,
                          uint16_t payloadLength,
                          EmberCoapResponseHandler responseHandler,
                          const EmberCoapSendInfo *info);
```

2. Use the `sessionId` obtained with `emberDtlsSecureSessionEstablished` or `emberGetSecureDtlsSessionId` and pass it in to `emberCoapSend`. This ensures that the data will be sent securely.

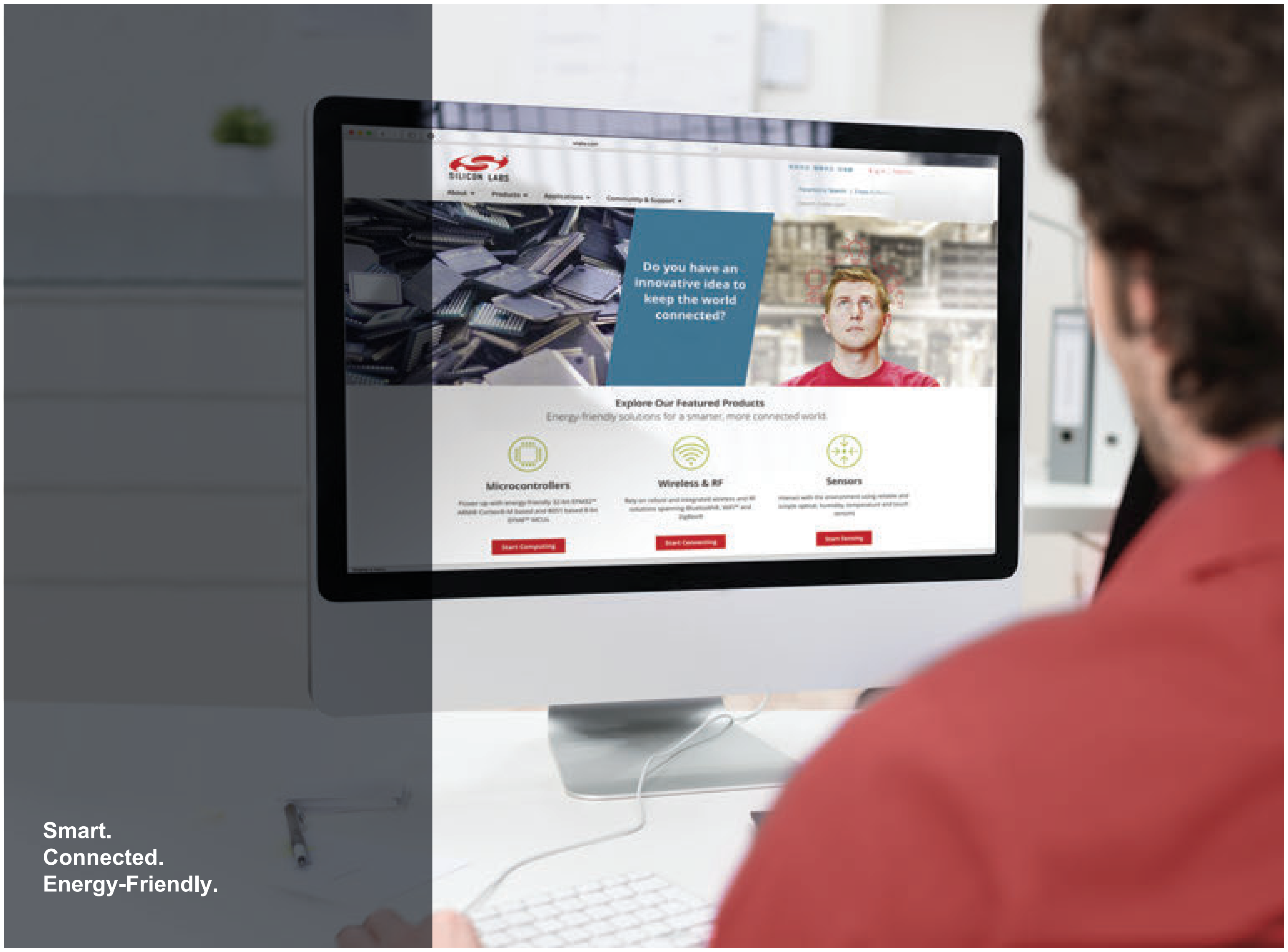
```
info.transmitHandlerData = (void *) (unsigned long) sessionId;
```

3. Set the `transmitHandler` of the `EmberCoapSendInfo` argument like this:

```
info.transmitHandler = &emberDtlsTransmitHandler;
```

The secure data is received on the other end via the following callback. The `sessionId` in the `EmberCoapRequestInfo` indicates which secure session this data is part of.

```
void emberProcessCoap(const uint8_t *message,
                      uint16_t messageLength,
                      EmberCoapRequestInfo *info);
```


Smart.
Connected.
Energy-Friendly.
Do you have an innovative idea to keep the world connected?
Explore Our Featured Products
Energy-friendly solutions for a smarter, more connected world.
Microcontrollers
Wireless & RF
Sensors

Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com