NXP Semiconductors
Application Note

Document Number: AN12257

Rev. 0, 09/2018

IoT Device Secure Connection with LoRa

1. Introduction

The importance of secure connection is realized by many people but the focus is on the security of gateway or IP devices to Internet. Security between gateway and end devices is ignored. You can find the device security from some wireless protocols, like Bluetooth/ZigBee, but it is very complicated to copy to other platforms.

This application note explains how to establish IoT secure connection with LoRa between gateway and end devices.

The hardware is based on i.MX RT1050 EVK/LPC845 MAX boards and LoRa module to consist "star" network. The software of security library can be ported to other NXP MCUs easily.

The base firmware and device drivers are implemented in C while the application layer is implemented in C++ language.

Contents

2. Abbreviations

This chapter provides an overview of the abbreviations as used in this documents.

Abbreviation	Description
ACK	Acknowledge
AES	Advanced Encryption Standard
CMAC	Cipher-based Message Authentication
CCM	Counter with Cipher Block Chaining-Message Authentication Code
DES	Data Encryption Algorithm
ECC	Elliptic Curve Cryptography
ECDH	Elliptic-curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
HMAC	Keyed-Hash Message Authentication Code
MAC	Message Authentication Code
SHA1	Secure Hash Algorithm 1
SHA256	Secure Hash Algorithm 256
TLS	Transport Layer Security
ECB	Electronic Cookbook Mode
CBC	Cipher Block Chaining mode
CFB	Cipher Feedback Mode
CTR	Counter Mode
CCM	Counter Mode with CBC-MAC
HMAC	Keyed-Hash Message Authentication Code
MIC	Message Integrity Code
Crt	Certificate. In the document, Crt is just consisted by public key and ECDSA signature value r/s.

CID	Command Identifier
TRNG	True Random Number Generator
NVM	nonvolatile memory

3. Overview

Security always comes at the prize of higher complexity. Hence, the security mechanisms should only be used when they are really needed. When and how to use the mechanisms is determined by the security policies of a device. This AN introduces two methods for enforcing link-level security and building more advanced security policies. The security methods are Symmetric and Asymmetric connection.

Figure 1. shows the system block diagram. LPC845/i.MXRT1050 is as LoRa controller via SPI interface.

This is a star network, i.MXRT1050 is a Gateway(server) and LPC845 is a node(client). The Gateway can be connected by 250 nodes. Nodes cannot communicate each other directly. Actually, the PHY layer can be changed to others, e.g. GFSK. To communicate securely, client and server need to establish secure channel.

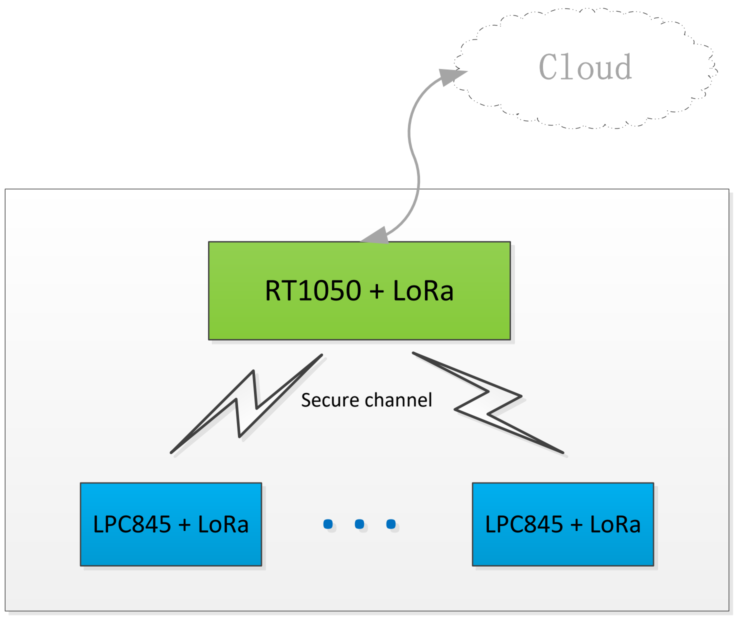


Figure 1. System Block Diagram

4. Software architecture

The software architecture is 3-layer architecture. The layer 1 is based on MCUXpresso SDK for server and code bundle for client. Secure library and LoRa driver are added to layer 1. The layer 2 is C++ Framework. Due to limited resources, this layer is removed in client project. The layer 3 is C++ application code to implement the user purpose.

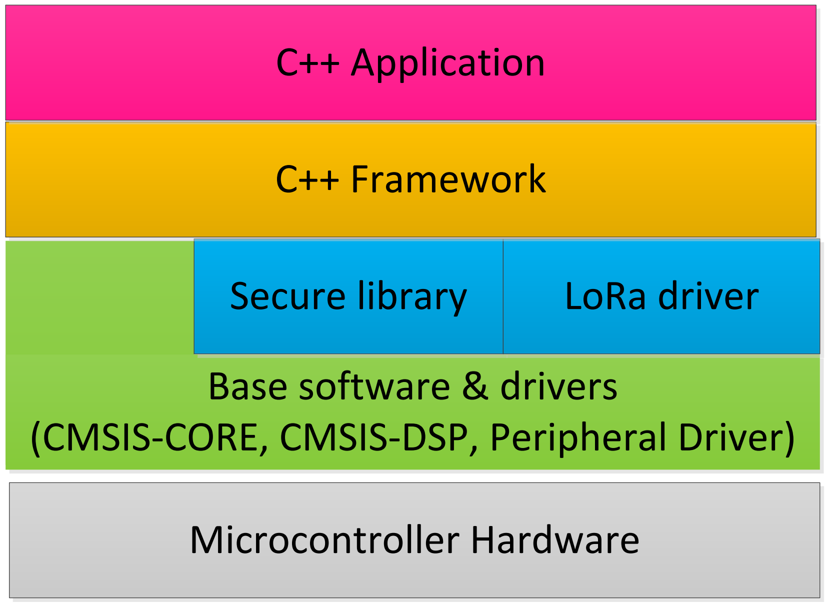


Figure 2. Software Architecture

4.1. Secure library

The secure library is based on mbed TLS under the Apache 2.0 license. Mbed TLS provides SSL/TLS functionality on the server and client side to application and provides the cryptographic building blocks for building other cryptographic protocols. But we modified and cropped mbed TLS source code for secure library including necessary IoT security. Detailed IoT secure library is:

- Symmetric encryption algorithms

  AES, DES
- Modes of operation

  ECB, CBC, CFB, CTR, CCM
- Hash algorithms

  SHA1-1, SHA-224, SHA-256
- MAC modes

  HMAC-SHA1, HMAC-SHA224, HMAC-SHA256, CMAC
- Elliptic Curve Cryptography(ECC)

  Secure library has its own big number library based on mbed TLS for ECC implementation and supports both Elliptic Curve Diffie Hellman(ECDH) and Elliptic Curve Digital Signature Algorithm(ECDSA). The following standardized curves/ECP groups are supported:

  - secp192r1 - 192-bits NIST curve
  - secp224r1 - 224-bits NIST curve
  - secp256r1 - 256-bits NIST curve

- secp384r1 - 384-bits NIST curve
- secp521r1 - 521-bits NIST curve
- secp192k1 - 192-bits Koblitz curve
- secp224k1 - 224-bits Koblitz curve
- secp256k1 - 256-bits Koblitz curve
- bp256r1 - 256-bits Brainpool curve
- bp384r1 - 384-bits Brainpool curve
- bp512r1 - 512-bits Brainpool curve
- m255 - 255-bits Curve25519

➢ Random number generation

We provide NIST standardized HMAC_DRBG pseudo-random number generator.

You can find the IoT secure library(**RT1050_Security_lib.lib and LPC84x_Security_lib.lib by MDK tool**) in the projects.

4.2. C++ framework

A framework for C++ project is established as in Figure 3. . Four main classes are :

CSystem : One task scheduler is included. The types of task include high priority task, low priority task and slice task. If any class wants to create one task, need to inherit COneTask class and register.

CHardware: Includes the classes related with hardware, e.g. Radio.

CCommunication: Includes classes related with communication, e.g. secureMAC.

CBusiness: The CBusiness class includes related or virtual objects. E.g. one client object will be created when received the client joining command and deleted when sent/received disconnect command. KeyManagement class will be included in CBusiness class.

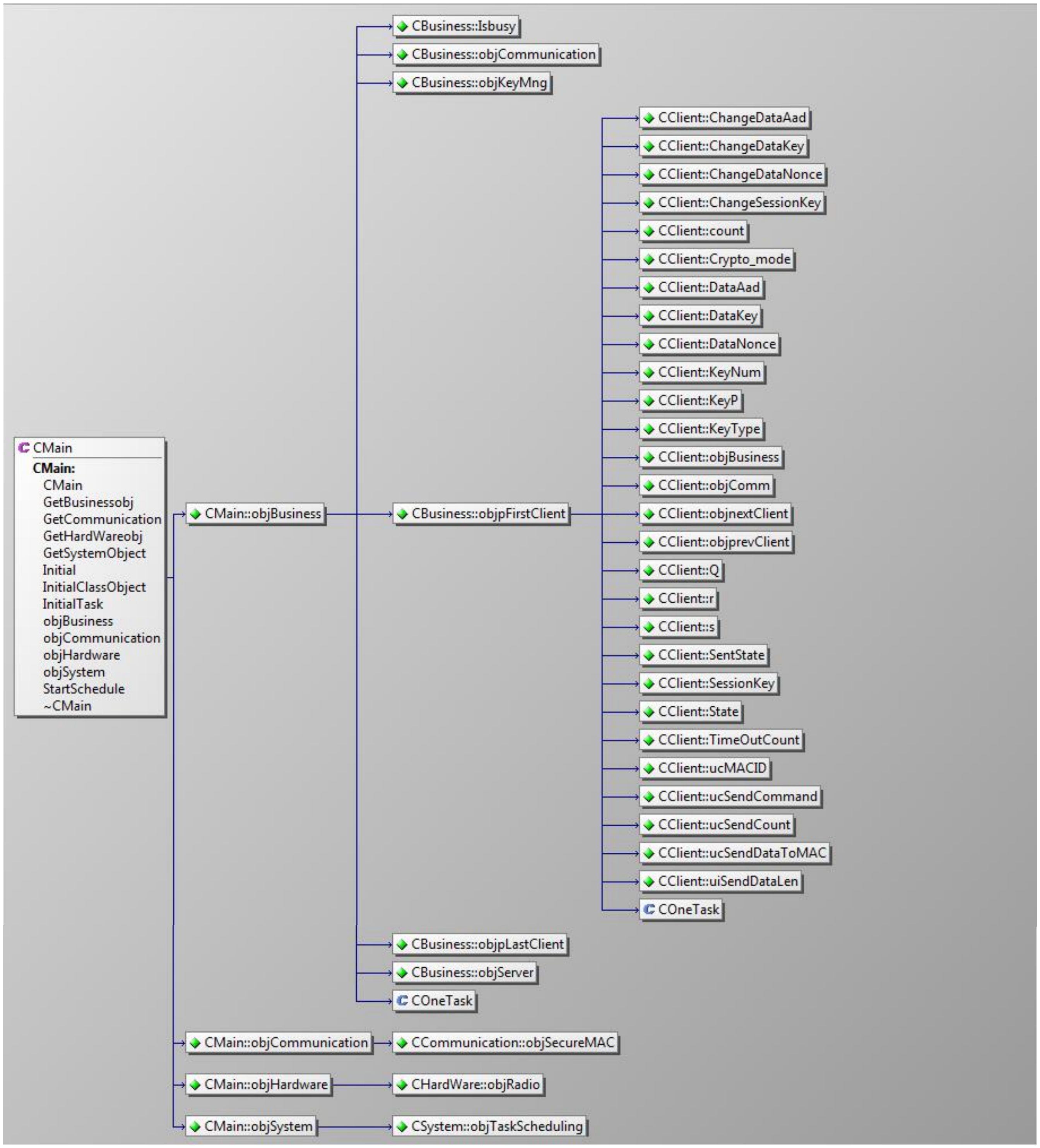


Figure 3. C++ Framework

5. Secure Link Layer Message Formats

A secure Link layer is added for secure connection. All uplink and downlink messages start with **Link header** and two Link IDs(**From Link ID and To Link ID**), followed by Link payload(**PayLoad**), and end with message integrity code(**MIC**).

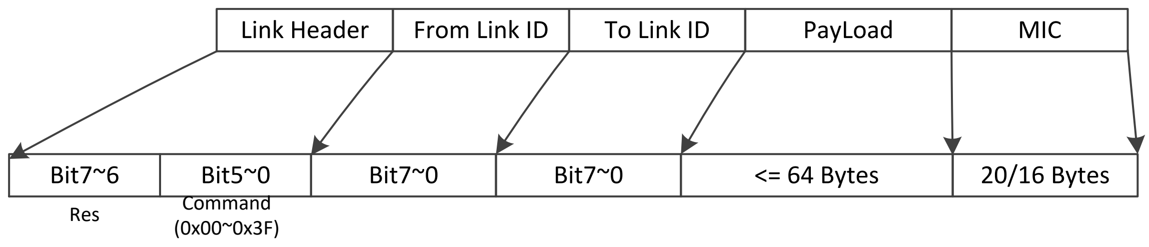


Figure 4. Secure Link Layer Message Format

5.1. Link Header

The Link Header reserves 2 bits(**Bit7~6**) and specifies the message commands(**Bit5~0**). The list of detailed commands is in Table 1.

Table 1. Link Commands

CID	COMMAND	TRANSMITTED BY		DESCRIPTION
		Client	Server	
0x01	***Join Request***	√		Used by a client to initiate joining network request.
0x02	***Join Accept***		√	Answer to "Join Request" command. Server allows this client to join the network.
0x03	***Join Type***	√		"Join Type" contains two methods: Symmetric and Asymmetric. To judge the method from payload message.
0x04	***Type Accept***		√	Answer to "Type Accept" command.
0x10	***Key Type***	√		If "Join Type" is Symmetric, CIDs(0x10 ~0x14) are available. Client need send own Key to Server.
0x11	***Key number***		√	Answer to "Key Type" command and server confirms the Key Type is right. then server sends "Key number" command and tell client which Key will be used.
0x12	***Key confirm***	√		Answer to "Key number" command. Tell server

				Key is OK.
0x13	***Symmetric session key***		√	Server generates session key, then sends it to client.
0x14	***Symmetric session key confirmed***	√		Answer to “Symmetric session key” command.
0x20	***Request server Crt***	√		If “Join Type” is Asymmetric, CIDs(0x20~0x28) are available. Client request server’s Crt. Need several times to get Crt.
0x21	***Response server Crt***		√	Answer to “Request server Crt” command. Server sends own Crt to client. Need several times to send Crt.
0x22	***Server Crt confirmed***	√		After received server CA, client will verify CA. if the CA is correct, client will send “Server CA confirm” command.
0x23	***Request client Crt***		√	Server request client’s Crt. Need several times to get Crt.
0x24	***Response client Crt***	√		Answer to “Request client Crt” command. Client sends own Crt to server. Need several times to send Crt.
0x25	***Client Crt confirmed***		√	After received client Crt, client will verify Crt. if the Crt is correct, server will send “Client Crt confirmed” command.
0x26	***Shared Key confirmed***	√		After both Crts are confirmed, Client and server will generate same shared Key. Client sends this command.
0x27	***Asymmetric session key***		√	Server generates session Key, then sends it to client.
0x28	***Asymmetric session key confirmed***	√		Client confirmed that session key is received successfully.
0x30	***Data Key***		√	Server generates data Key, then sends it to client.
0x31	***Data Key confirmed***	√		Client confirmed that data key is received successfully.

0x32	***Security connected***		√	Server notifies secure channel is established.
0x35	***Change Session Key***	√	√	Client or server can send the command to request to change session key.
0x36	***Change Session Key confirmed***	√	√	Answer to “Change Session Key” command. Receiver generates new session key, sends it to the side requested.
0x37	***Change Data Key***	√	√	Client or server can send the command to request to change data key.
0x38	***Change Data Key confirmed***	√	√	Answer to “Change Data Key” command. Receiver generates new data key, sends it to the side requested.
0x3A	***Data***	√	√	After the secure channel is established, send useful data to another side.
0x3B	***ACK***	√	√	Answer to “Data” command. Receiver send the command to tell sender the message is received.
0x3F	***Disconnect***	√	√	Send the command to disconnect secure channel.

5.2. Link IDs

There are two Link IDs in the secure Link layer. One Link ID is “**From Link ID**” means where the message is from, another Link ID is “**To Link ID**” means where the message is to. When received one package, firstly need to judge whether the package is for “me” according to “To Link ID” and whether the package is needed to process according to “From Link ID”.

5.3. PayLoad

If a data frame carries a payload, the PayLoad maybe is encrypted before the message integrity code is calculated and you can get some information from PayLoad. about how to encrypt/decrypt the Payload, please see Application Layer Data Encryption.

5.4. MIC

The message integrity code(MIC) is calculated over all the fields in the message.

msg = **Link Header | From Link ID | To Link ID | PayLoad**

There are two methods to do message integrity code(MIC) including **HMAC-SHA1** and **CMAC.**

HMAC-SHA1(K, msg) = $\boldsymbol{H((K \oplus opad)\,||\,H((K \oplus ipad)\,||\,msg))}$

Where

K is secret key.

H is approved hash function.

$\oplus$ is exclusive-or operation

ipad is inner pad.

opad is outer pad.

MIC = HMAC-SHA1[0…19], where MIC should be 20 bytes.

CMAC = aes128-cmac(K, msg)

MIC = CMAC[0…15], where MIC should be 16 bytes.

6. Key management

Key management involves the generation, distribution, storage, and handling of the cryptographic keys. Many keys are involved in a secure connection. The keys stored are very important, since they are used to encrypt/decrypt messages during establishing secure channel. Table 2. lists all these keys.

Table 2. Keys Introduction

KEY NAME	METHOD GENERATED	LINK COMMAND USED(CID)	FILED (LINK LAYER MIC / DATA ENCRYPTION)	DESCRIPTION
HKey①	Stored④	0x01 ~ 0x26	Link Layer MIC	Client and server should have same Hkey. It is stored in nonvolatile memory.
ECC Key②	Stored	-	-	If the connection mode is “Asymmetric”, Server and Client should store own ECC Key
SymmKey③	Stored	-	-	If the connection mode is “Symmetric”, client stores 5 Keys and server also stores same client’s Keys.
Keyx	Stored	0x13 ~ 0x14 0x30 ~ 0x32	DATA ENCRYPTION	This key is one of SymmKey. If the connection mode is “Symmetric”, server will select one of SymmKey randomly as temporary data key.
Keyp	Server/Cli	0x27 ~ 0x28	Link Layer MIC	After Crts are verified, server and client

	ent	0x30 ~ 0x32		will get each other's public key. They can generate same shared key called "Keyp" by ECDH.
Session key	Server	0x30 ~ 0x3F	Link Layer MIC	Server generates session key during establishing secure channel by TRNG. If responding "Change Session Key" command, server or client generates it.
Data key	Server/Client	0x35 ~ 0x3F	DATA ENCRYPTION	Server generates Data key during establishing secure channel by TRNG. If responding "Change Data Key" command, server or client generates it.

NOTE

1. For example, in ConfigutationInfo.c file of security_sw_RT and security_sw_LPC84x projects, the arrays of ConstHKey are for this.
2. For example, in ConfigurationInfo.c file of security_sw_RT and security_sw_LPC84x projects, the arrays of Serverd/Clientd are ECC private key and the arrays of Q_XYZ are ECC public key.
3. For example, in ConfigurationInfo.c file of security_sw_RT project, array of ConstsymmKey consists of 5 clients' SymmKey, and every client SymmKey consists of 5 Keys.
4. "Stored" means the key is stored in nonvolatile memory before system works.All keys stored need inject NVM beforehand and it is better saved in OTP memory. All keys non-stored need destroyed after secure channel disconnected. Session key and Data key need to be changed after a certain time.
5. Key management is related with entire system security, so we just list the keys used. If you want to learn more about this, you can check from NXP Secure MCU features, e.g PUF/OTP/DICE…

7. Application Layer Data Encryption

Eencryption of PayLoad of secure Link Layer, called Application Layer Data Encryption. There are two different methods of encryption/decryption: AES128-CBC, AES128-CCM in different CID package. The PayLoads with CIDs(0x01~0x12, 0x20~0x26) are not encrypted.

7.1. AES128-CBC encryption/decryption

The message is encrypted by AES-128 in CBC mode. This encryption is just used to exchange session key and data key during establishing secure channel, you can find it in the packages with CIDs(0x13~0x14, 0x27~0x28, 0x30~0x32).

7.2. AES128-CCM encryption/decryption

The message is encrypted and authenticated by AES-128 in CCM mode. CCM is used to provide assurance of the confidentiality and the authenticity of data by combining the techniques of the Counter(CTR) mode and the Cipher Block Chaining-Message Authentication Code(CBC-MAC) algorithm. This encryption/decryption is used after the secure channel is established, you can find it in the packages with CIDs(0x35~0x3F). After the plain text is encrypted, the encrypted data with same length and 8 bytes tag will be generated.

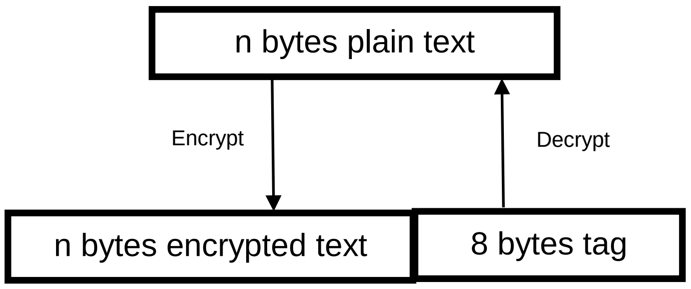


Figure 5. Message Encryption and Authentication

8. The procedure of establishing secure channel

A secure channel should be established before starting communication between client and server. There are two methods to establish a secure channel: Symmetric connection and Asymmetric connection. Same initiating connection and secure data exchange procedure is used in both methods.

8.1. Initiate connection

The secure connection is initiated by client. Server verifies the package integrity by Hkey and if permitted the client to add network by “From Link ID”. After client received the “Join Accept” command, will send “Join Type” to tell server the establishing secure channel method (to judge via first byte of payload: 0x00(Symmetric connection), 0x01(Asymmetric connection)).

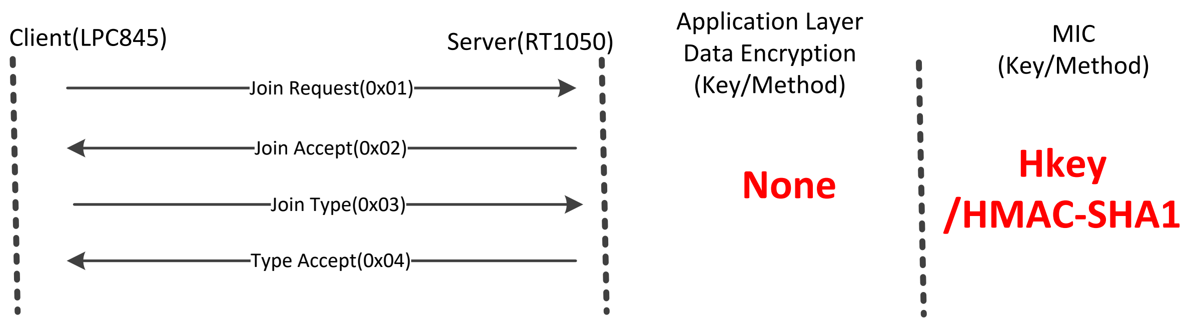


Figure 6. Initiate connection

8.2. Symmetric connection

After client received the “Type Accept” command, client sends “Key Type” command to tell server which 5 SymmKeys can be used. Server uses TRNG to generate random number to select one SymmKey(Keyx) and tell client the key number. Then server generates session key by TRNG and sends it encrypted by Keyx in CBC mode.

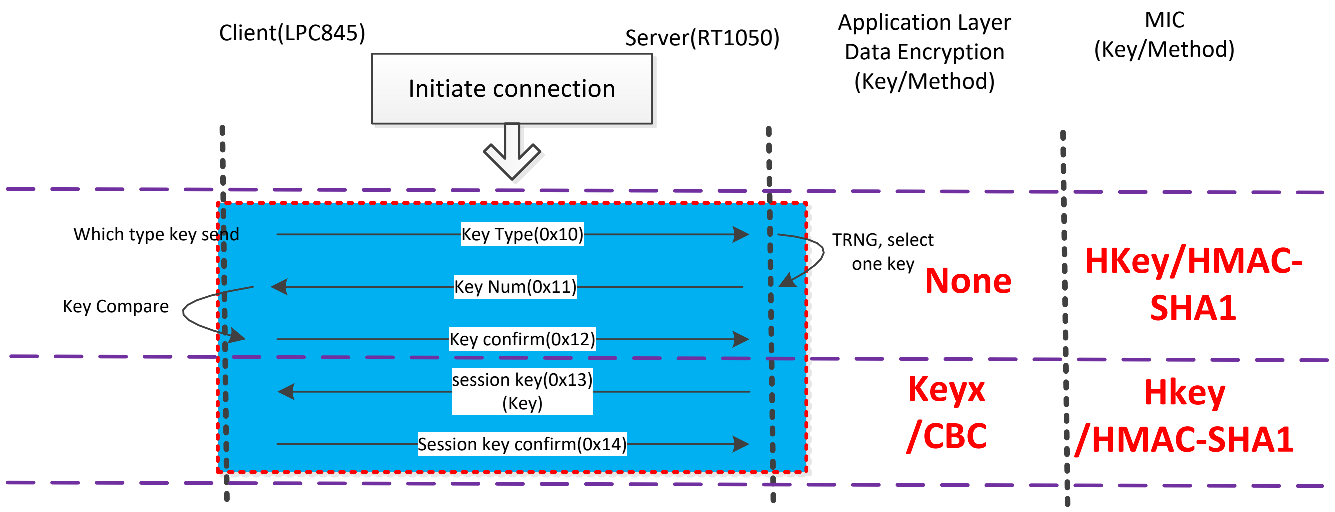


Figure 7. Symmetric connection

8.3. Asymmetric connection

After client received the “Type Accept” command, client and server get each other’s Crt and verify it by ECDSA. Then they can get same shared key(keyp) by ECDH and use it to encrypt/decrypt Application Layer Data. The session key is generated by server’s TRNG.

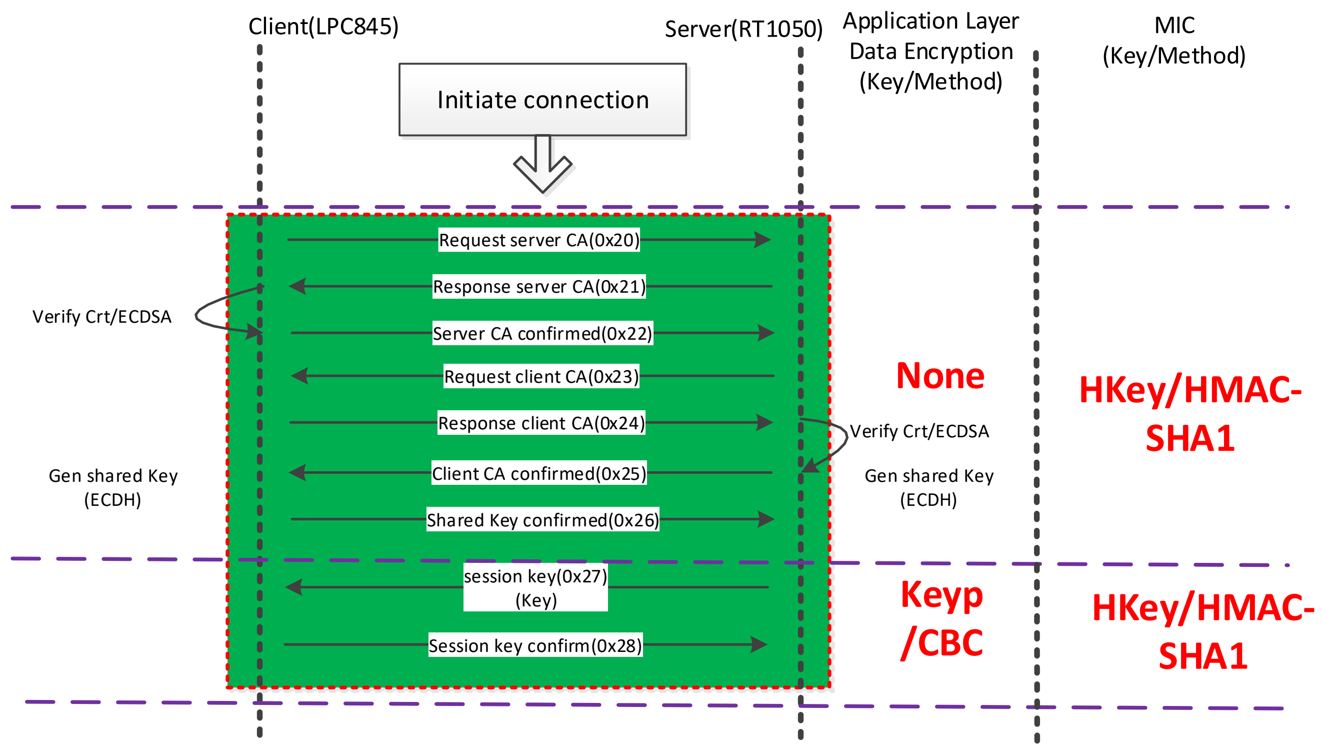


Figure 8. Asymmetric connection

8.4. secure data exchange

MIC is calculated by session key in CMAC mode, after the session key is generated. Server generates Data Key(including Key, Nonce, AAD) by TRNG and sends it encrypted by Keyp or Keyx in CBC mode. When client received the command with CID 0x32, the security channel is established. Server and client can communicate with command with CID 0x3A and 0x3B under security. the command with CID 0x3F means that the sender requests disconnect secure connection.

NOTE

In a while, the session key and data key should be updated.

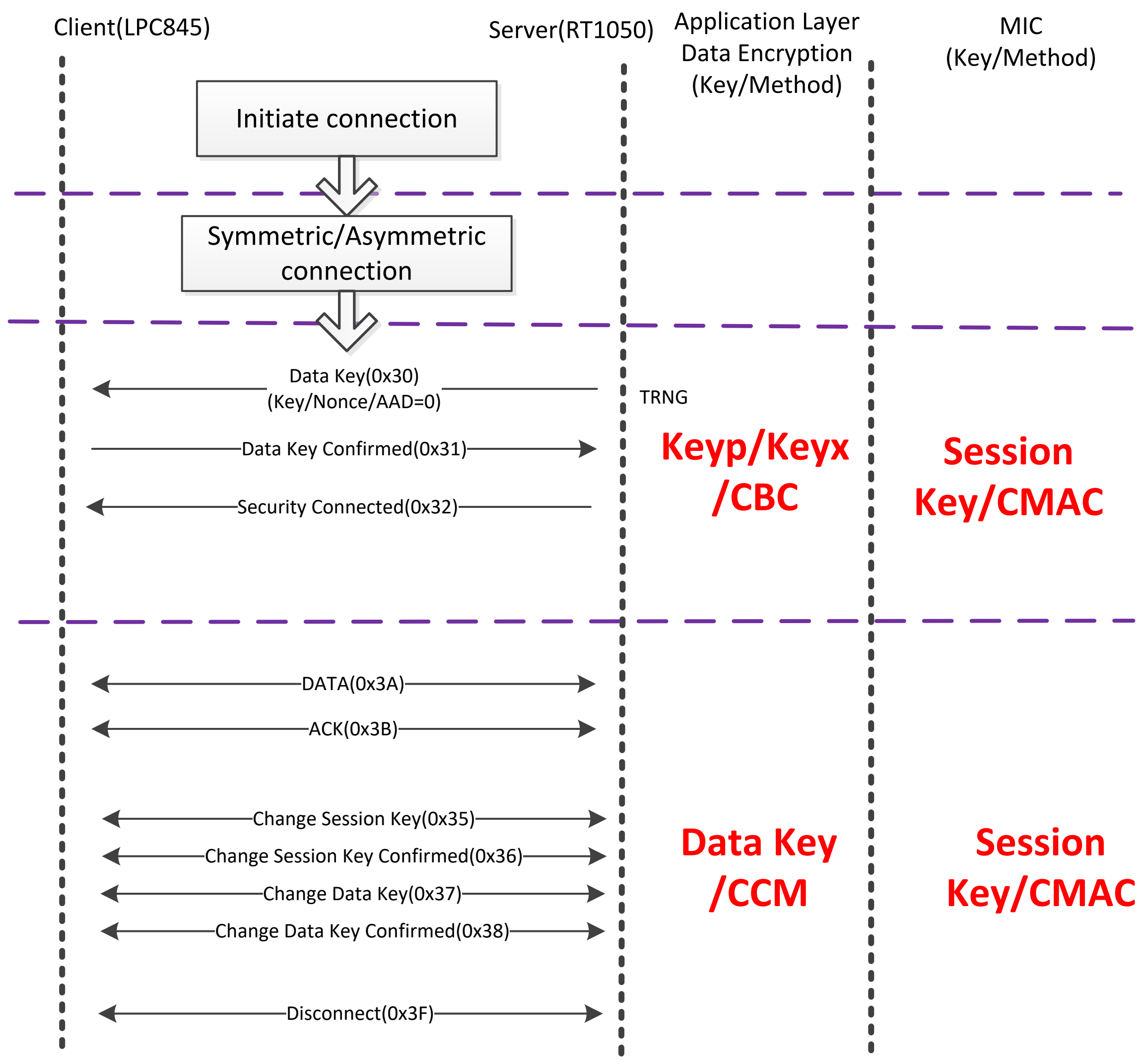


Figure 9. Secure data exchange

9. Hardware platform

This section presents introduction of the hardware platform for the demo application.

9.1. LoRa board

The LoRa board consists of LoRa module and LoRa baseboard with Arduino interface.

➢ Key features of LoRa module

- Radio frequency: 400 ~ 500 MHz
- Up to +20 dBm constant RF output
- High sensitivity: down to -135 dBm.
- Preamble detection

- Packet engine up to 256 bytes with CRC

➢ Key features of LoRa baseboard

- Arduino interface
- User LEDs to show the status of LoRa module
- Temperature sensor: PCT2075
- SMA interface

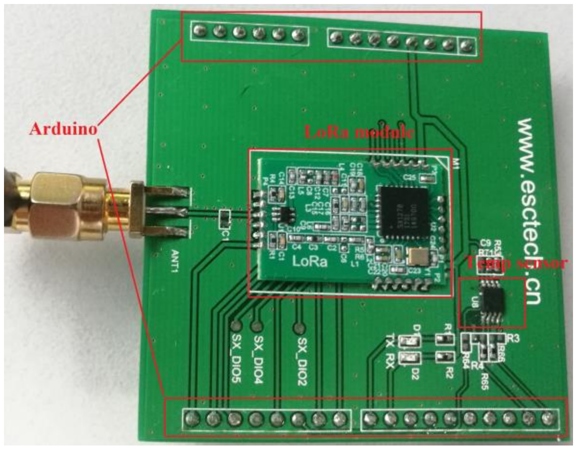


Figure 10. LoRa board

9.2. Server hardware

i.MX RT1050 EVK board is as LoRa board's baseboard.

9.2.1. i.MXRT1050 EVK board introduction

➢ The key features of i.MXRT1050 EVK

- Memory: 256 Mbit SDRAM, 64 Mbit Quad SPI Flash, 512 Mbit Hyper Flash, TF Card Slot
- Communication interfaces: USB 2.0 OTG connector, USB 2.0 host connector, 10/100 Mbit/s Ethernet connector, CAN bus connector
- Multimedia interfaces: CMOS sensor connector, LCD connector
- Audio interfaces: 3.5 mm stereo headphone hack, board-mounted microphone, SPDIF connector (not mounted by default)
- Hardware and software platforms
- Debug interfaces: On-board debug adapter with DAP-Link, JTAG 20-pin connector
- Arduino interface
- User button and LEDs

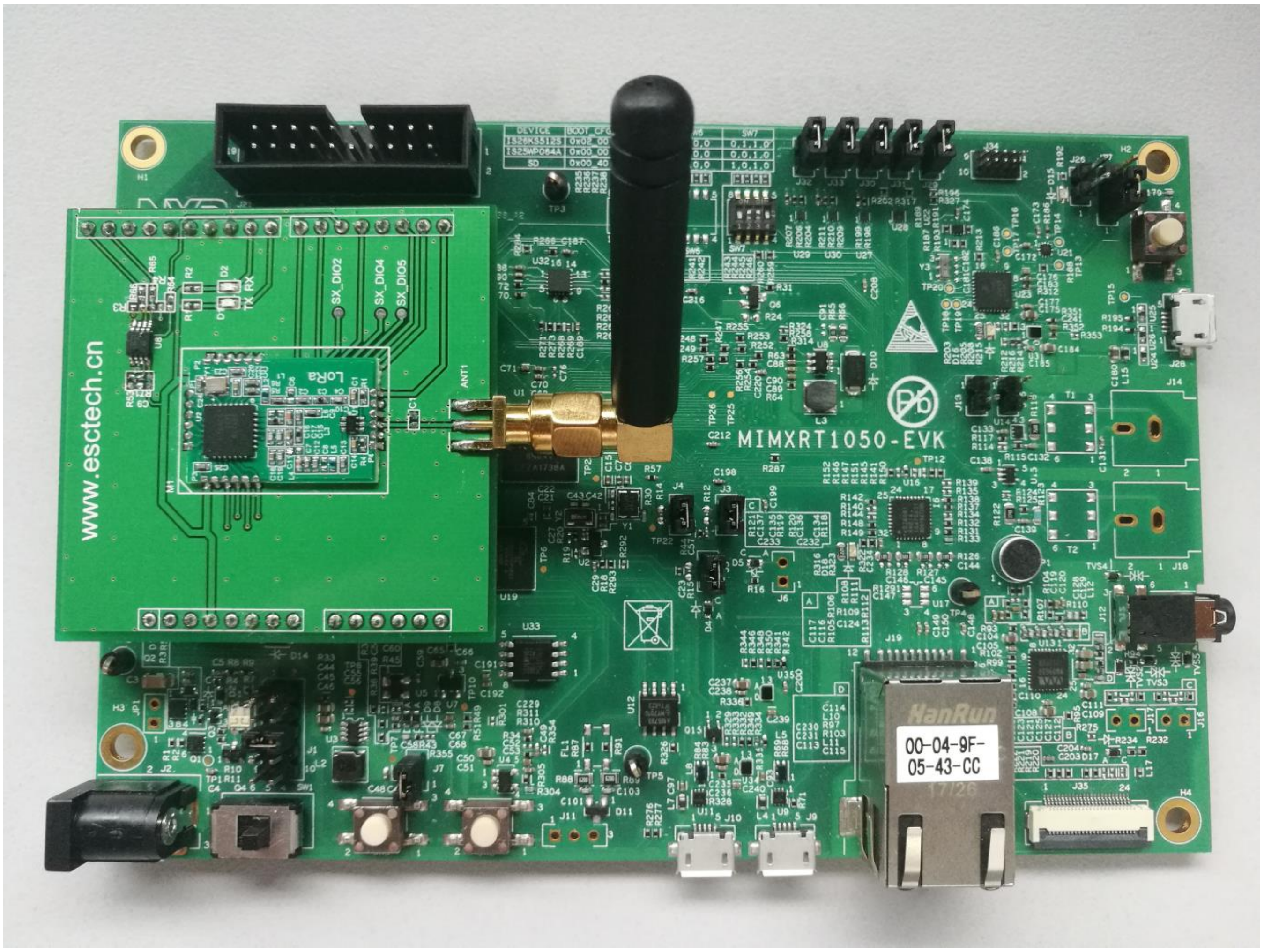


Figure 11. Server hardware

9.2.2. i.MXRT1050 EVK board settings

To enable spi and some GPIOs features, EVK board settings need to be changed.

Remove resistors: R341

Weld resistors: R278, R279, R280, R281, R288, R289, R276, R277

After changing these settings, i.MXRT1050 EVK board and LoRa board can work.

9.3. Client hardware

LPC845 MAX board is as LoRa board’s controller board.The key features of LPC845 MAX board are:

- On-board CMSIS-DAP (debug probe) with VCOM port, based on LPC11U35 MCU
- Debug connector to allow debug of target MCU using an external probe
- Red, green and blue user LEDs
- Target ISP and user/wake buttons
- Target reset button
- LPCXpresso expansion connector
- DAC output via speaker driver and speaker
- Arduino™ connectors compatible with the “Arduino UNO” platform

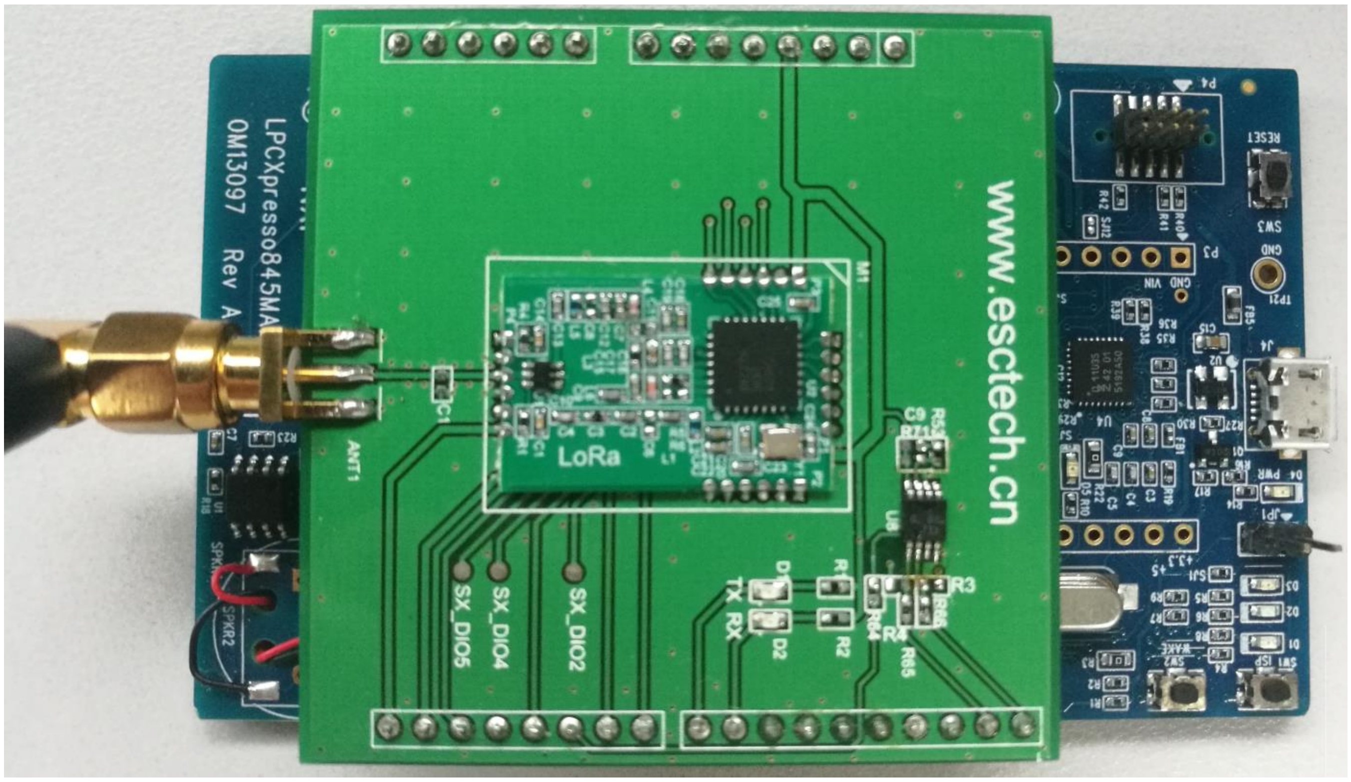


Figure 12. Client hardware

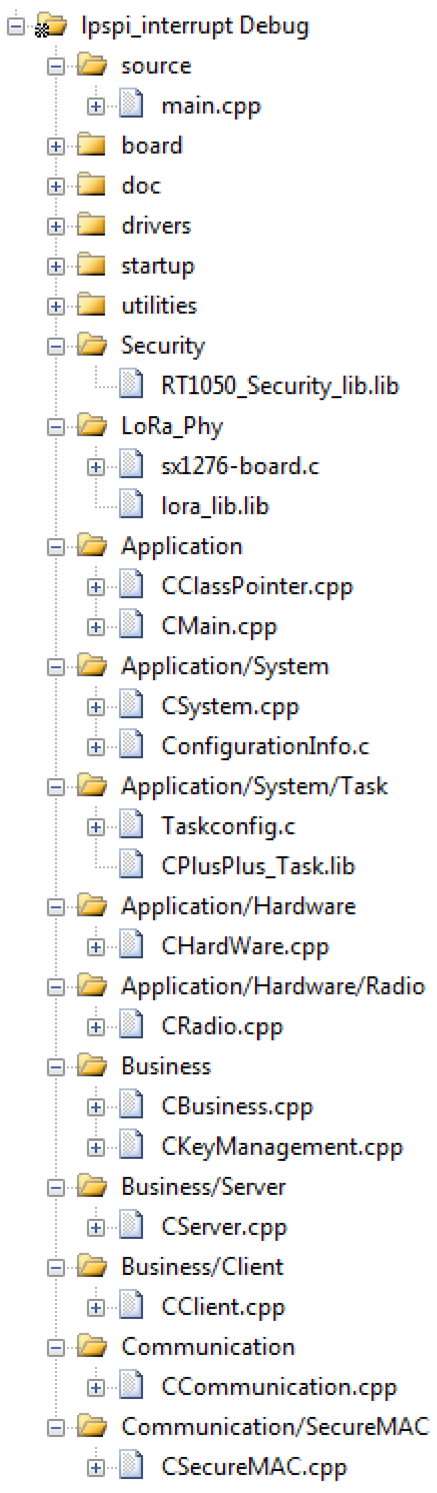
Don’t need to modify LPC845 MAX board to comply with LoRa board.

10. Software platform

There are two software platforms: one is server software platform on i.MXRT1050 SDK, another is client software platform on LPC845 code bundle. The application code is implemented in C++ language. The toolchain is Keil MDK 5.24.

10.1. Server software

The server code implements the functions of adding asymmetric client and symmetric client. Related device information be can found in “ConfigurationInfo.c” file.



```
lpspi_interrupt Debug
    source
        main.cpp
    board
    doc
    drivers
    startup
    utilities
    Security
        RT1050_Security_lib.lib
    LoRa_Phy
        sx1276-board.c
        lora_lib.lib
    Application
        CClassPointer.cpp
        CMain.cpp
    Application/System
        CSystem.cpp
        ConfigurationInfo.c
    Application/System/Task
        Taskconfig.c
        CPlusPlus_Task.lib
    Application/Hardware
        CHardWare.cpp
    Application/Hardware/Radio
        CRadio.cpp
    Business
        CBusiness.cpp
        CKeyManagement.cpp
    Business/Server
        CServer.cpp
    Business/Client
        CClient.cpp
    Communication
        CCommunication.cpp
    Communication/SecureMAC
        CSecureMAC.cpp
```

Figure 13. Server software

10.2. Client software

Due to limited resources of LPC845, C++ framework is removed and do not have task scheduler.

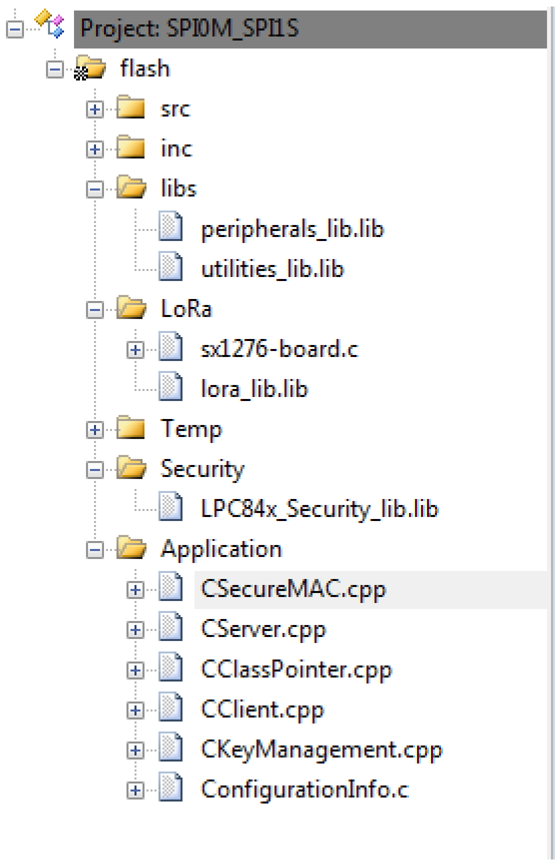

Project: SPI0M_SPI1S
flash
src
inc
libs
peripherals_lib.lib
utilities_lib.lib
LoRa
sx1276-board.c
lora_lib.lib
Temp
Security
LPC84x_Security_lib.lib
Application
CSecureMAC.cpp
CServer.cpp
CClassPointer.cpp
CClient.cpp
CKeyManagement.cpp
ConfigurationInfo.c

Figure 14. Client software

11. Hands-on

In this hands-on, client 1 with asymmetric connection and client 2 with symmetric connection will be added to the star network. See Figure 15.

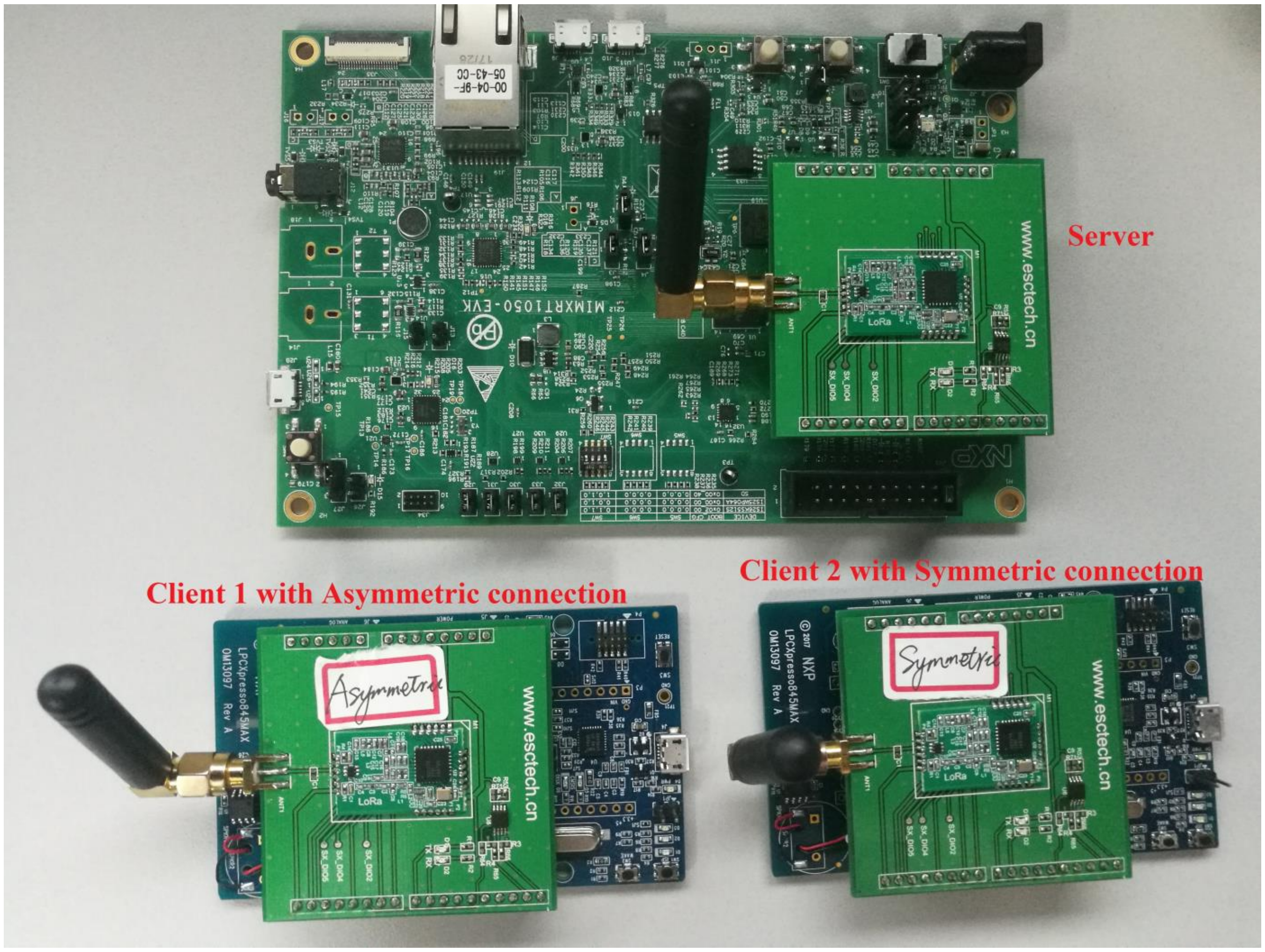

Server
Client 1 with Asymmetric connection
Client 2 with Symmetric connection
Asymmetric
Symmetric
MIMXRT1050-EVK
LPCXpresso845MAX
www.esctech.cn

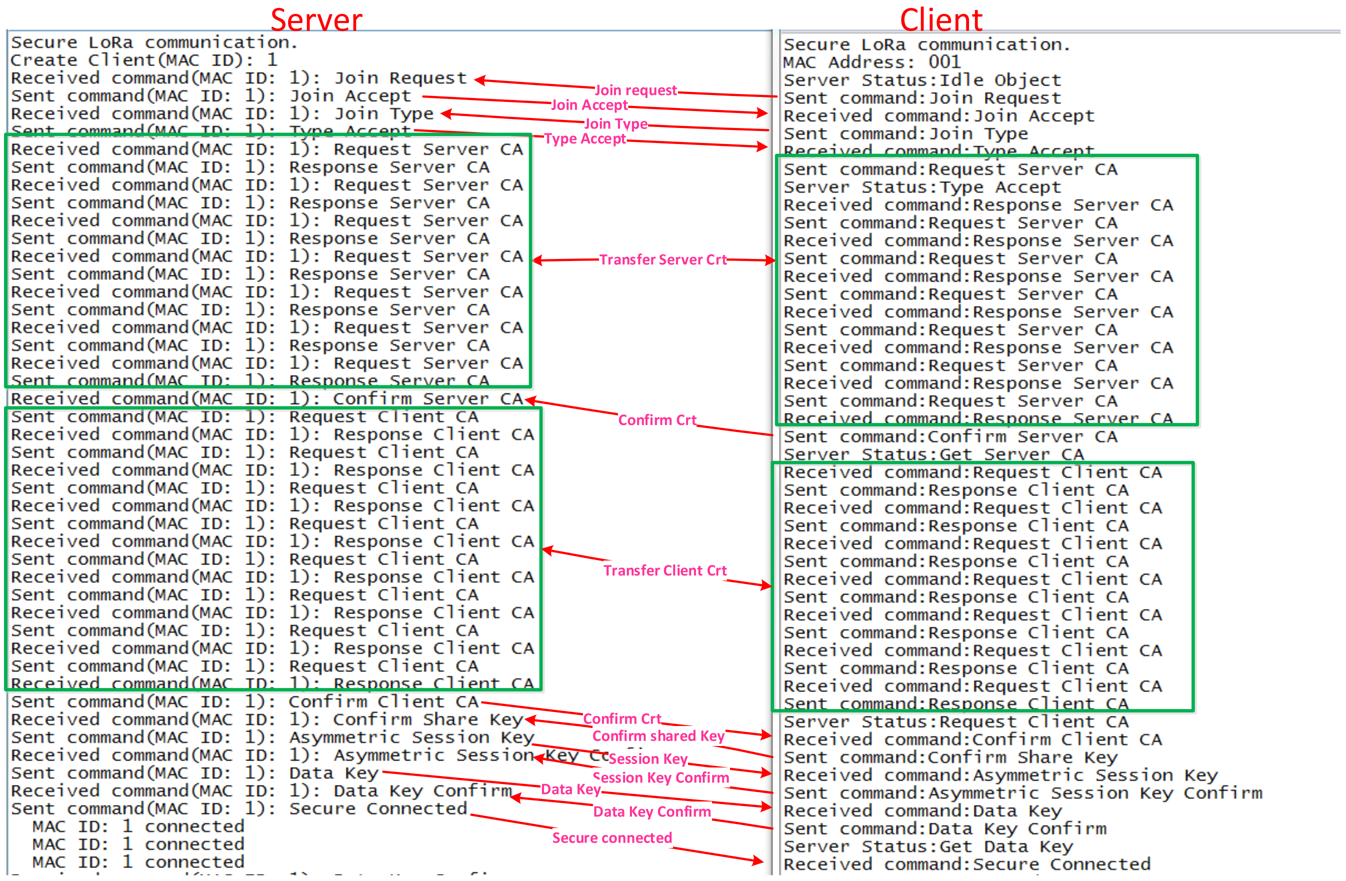
Figure 15. Demo boards

To let the 3 boards to work, follow the below steps:

1) Connect a micro USB cable between PC host and the OpenSDA USB port on the board for the three boards.
2) Open a serial terminal on PC for OpenSDA serial device with these setting:
   - 115200 baud rate
   - 8 data bits
   - One stop bit
   - No flow control
3) Build the related projects(note: if symmetric connection, remember to define “DefSYMMETRIC”), download the program to the target board.

4) Press the reset button on the 3 boards.

When the code runs successfully, you can see the similar information from the terminal as in Figure 16. below is asymmetric connection.

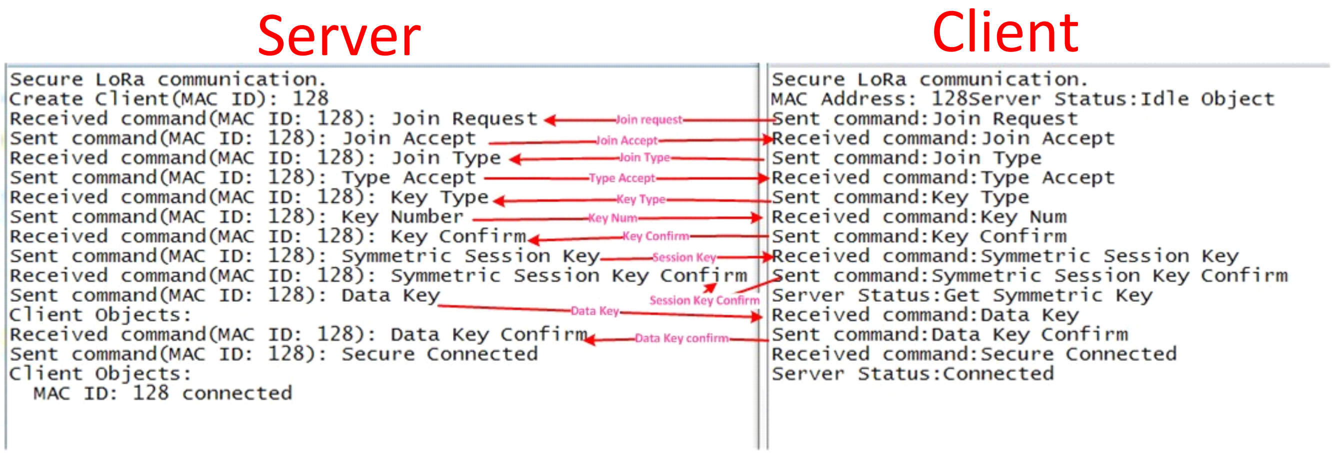


```
Secure LoRa communication.
Create Client(MAC ID): 1
Received command(MAC ID: 1): Join Request
Sent command(MAC ID: 1): Join Accept
Received command(MAC ID: 1): Join Type
Sent command(MAC ID: 1): Type Accept
Received command(MAC ID: 1): Request Server CA
Sent command(MAC ID: 1): Response Server CA
Received command(MAC ID: 1): Request Server CA
Sent command(MAC ID: 1): Response Server CA
Received command(MAC ID: 1): Request Server CA
Sent command(MAC ID: 1): Response Server CA
Received command(MAC ID: 1): Request Server CA
Sent command(MAC ID: 1): Response Server CA
Received command(MAC ID: 1): Request Server CA
Sent command(MAC ID: 1): Response Server CA
Received command(MAC ID: 1): Request Server CA
Sent command(MAC ID: 1): Response Server CA
Received command(MAC ID: 1): Request Server CA
Sent command(MAC ID: 1): Response Server CA
Received command(MAC ID: 1): Confirm Server CA
Sent command(MAC ID: 1): Request Client CA
Received command(MAC ID: 1): Response Client CA
Sent command(MAC ID: 1): Request Client CA
Received command(MAC ID: 1): Response Client CA
Sent command(MAC ID: 1): Request Client CA
Received command(MAC ID: 1): Response Client CA
Sent command(MAC ID: 1): Request Client CA
Received command(MAC ID: 1): Response Client CA
Sent command(MAC ID: 1): Request Client CA
Received command(MAC ID: 1): Response Client CA
Sent command(MAC ID: 1): Request Client CA
Received command(MAC ID: 1): Response Client CA
Sent command(MAC ID: 1): Request Client CA
Received command(MAC ID: 1): Response Client CA
Sent command(MAC ID: 1): Request Client CA
Received command(MAC ID: 1): Response Client CA
Sent command(MAC ID: 1): Confirm Client CA
Received command(MAC ID: 1): Confirm Share Key
Sent command(MAC ID: 1): Asymmetric Session Key
Received command(MAC ID: 1): Asymmetric Session Key C
Sent command(MAC ID: 1): Data Key
Received command(MAC ID: 1): Data Key Confirm
Sent command(MAC ID: 1): Secure Connected
  MAC ID: 1 connected
  MAC ID: 1 connected
  MAC ID: 1 connected
```

```
Secure LoRa communication.
MAC Address: 001
Server Status:Idle Object
Sent command:Join Request
Received command:Join Accept
Sent command:Join Type
Received command:Type Accept
Sent command:Request Server CA
Server Status:Type Accept
Received command:Response Server CA
Sent command:Request Server CA
Received command:Response Server CA
Sent command:Request Server CA
Received command:Response Server CA
Sent command:Request Server CA
Received command:Response Server CA
Sent command:Request Server CA
Received command:Response Server CA
Sent command:Request Server CA
Received command:Response Server CA
Sent command:Request Server CA
Received command:Response Server CA
Sent command:Confirm Server CA
Server Status:Get Server CA
Received command:Request Client CA
Sent command:Response Client CA
Received command:Request Client CA
Sent command:Response Client CA
Received command:Request Client CA
Sent command:Response Client CA
Received command:Request Client CA
Sent command:Response Client CA
Received command:Request Client CA
Sent command:Response Client CA
Received command:Request Client CA
Sent command:Response Client CA
Received command:Request Client CA
Sent command:Response Client CA
Server Status:Request Client CA
Received command:Confirm Client CA
Sent command:Confirm Share Key
Received command:Asymmetric Session Key
Sent command:Asymmetric Session Key Confirm
Received command:Data Key
Sent command:Data Key Confirm
Server Status:Get Data Key
Received command:Secure Connected
```

Figure 16. Asymmetric connection

Figure 17. shows a symmetric connection



```
Secure LoRa communication.
Create Client(MAC ID): 128
Received command(MAC ID: 128): Join Request
Sent command(MAC ID: 128): Join Accept
Received command(MAC ID: 128): Join Type
Sent command(MAC ID: 128): Type Accept
Received command(MAC ID: 128): Key Type
Sent command(MAC ID: 128): Key Number
Received command(MAC ID: 128): Key Confirm
Sent command(MAC ID: 128): Symmetric Session Key
Received command(MAC ID: 128): Symmetric Session Key Confirm
Sent command(MAC ID: 128): Data Key
Client Objects:
Received command(MAC ID: 128): Data Key Confirm
Sent command(MAC ID: 128): Secure Connected
Client Objects:
  MAC ID: 128 connected
```

```
Secure LoRa communication.
MAC Address: 128Server Status:Idle Object
Sent command:Join Request
Received command:Join Accept
Sent command:Join Type
Received command:Type Accept
Sent command:Key Type
Received command:Key Num
Sent command:Key Confirm
Received command:Symmetric Session Key
Sent command:Symmetric Session Key Confirm
Server Status:Get Symmetric Key
Received command:Data Key
Sent command:Data Key Confirm
Received command:Secure Connected
Server Status:Connected
```

Figure 17. Symmetric connection

12. Use cases

This AN involves secure library and how to establish secure channel, so these can be used in different situations. We list some use cases.

12.1. Point to point secure connection

Two devices want to communicate under secure channel by wireless or wired way.

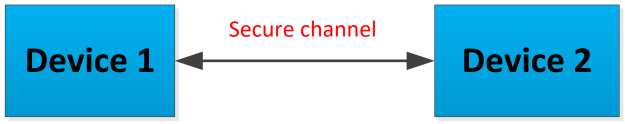


Figure 18. Point to point secure connection

12.2. Secure star/mesh network

In a local area network, sometimes need to create a secure network to prevent the man-in-the-middle attack. E.g. LoRa/GFSK network doesn't have standard secure protocol, it is an appropriate choice to copy this AN.

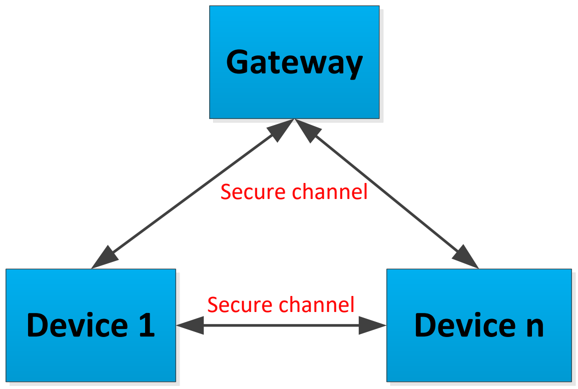


Figure 19. Secure star/mesh network

12.3. Secure connection for RS484/CAN

In industrial field, RS485/CAN is a low-level protocol and does not support any security features. In most implementations, applications are expected to deploy their own security mechanisms, we have given an example for this situation.

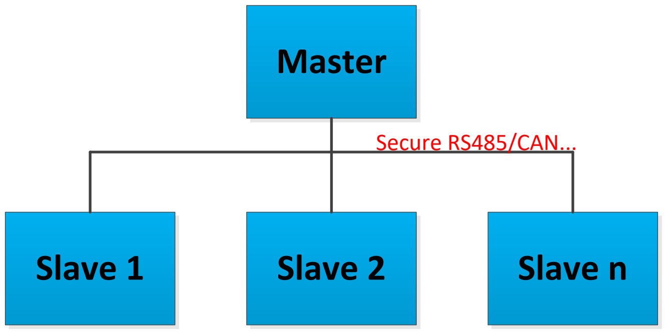


Figure 20. Secure connection for RS485/CAN

13. Conclusion

This application note describes how to set up secure connection between devices. An IoT secure library is provided for NXP MCU customers. You can use similar secure connection for your products.

If you are interested in the hardware and software of this reference design, please send email to marketing team to request.

14. Reference

- Bluetooth Security
- LoRaWAN Specification
- https://www.mbed.com/en/technologies/security/mbed-tls
- ARM Cortex-M7 Processor Technical Reference Manual (Revision: r1p1)
- i.MX RT1050 Processor Reference Manual
- LoRa SX1276/77/78/79 data sheet

15. Revision history

Revision history

Revision number	Date	Substantive changes
0	09/2018	Initial release