Bluetooth Low Energy Protocol Stack

Security Library

Introduction

Security Library provides APIs to ease the usage of security features provided by BLE protocol stack. Security Library shall use with BLE protocol stack V1.20. The library can be used with Central / Peripheral. Also, can be used on Embedded / Modem Configuration.

Target Device

RL78/G1D

Contents

Terminology

Terminology	Remark
Role	Each device has a role defined by profile or service.
Advertising	The act to advertise a presence of itself to scanning devices.
Scanning	The act to search for advertising devices.
Central	One of the role defined by Link Layer and GAP. The device which was scanning, or initiating a connection before establishing the connection, becomes Central after the connection. Central defines the timings of transmission during the connection.
Peripheral	One of the role defined by Link Layer and GAP. The device which was advertising before a connection, becomes Peripheral after the connection.
MITM attack	Man In The Middle attack. An attacker device intervenes between two communicating devices and eavesdrop, insert or modify information on communication.
Authentication	Authentication is executed to establish security between two devices.
Just Works	One of the pairing method. This method is used when devices have no input / output capabilities, or have no requirement on high security. This method has no protection against MITM attack. Unauthenticated pairing.
Passkey Entry	One of the pairing method. This method is used when devices have 6-digits number input / output capabilities, such as a display / keyboard. This method has protection against MITM attack. Authenticated pairing.
OOB	Out of Band. One of the pairing method. Two devices exchange the information needed for authentication by using the other than Bluetooth.
RPA	Resolvable Privacy Address. One of the Bluetooth device address type. This address is changed in a period to reduce the ability to track the device from an attacker device.
IRK	Identity Resolving Key. This key is used to generate RPA. The device uses RPA distributes IRK to the peer device during pairing. The peer device can identify the device by using the IRK.
LTK	Long Term Key. This key is used for encryption. The key distributed by Slave is used for encryption.
GAP	Generic Access Profile. This profile defines the functions for two devices to discover each other and establish a connection.
GATT	Generic Attribute Profile. This profile defines a Service and the procedures to access the Service. This profile has Server role and Client role. Server role provides Service, and Client role make access (ex. Service discovery, write request) to the Service.

1. Overview

Security Library provides APIs to ease the usage of security features provided by BLE protocol stack. Security Library shall use with BLE protocol stack V1.20 or later. The library can be used with Central / Peripheral. Also, can be used on Embedded / Modem Configuration.

The following features of Security Library makes it easier to provide security features than using rBLE API.

- Generates Passkey / LTK / IRK automatically
- Manages security information automatically
- Execute privacy functionality automatically
- Execute appropriate procedure depends on security status or service request error with a peer device

Table 1-1 shows the overview of security features provided by Security Library.

Table 1-1 Security Features

Function	Description
Pairing	Pairing is executed to establish keys used for encryption or privacy. After completing pairing, BLE communication is encrypted. There are two types of pairing. • Unauthenticated Pairing which have no protection against MITM • Authenticated Pairing which have a protection against MITM
Encryption	Encrypt BLE communication. By encryption, BLE communication is protected from an attacker device. To enable encryption, security information should be exchanged by pairing in advance.
Bonding	The act of storing security information. The stored security information is used subsequent encryption or privacy.
Privacy	The feature that reduces the ability to track a device by using RPA. Even with a device using RPA, the peer device can identify the device by exchanging security information with the peer device during pairing.

This document has following structures. Section 2 describes how to use Security Library. You should read through the section before you use Security Library. Section 3 describes API interface. Section 4 describes the procedure to introduce Security Library into existing project. Section 5 describes the behavior of Security Library. You should read this section when you want to understand Security Library internal behavior.

2. Security Library Usage

This section describes how to use Security Library. Security Library has APIs that required to execute, and APIs that execute per Application needs. Regarding the usage of each API, refer the section number describes in the "Refer" column.

Table 2-1 shows APIs that required to execute when you use Security Library.

Table 2-1 APIs that required to execute

Procedure	API		Refer
Initialization	Function	SecLib_Init	2.1
	Event	SECLIB_EVENT_INIT_COMP	
Set security parameters	Function	SecLib_Set_Param	2.2
	Event	SECLIB_EVENT_SET_PARAM	

Table 2-2 shows APIs that execute per Application needs.

Table 2-2 APIs that execute per Application needs

Procedure	API		Refer
Pairing / Encryption	Function	SecLib_Start_Encryption SecLib_Pairing_Req_Resp SecLib_Passkey_Req_Resp	2.3
	Event	SECLIB_EVENT_PAIRING_COMP SECLIB_EVENT_ENC_COMP SECLIB_EVENT_PAIRING_REQ SECLIB_EVENT_PASSKEY_IND SECLIB_EVENT_PASSKEY_REQ	
Service Request Error Handling	Function	SecLib_SrvcReq_Error_Resp	2.4
	Event	-	
Security Information Management	Function	SecLib_Delete_Bonding_Info	2.5
	Event	SECLIB_EVENT_DELETE_BONDING_INFO_COMP	

Security Library API have similar behavior to rBLE API. The result of the call is reported to a specified callback function as an event. You must not call another function before being reported the event for the previous function call.

Security Library is implemented using rBLE API. Security Library realizes security features by using rBLE functions. Security Library has a callback function to receive rBLE event. The callback handles security related rBLE events, but security un-related rBLE events are reported to Application.

2.1 Initialization

To use Security Library, you need to initialize Security Library first. The initialization is executed by SecLib_Init function. The result is reported as SECLIB_EVENT_INIT_COMP event.

In Application without Security Library, the initialization is executed by RBLE_GAP_Reset function. But when you use Security Library, SecLib_Init function shall be used instead of RBLE_GAP_Reset function. RBLE_GAP_Reset function is internally called in SecLib_Init function.

Figure 2-1 shows the usage example of SecLib_Init function. SecLib_Init function shall be called after rBLE mode is in RBLE_MODE_ACTIVE.

```
static void app_seclib_callback(SECLIB_EVENT *event)
{
    switch (event->type) {
/* ... skip ... */
    case SECLIB_EVENT_INIT_COMP:
        if (event->param.status == RBLE_OK) {
            /* SecLib_Init has finshed with OK. */
        }
        else {
            /* SecLib_Init has finshed with Error. */
        }
        break;
/* ... skip ... */
    }
}

static void app_callabck(RBLE_MODE mode)
{
    switch (mode) {
/* ... skip ... */
    case RBLE_MODE_ACTIVE:
        SecLib_Init(&app_gap_callback, NULL,  &app_seclib_callback);
        break;
/* ... skip ... */
    }
}

BOOL RBLE_App_Init(void)
{
/* ... skip ... */
    RBLE_Init(&app_callback);
/* ... skip ... */
}
```

Figure 2-1 SecLib_Init function usage example

2.2 Set Security Parameters

After completing the initialization, you need to set security parameters. SecLib_Set_Param function is used to set the security parameters. The result is reported as SECLIB_EVENT_SET_PARAM_COMP event.

SecLib_Set_Param function also can be used to change security parameters.

Figure 2-2 shows the usage example of SecLib_Set_Param function. SecLib_Set_Param function shall be called after SECLIB_EVENT_INIT_COMP event.

```
static SECLIB_PARAM app_sec_param = {
    RBLE_MASTER,                /* role */
    RBLE_AUTH_REQ_MITM_BOND,    /* auth_req */
    RBLE_IO_CAP_KB_ONLY,        /* iocap */
    TRUE,                       /* rpa_generate */
};

static void app_seclib_callback(SECLIB_EVENT *event)
{
    switch (event->type) {
/* ... skip ... */
    case SECLIB_EVENT_INIT_COMP:
        if (event->param.status == RBLE_OK) {
            SecLib_Set_Param(&app_sec_param);
        }
        else {
            /* SecLib_Init has finished with Error. */
        }
        break;

    case SECLIB_EVENT_SET_PARAM_COMP:
        if (event->param.status == RBLE_OK) {
            /* SecLib_Set_Param has finished with OK. */
        }
        else {
            /* SecLib_Set_Param has finished with Error. */
        }
        break;
/* ... skip ... */
    }
}
```

Figure 2-2 SecLib_Set_Param function usage example

Table 2-3 shows security parameter list. The security parameters are defined as SECLIB_PARAM structure.

Table 2-3 Security Parameters

Security Parameters	Descriptions	Refer
role	Role	2.2.1
auth_req	Authentication Requirement	2.2.2
iocap	IO Capabilities	2.2.3
rpa_generate	Privacy Feature	2.2.4

2.2.1 Role

Depends on the device's role, set either of the value shown in Table 2-4 to "role" field.

Table 2-4 Role settings

Value	Descriptions
RBLE_MASTER	The device is Central.
RBLE_SLAVE	The device is Peripheral.

2.2.2 Authentication Requirement

Depends on the security requirement of the device, set either of the value in Table 2-5 to "auth_req" field. This setting is used in section 2.3.1.

Table 2-5 Authentication Requirement settings

Value	Descriptions
RBLE_AUTH_MITM_BOND	Request Authentication Pairing. (This case has a protection against MITM, Use Passkey as the pairing method.)
RBLE_AUTH_NO_MITM_BOND	Request Unauthentication Pairing. (This case does not have a protection against MITM. Use Just Works as the pairing method.)

2.2.3 IO Capabilities

Depends on the Input / Output capabilities of the device, set either of the value shown in Table 2-6 to "iocap" field. This setting is used in section 2.3.1.

Table 2-6 IO Capabilities settings

Value	Descriptions
RBLE_IO_CAP_DISPLAY_ONLY	6-digits output capability (ex. Display)
RBLE_IO_CAP_DISPLAY_YES_NO	6-digits output capability (ex. Display) and Yes/No input capability (ex. Button)
RBLE_IO_CAP_KB_ONLY	0~9 number input capability (ex. Keyboard)
RBLE_IO_CAP_NO_INPUT_NO_OUTPUT	No input / output capabilities
RBLE_IO_CAP_KB_DISPLAY	0~9 number input capability (ex. Keyboard) and 6-digits output capability (ex. Display)

2.2.4 Privacy

Depends on the privacy requirement of the device, set either of the value shown in Table 2-7 to "rpa_generate" field. When you set TRUE to this field, RPA is used as the Bluetooth device address and the address is changed periodically. The address change interval is determined by GAP_RESOLVABLE_PRIVATE_ADDR_INTV.

Table 2-7 Privacy settings

Value	Descriptions
TRUE	Use RPA as Bluetooth device address and periodically change the address.
FALSE	Not use RPA.

2.3 Pairing / Encryption

Pairing / Encryption is executed by SecLib_Start_Encryption function. This function performs as follows depends on the security status with a peer device.

- If pairing with the peer device is not completed, it starts pairing. The pairing result is reported as SECLIB_EVENT_PAIRING_COMP event. After completing the pairing, the communication is encrypted.
- If pairing with the peer device have completed, enable encryption by using keys exchanged during pairing. The result of encryption is reported as SECLIB_EVENT_ENC_COMP event.

Security Library checks the pairing with the peer device have completed or not after connecting. This result is reported as SECLIB_EVENT_CHK_ADDR_COMP event.

Even when either pairing or encryption is performed, communication is encrypted after completion. But pairing uses a temporally key for encryption. To enable encryption using keys exchanged during pairing, re-call SecLib_Start_Encryption function after completing pairing with the peer device.

2.3.1 Pairing

When you call SecLib_Start_Encryption function without completing pairing with the peer device, pairing is started. Figure 2-3 shows the sequence of pairing. Pairing executes following procedures.

- Pairing request / response (Figure 2-3-A)
- Just Works / Passkey Entry (Figure 2-3-B)
- Exchange / store Security information (Figure 2-3-C)
- Pairing completion (Figure 2-3-D)

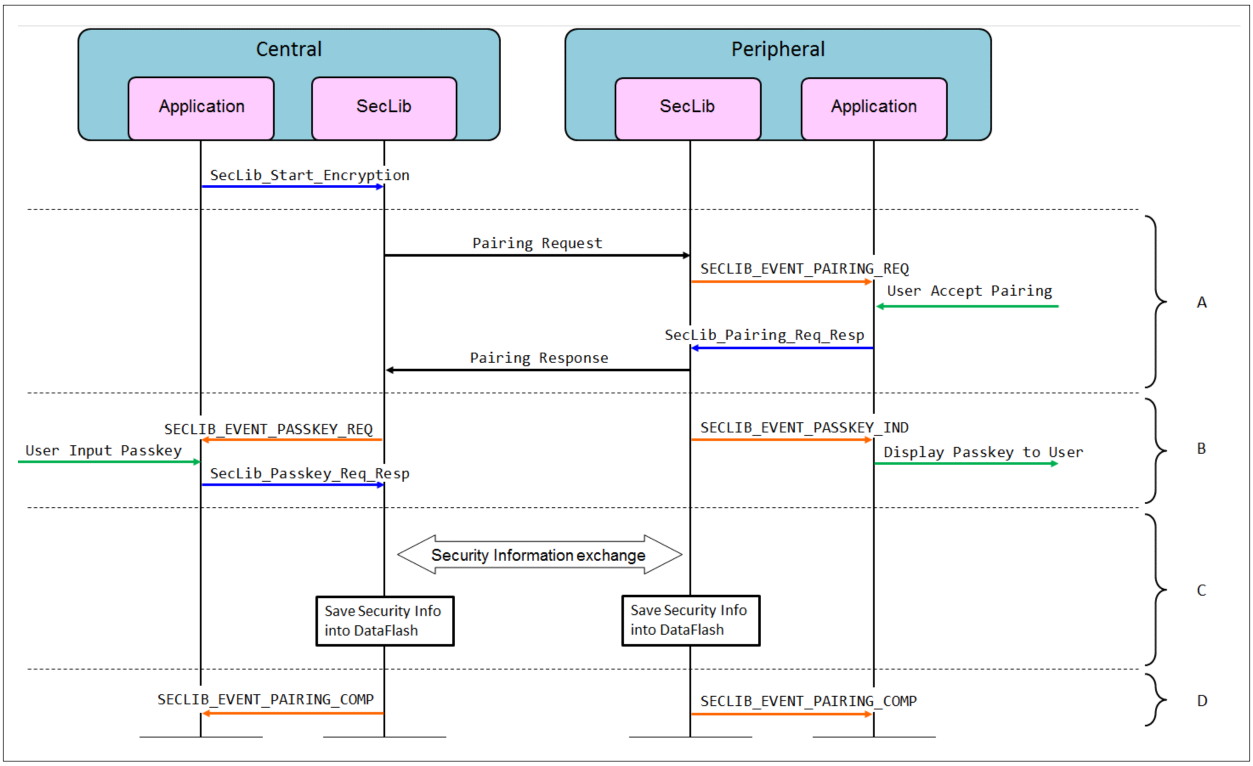


Figure 2-3 Pairing sequence

Figure 2-3 is the one example for pairing sequence. It is changed depends on security parameters settings.

(1) **Pairing request / response**

Pairing starts from sending pairing request to the peer device. The device receiving the pairing request responds to the request by sending pairing response. Pairing response includes decision about accept or decline the request.

Pairing request

When calling SecLib_Start_Encryption function without completing pairing, this function sends pairing request to the peer device.

Pairing response

When a device receives pairing request, it is reported to Application as SECLIB_EVENT_PAIRING_REQ event. Application responds to the event by SecLib_Pairing_Req_Resp function with the decision, accept or decline for the request. The decision should be selected by User.

Figure 2-4 shows the SecLib_Pairing_Req_Resp function usage example. In this example, when SECLIB_EVENT_PAIRING_REQ event is happened, report it to User to get the user decision, and responds to it by SecLib_Pairing_Req_Resp function.

```
static uint16_t conhdl;

/* This is pseudo function which receives user input (yes/no). */
static void app_user_input_yes_no(BOOL yes_no)
{
    SecLib_Pairing_Req_Resp(conhdl, yes_no);
}

static void app_seclib_callback(SECLIB_EVENT *event)
{
    switch (event->type) {
/* ... skip ... */
    case SECLIB_EVENT_PAIRING_REQ:
        /* Confirm to user whether accept the pairing request or not. */
        Printf("Accept pairing request?: (yes/no)\n");
        /* Save conhdl to use for SecLib_Pairing_Req_Resp argument. */
        conhdl = event->param.pairing_req.conhdl;
        break;
/* ... skip ... */
    }
}
```

Figure 2-4 SecLib_Pairing_Req_Resp function usage example

(2) **Passkey Entry / Just Works**

Security Library supports two pairing method, Just Works and Passkey Entry (OOB is not supported). Which pairing method is used during pairing is determined from the combination of Authentication Requirement and IO Capabilities of both devices. Table 2-8 and Table 2-9 shows the decision flow.

Table 2-8 Decision by Authentication Requirements

Peripheral \ Central	MITM_BOND	NO_MITM_BOND
MITM_BOND	Refer Table 2-9	Refer Table 2-9
NO_MITM_BOND	Refer Table 2-9	Unauthenticated *Just Works*

Table 2-9 Decision by IO Capabilities

Peripheral \ Central	DISPLAY_ ONLY	DISPLAY_ YES_NO	KB_ ONLY	NO_INPUT_ NO_OUTPUT	KB_ DISPLAY
DISPLAY_ ONLY	Unauthenticated *Just Works*	Unauthenticated *Just Works*	Authenticated *Passkey Entry* *Central: Input* *Peripheral: Display*	Unauthenticated *Just Works*	Authenticated *Passkey Entry* *Central: Input* *Peripheral: Display*
DISPLAY_ YES_NO	Unauthenticated *Just Works*	Unauthenticated *Just Works*	Authenticated *Passkey Entry* *Central: Input* *Peripheral: Display*	Unauthenticated *Just Works*	Authenticated *Passkey Entry* *Central: Input* *Peripheral: Display*
KB_ ONLY	Authenticated *Passkey Entry* *Central: Display* *Peripheral: Input*	Authenticated *Passkey Entry* *Central: Display* *Peripheral: Input*	Authenticated *Passkey Entry* *Central: Input* *Peripheral: Input*	Unauthenticated *Just Works*	Authenticated *Passkey Entry* *Central: Display* *Peripheral: Input*
NO_INPUT_N O_OUTPUT	Unauthenticated *Just Works*	Unauthenticated *Just Works*	Unauthenticated *Just Works*	Unauthenticated *Just Works*	Unauthenticated *Just Works*
KB_ DISPLAY	Authenticated *Passkey Entry* *Central: Display* *Peripheral: Input*	Authenticated *Passkey Entry* *Central: Display* *Peripheral: Input*	Authenticated *Passkey Entry* *Central: Input* *Peripheral: Display*	Unauthenticated *Just Works*	Authenticated *Passkey Entry* *Central: Display* *Peripheral: Input*

Just Works

Application have no things to do with Just Works pairing method (Table 2-3-B is not executed).

Passkey Entry

Depends on IO Capabilities setting, devices shall display passkey to User or receive passkey from User.

- SECLIB_EVENT_PASSKEY_IND event is reported to the device. The device shall output the passkey on the display to show it to User.

```
static void app_seclib_callback(SECLIB_EVENT *event)
{
    switch (event->type) {
/* ... skip ... */
    case SECLIB_EVENT_PASSKEY_IND:
        /* Display passkey to user. */
        Printf("Passkey: %06ld\n", event->param.passkey_ind.passkey);
        break;
/* ... skip ... */
    }
}
```

Figure 2-5 SECLIB_EVENT_PASSKEY_IND event usage example

- SECLIB_EVENT_PASSKEY_REQ is reported to the device to request User to input the passkey displayed on the peer device. The user input passkey is passed to Security Library through SecLib_Passkey_Req_Resp function.
  Figure 2-6 shows the example usage of SecLib_Passkey_Req_Resp function.

```
static uint16_t conhdl;

/* This is pseudo function which receives user input (passkey). */
static void app_user_input_passkey(uint32_t passkey)
{
    SecLib_Passkey_Req_Resp(conhdl, passkey);
}

static void app_seclib_callback(SECLIB_EVENT *event)
{
    switch (event->type) {
/* ... skip ... */
    case SECLIB_EVENT_PASSKEY_REQ:
        /* Request user to input passkey displayed on Peer Device. */
        Printf("Input passkey:\n");
        /* Save conhdl to use for SecLib_Passkey_Req_Resp argument. */
        conhdl = event->param.passkey_req.conhdl;
        break;
/* ... skip ... */
    }
}
```

Figure 2-6 SecLib_Passkey_Req_Resp function usage example

(3) **Exchange / store Security information**

Exchange security information used for encryption or privacy. Security information is stored in Data Flash by Bonding. The information will be used for subsequent encryption or privacy.

(4) **Pairing completion**

The completion of pairing is reported as SECLIB_EVENT_PAIRING_COMP event. Figure 2-7 shows the usage example of SECLIB_EVENT_PAIRING_COMP event.

```
static void app_seclib_callback(SECLIB_EVENT *event)
{
    switch (event->type) {
/* ... */
    case SECLIB_EVENT_PAIRING_COMP:
        if (event->param.pairing_comp.status == RBLE_OK) {
            /* Pairing has finished with OK. */
        }
        else {
            /* Pairing has finished with Error. */
        }
        break;
/* ... */
    }
}
```

Figure 2-7 SECLIB_EVENT_PAIRING_COMP event usage example

(5) **Pairing fail**

Pairing will fail with the following cases. To re-start pairing, the connection should be disconnected once, re-connect and then re-start pairing.

- When a pairing request is declined. The error is reported as SECLIB_EVENT_PAIRING_COMP event with the status is RBLE_SM_PAIR_ERR_PAIRING_NOT_SUPPORTED.
- When pairing is not completed with 30 seconds. The error is reported as SECLIB_EVENT_PAIRING_COMP event with the status is RBLE_ERR.
- When a user input wrong passkey. The error is reported as SECLIB_EVENT_PAIRING_COMP event with the status is RBLE_SM_PAIR_ERR_CFM_VAL_FAILED.
- When at least one of Central or Peripheral requesting Authenticate Pairing (Passkey), but Unauthenticated Pairing (Just Works) is selected as the pairing method due to IO Capabilities combination. The pairing is failed and the error is reported as SECLIB_EVENT_PAIRING_COMP event with the status is RBLE_SM_PAIR_ERR_AUTH_REQUIREMENTS.

2.3.2 Encryption

When you call SecLib_Start_Encryption function after completing pairing with the peer device, encryption is started. Figure 2-8 shows the sequence of encryption. Encryption executes following procedures.

- Start encryption (Figure 2-8-A)
- Store management data (Figure 2-8-B)
- Encryption completion (Figure 2-8-C)

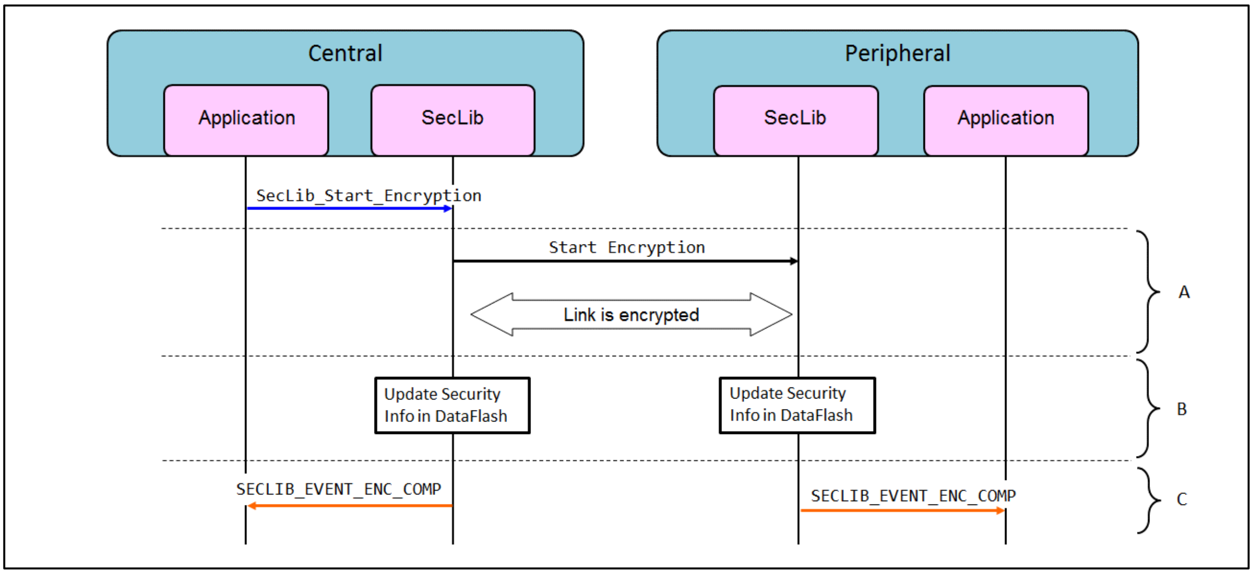


Figure 2-8 Encryption sequence

(1) **Start Encryption**

Encryption is executed with keys exchanged during pairing.

(2) **Store management data**

As described in section 2.5.1, information for LRU is updated.

(3) **Encryption completion**

When encryption is completed, SECLIB_EVENT_ENC_COMP event is reported. Figure 2-9 shows SECLIB_EVENT_ENC_COMP event usage example. This example checks encryption result.

```
static void app_seclib_callback(SECLIB_EVENT *event)
{
    switch (event->type) {
/* ... */
    case SECLIB_EVENT_ENC_COMP:
        if (event->param.enc_comp.status == RBLE_OK) {
            /* Encryption has finished with OK. */
        }
        else {
            /* Encryption has finished with Error. */
        }
        break;
/* ... */
    }
}
```

Figure 2-9 SECLIB_EVENT_ENC_COMP event usage example

2.4 Service Request Error Response

When one device start service request, such as GATT Write, but the request is failed due to the permission setting of the characteristic or security status with the peer device, a service request error will be occurred.

SecLib_SrvcReq_Error_Resp function executes pairing (2.3.1) or encryption (2.3.2) depend on the security status. If the security status does not permit the procedure, it returns an error. Table 2-10 shows SecLib_SrvcReq_Error_Resp function behavior of Security Library included in sample programs depending on each combination of security status.

Table 2-10 Security request error and action depends on security status

Service Request Error	Pairing	Encryption	Action
Insufficient Authentication	None	Disabled	Start pairing. (Execute Figure 2-3 A~D)
	None	Enabled	Return RBLE_STATUS_ERROR (Never be happened.)
	Unauthenticated	Disable	Start encryption. If it fails start pairing.[Note 1] (Execute Figure 2-8 A~C. If encryption failed, Figure 2-8 C is not reported and execute Figure 2-3 A~D.)
	Unauthenticated	Enabled	Start Authenticated Pairing. (Execute Figure 2-3 A~D with a MITM protection)
	Authenticated	Disabled	Start encryption. If it fails start pairing. [Note 1] (Execute Figure 2-8 A~C. If encryption failed, Figure 2-8 C is not reported and execute Figure 2-3 A~D.)
	Authenticated	Enabled	Return RBLE_ERR. (When the peer device requests LE Secure Connection which is not supported by RL78/G1D.)
Insufficient Encryption	None	Disabled	Start pairing. (Execute Figure 2-3 A~D)
	None	Enabled	Return RBLE_STATUS_ERROR. (Never be happened.)
	Unauthenticated	Disable	Start encryption. If it fails start pairing. [Note 1] (Execute Figure 2-8 A~C. If encryption failed, Figure 2-8 C is not reported and execute Figure 2-3 A~D.)
	Unauthenticated	Enabled	Return RBLE_STATUS_ERROR. (Never be happened.)
	Authenticated	Disabled	Start encryption. If it fails start pairing. [Note 1] (Execute Figure 2-8 A~C. If encryption failed, Figure 2-8 C is not reported and execute Figure 2-3 A~D.)
	Authenticated	Enabled	Return RBLE_STATUS_ERROR. (Never be happened.)

Note 1: If you connect to a malicious device with the same BD address as the paired device, the encryption will fail because the malicious device does not have the encryption key. The included Security Library automatically starts pairing when encryption fails, and pairing with the malicious device may be established. To avoid this vulnerability, it is recommended to follow Section 6.2 to prevent pairing from starting automatically when encryption fails.

Figure 2-10 shows the usage example of SecLib_SrvcReq_Error_Resp function. This example set GATT Write ATT code to att_code argument.

```
static void app_gatt_callback(RBLE_GATT_EVENT *event)
{
    switch (event->type) {
/* ... skip ... */
    case RBLE_GATT_EVENT_WRITE_CHAR_RESP:
        SecLib_SrvcReq_Error_Resp(conhdl, event->param.write_char_resp.att_code);
        break;
/* ... skip ... */
    }
}
```

Figure 2-10 SecLib_SrvcReq_Error_Resp function usage example

2.5 Security Information Management

Security information is stored to Data Flash when pairing / encryption is completed. Application can delete the security information by calling Security Library API.

2.5.1 Saving Security Information

Security Library saves security information when pairing / encryption is completed. Table 2-11 shows security information items. If Application try to save more bonding information than CFG_SECLIB_BOND_NUM, depend on LRU algorithm (Least Recently Used), the oldest bonding information is overwritten by a new one. To execute LRU, access information of each bonding information is also saved when pairing / encryption is completed.

Table 2-11 Security Information

Data	Description
Local device IRK	LTK generated by the local device.
Management Data	Record of the access for each bonding information. This is used for LRU algorithm.
Bonding information	Security information for each peer devices, see Table 2-12. (The number of storable bonding information on Data Flash is determined by CFG_SECLIB_BOND_NUM.)

Table 2-12 Bonding Information

Data	Description
Security Property	Security Level with the peer device.
Key Size	LTK key size.
Peer Device Address	The address of the peer device.
Peer Device Address Type	The address type of the peer device.
Peer Device IRK	IRK generated by the peer device.
Peer Device LTK	LTK generated by the peer device.
Local Device LTK	LTK generated by the local device.

2.5.2 Deletion of Security Information

Security information will be deleted with the following cases.

- When Application executes SecLib_Delete_Bonding_Info function. The specified bonding information is deleted. The result is reported as SECLIB_EVENT_DELETE_BONDING_INFO_COMP.
- When bonding information is stored more than CFG_SECLIB_BOND_NUM and try to save new one. The least recently used bonding information is overwritten by a new one.
- When encryption failed. The failed bonding information is deleted.[Note 1] In this case, as described in "Bluetooth Core Specification 4.2 Vol 3 10.6 Encryption Procedure", RPA should be changed by calling SecLib_Set_Param function with the "rpa_generate" field is TRUE.
  Note 1: If you follow section 6.2, the Security Library will not delete the bonding information.
- The local IRK included in security information is deleted only when all bonding information are deleted.

3. Interface Definitions

3.1 Functions

3.1.1 SecLib_Init

```
RBLE_STATUS SecLib_Init(RBLE_GAP_EVENTHANDLER gap_callback,
                        RBLE_VS_EVENTHANDLER vs_callback,
                        SECLIB_EVENTHANDLER lib_callback)
```

- This function initializes Security Library.
- The result is reported by SECLIB_EVENT_INIT_COMP event.
- This function shall be called after rBLE mode is in RBLE_MODE_ACTIVE.
- This function shall be called instead of RBLE_GAP_Reset function. RBLE_GAP_Reset function is called in this function. Application shall not call RBLE_GAP_Reset directly.
- RBLE_VS_Enable is called in this function. Application shall not call RBLE_VS_Enable function directly. Vendor Specific functions can be used after SECLIB_EVENT_INIT_COMP event.

Parameter:	
gap_callback	The callback used to report GAP Event.
vs_callback	The callback used to report Vendor Specific Event.
seclib_callback	The callback used to report Security Library Event.
Return:	
RBLE_OK	Success.
RBLE_PARAM_ERR	gap_callback or seclib_callback is NULL.

3.1.2 SecLib_Set_Param

```
RBLE_STATUS SecLib_Set_Param(SECLIB_PARAM *param)
```

- This function set security parameters.
- The result is reported by SECLIB_EVENT_SET_PARAM_COMP event.
- This function can be used to change security parameters.
- This function cannot be used when connecting with peer devices.

Parameter:	
param	Security Parameters
Return:	
RBLE_OK	Success.
RBLE_PARAM_ERR	param is NULL.
RBLE_STATUS_ERROR	This function is called before SECLIB_EVENT_INIT_COMP event occurred. This function is called during in connection with peer devices.

3.1.3 SecLib_Start_Encryption

```
RBLE_STATUS SecLib_Start_Encryption(uint16_t conhdl)
```

- If pairing is not completed with the peer device, execute pairing. The result is reported as SECLIB_EVENT_PAIRING_COMP.
- If pairing is already completed with the peer device, execute encryption. The result is reported as SECLIB_EVENT_ENC_COMP.

Parameter:	
conhdl	Connection Handle
Return:	
RBLE_OK	Success.
RBLE_PARAM_ERR	conhdl is invalid.
RBLE_STATUS_ERROR	This function is called before SECLIB_EVENT_SET_PARAM event. This function is called before SECLIB_EVENT_CHK_ADDR_COMP event.

3.1.4 SecLib_Pairing_Req_Resp

```
RBLE_STATUS SecLib_Pairing_Req_Resp(uint16_t conhdl, BOOL accept)
```

- This function responses to pairing request.
- This function shall be used only for responding SECLIB_EVENT_PAIRING_REQ event. Application shall not call this function in another situation.

Parameter:	
conhdl	Connection Handle.
accept	TRUE (Accept the pairing request) or FALSE (Decline the pairing request).
Return:	
RBLE_OK	Success.
RBLE_PARAM_ERR	conhdl is invalid.

3.1.5 SecLib_Passkey_Req_Resp

```
RBLE_STATUS SecLib_Passkey_Req_Resp(uint16_t conhdl, uint32_t passkey)
```

- This function responses to passkey request during pairing.
- This function shall be used only for responding SECLIB_EVENT_PASSKEY_REQ event. Application shall not call this function in another situation.

Parameter:	
conhdl	Connection Handle
passkey	Passkey
Return:	
RBLE_OK	Success.
RBLE_PARAM_ERR	conhdl is invalid.

3.1.6 SecLib_SrvcReq_Error_Resp

```
RBLE_STATUS SecLib_SrvcReq_Error_Resp(uint16_t conhdl, uint8_t att_err)
```

- This function executes pairing / encryption per the service request error.
- This function can handle Insufficient Authentication, Insufficient Encryption. Application calls this function by set the service request error to att_err argument.

Parameter:

conhdl	Connection Handle	
att_err	RBLE_ATT_ERR_INSUFF_AUTHEN	Handle Insufficient Authentication error.
	RBLE_ATT_ERR_INSUFF_ENC	Handle Insufficient Encryption error.
	RBLE_ATT_ERR_NO_ERROR	Do nothing.

Return:

RBLE_OK	Success.
RBLE_PARAM_ERR	att_err is invalid. conhdl is invalid.
RBLE_STATUS_ERROR	This function is called before SECLIB_EVENT_SET_PARAM_COMP event. This function is called before SECLIB_EVENT_CHK_ADDR_COMP event. Invalid security status.
RBLE_ERR	The peer device requests Authenticated pairing, but the local device only has RBLE_IO_CAP_NO_INPUT_NO_OUTPUT IO Capabilities. Authenticated pairing is already completed with the peer device, and Insufficient Authentication is occurred (The peer device requests LE Secure Connection.)

3.1.7 SecLib_Delete_Bonding_Info

```
RBLE_STATUS SecLib_Delete_Bonding_Info(SECLIB_DELETE_TARGET target)
```

- Delete bonding information from Data Flash.
- The result is reported by SECLIB_EVENT_DELETE_BONDING_INFO_COMP event.

Parameter:

target	SECLIB_DELETE_ALL_BONDS	Delete all bonding information.
	SECLIB_DELETE_ALL_BUT_ACTIVE_BOND	Delete bonding information for devices not in connection.

Return:

RBLE_OK	Success.
RBLE_PARAM_ERR	target is invalid.
RBLE_STATUS_ERROR	This function is called before SECLIB_EVENT_INIT_COMP event.
RBLE_BUSY	Failed to access Data Flash.

3.1.8 SecLib_Rand

```
uint16_t SecLib_Rand(void)
```

- This function generates 16-bits random number.
- This function is used in Security Library to generate Passkey / LTK / IRK.
- This function shall be defined by Application.

3.2 Events

3.2.1 SECLIB_EVENT_INIT_COMP

SECLIB_EVENT_INIT_COMP	
RBLE_STATUS status	The result of SecLib_Init function call.

Value	Descriptions
RBLE_OK	Success.
RBLE_ERR	Fail.

3.2.2 SECLIB_EVENT_SET_PARAM_COMP

SECLIB_EVENT_SET_PARAM_COMP	
RBLE_STATUS status	The result of SecLib_Set_Param function call.

Value	Descriptions
RBLE_OK	Success.
RBLE_ERR	Fail.

3.2.3 SECLIB_EVENT_PAIRING_COMP

SECLIB_EVENT_PAIRING_COMP		
RBLE_STATUS	status	The result of pairing. <table><tr><th>Value</th><th>Descriptions</th></tr><tr><td>RBLE_OK</td><td>Success.</td></tr><tr><td>RBLE_ERR</td><td>Failed to save security information. Pairing is not completed within 30 seconds.</td></tr><tr><td>RBLE_SM_PAIR_ERR_CFM_VAL_FAILED</td><td>The input passkey is invalid.</td></tr><tr><td>RBLE_SM_PAIR_ERR_AUTH_REQUIREMENTS</td><td>Authentication requirement is not satisfied.</td></tr><tr><td>RBLE_SM_PAIR_ERR_PAIRING_NOT_SUPPORTED</td><td>Pairing request is declined.</td></tr></table>
uint16_t	conhdl	Connection Handle
uint16_t	sec_prop	Security Property of completed pairing. <table><tr><th>Value</th><th>Descriptions</th></tr><tr><td>RBLE_SMP_SEC_NONE</td><td>Pairing failed.</td></tr><tr><td>RBLE_SMP_UNAUTHENTICATED</td><td>Unauthenticated pairing.</td></tr><tr><td>RBLE_SMP_AUTHENTICATED</td><td>Authenticated pairing.</td></tr></table>

3.2.4 SECLIB_EVENT_ENC_COMP

SECLIB_EVENT_ENC_COMP		
RBLE_STATUS	status	The result of encryption. <table><tr><th>Value</th><th>Descriptions</th></tr><tr><td>RBLE_OK</td><td>Success.</td></tr><tr><td>RBLE_ERR</td><td>Fail.</td></tr></table>
uint16_t	conhdl	Connection Handle
uint16_t	sec_prop	Security Property of completed pairing. <table><tr><th>Value</th><th>Descriptions</th></tr><tr><td>RBLE_SMP_SEC_NONE</td><td>Pairing failed.</td></tr><tr><td>RBLE_SMP_UNAUTHENTICATED</td><td>Unauthenticated pairing.</td></tr><tr><td>RBLE_SMP_AUTHENTICATED</td><td>Authenticated pairing.</td></tr></table>

3.2.5 SECLIB_EVENT_PAIRING_REQ

SECLIB_EVENT_SET_PARAM_COMP		
uint16_t	conhdl	Connection Handle
uint8_t	auth_req	The peer device's Authentication Requirement.
uint8_t	iocap	The peer device's IO Capabilities. This filed is valid only when Central starts the pairing. When Peripheral starts the pairing, this field is invalid.

3.2.6 SECLIB_EVENT_PASSKEY_IND

SECLIB_EVENT_PASSKEY_IND		
uint16_t	conhdl	Connection Handle
uint32_t	passkey	Passkey

3.2.7 SECLIB_EVENT_PASSKEY_REQ

SECLIB_EVENT_PASSKEY_REQ		
uint16_t	conhdl	Connection Handle

3.2.8 SECLIB_EVENT_CHK_ADDR_COMP

SECLIB_EVENTCHK_ADDR_COMP		
uint8_t	status	The result of the confirmation which the pairing with the peer device is finished or not.
uint16_t	conhdl	Connection Handle

Status values for `status`:

Status	Description
RBLE_OK	Pairing is completed with the peer device.
RBLE_ERR	Pairing is not completed with the peer device.

3.2.9 SECLIB_EVENT_DELETE_BONDING_INFO_COMP

SECLIB_EVENT_DELETE_BONDING_INFO_COMP		
uint8_t	status	The result of the SecLib_Delete_Bonding_Info.

Status	Description
RBLE_OK	Success.
RBLE_ERR	Fail.

3.3 Definitions

3.3.1 SECLIB_EVENT_TYPE

```c
typedef enum {
    SECLIB_EVENT_INIT_COMP = 0x01,
    SECLIB_EVENT_SET_PARAM_COMP,
    SECLIB_EVENT_PAIRING_COMP,
    SECLIB_EVENT_ENC_COMP,
    SECLIB_EVENT_CHK_ADDR_COMP,
    SECLIB_EVENT_DELETE_BONDING_INFO_COMP,
    SECLIB_EVENT_PAIRING_REQ,
    SECLIB_EVENT_PASSKEY_IND,
    SECLIB_EVENT_PASSKEY_REQ,
} SECLIB_EVENT_TYPE;
```

3.3.2 SECLIB_EVENT

```c
typedef struct seclib_event_t {
    SECLIB_EVENT_TYPE type;
    union {
        /* SECLIB_EVENT_INIT_COMP */
        /* SECLIB_EVENT_SET_PARAM_COMP */
        /* SECLIB_EVENT_DELETE_BONDING_INFO_COMP */
        RBLE_STATUS status;

        /* SECLIB_EVENT_ENC_COMP */
        struct enc_t {
            RBLE_STATUS                 status;
            uint16_t                    conhdl;
            uint8_t                     sec_prop;
        } enc;

        /* SECLIB_EVENT_PAIRING_COMP */
        struct pairing_t {
            RBLE_STATUS                 status;
            uint16_t                    conhdl;
            uint8_t                     sec_prop;
        } pairing;

        /* SECLIB_EVENT_CHK_ADDR_COMP */
        struct chk_addr_t {
            RBLE_STATUS status;
            uint16_t    conhdl;
        } chk_addr;

        /* SECLIB_EVENT_PAIRING_REQ */
        struct pairing_req_t {
            uint16_t conhdl;
            uint8_t  auth_req;
            uint8_t  iocap;
        } pairing_req;

        /* SECLIB_EVENT_PASSKEY_IND */
        struct passkey_ind_t {
            uint16_t conhdl;
            uint32_t passkey;
        } passkey_ind;

        /* SECLIB_EVENT_PASSKEY_REQ */
        struct passkey_req_t {
            uint16_t conhdl;
        } passkey_req;
```

```
    } param;
} SECLIB_EVENT;
```

3.3.3 SECLIB_PARAM

```
typedef struct seclib_param_t {
    uint8_t role;
    uint8_t auth_req;
    uint8_t iocap;
    BOOL    rpa_generate;
} SECLIB_PARAM;
```

3.3.4 SECLIB_DELETE_TARGET

```
typedef enum {
    SECLIB_DELETE_ALL_BONDS = 0x01,
    SECLIB_DELETE_ALL_BUT_ACTIVE_BOND,
} SECLIB_DELETE_TARGET;
```

3.3.5 SECLIB_EVENT_HANDLER

```
typedef void (*SECLIB_EVENT_HANDLER)(SECLIB_EVENT *event);
```

3.3.6 CFG_SECLIB_DEBUG

When this macro is defined, Security Library debug log is enabled.

3.3.7 CFG_SECLIB_BOND_NUM

This macro defines the number of bonding information storable into Data Flash. This is configured from Development environment. Refer section 4.1.1 for the procedure.

4. How to Introduce Security Library into Existing Project

4.1 Procedure

This section describes the procedure to introduce Security Library into an existing project.

4.1.1 Project settings

(a) **CFG_SECLIB_BOND_NUM**

Configure the number of bonding information storable in DataFlash by setting CFG_SECLIB_BOND_NUM. The procedure to set the macro is shown in the "Add include path" section.

(b) **Add source code**

Add Security Library source code (seclib.c, secdb.c) into the project.

(c) **Add include path**

Add include path to Security Library header files (seclib.h, secdb.h). The addition of the include path is set on the development environment. The following shows the procedure on each development environment.

CS+

Double click on "CA78K0R (Build Tool)" or "CC-RL (Build Tool)" in the project tree. A setting menu is shown then select "Common Option" tab. Set "Macro definition" and "Include file directories".

e^2 studio

Right click on the project shown in "Project Explorer" and select "Renesas Tool Settings" from the dropdown menu. Select "C/C++ Build" → "Settings" → "Compiler" → "Source". Set "Macro definition" and "Include file directories".

IAR Embedded Workbench

Right click on the project shown in Workspace and select "options…" from the dropdown menu. Select "C/C++ Compiler" → "Preprocessor". Set "Defined symbols: (one per line)" and "Additional include directories: one per line)".

4.1.2 Security Library

(1) **Delete unusable API / Event with Security Library**

The following APIs cannot be used in Application with Security Library. If these APIs are in use remove the implementation.

- All SM APIs (rBLE APIs that have prefix RBLE_SM).
- The following GAP or VS API.

GAP / VS Function	GAP Event
• RBLE_GAP_Reset GAP • RBLE_GAP_Set_Bonding_Mode • RBLE_GAP_Set_Security_Request • RBLE_GAP_Set_Random_Address • RBLE_GAP_Set_Privacy_Feature • RBLE_GAP_Start_Bonding • RBLE_GAP_Bonding_Info_Ind • RBLE_GAP_Bonding_Response • RBLE_GAP_Authorized_Ind • RBLE_VS_Enable	• RBLE_GAP_EVENT_RESET_RESULT • RBLE_GAP_EVENT_SET_BONDING_MODE_COMP • RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP • RBLE_GAP_EVENT_RPA_RESOLVED • RBLE_GAP_EVENT_SET_RANDOM_ADDRESS_COMP • RBLE_GAP_EVENT_SET_PRIVACY_FEATURE_COMP • RBLE_GAP_EVENT_BONDING_COMP • RBLE_GAP_EVENT_BONDING_REQ_IND

(2) **SecLib_Rand function**

Implement SecLib_Rand function in Application. Use the random number generation function provided by your Application or Device. In Embedded configuration, you can use rand function provided by standard library. In Modem Configuration, Host CPU shall provide a random number generation function.

```
uint16_t SecLib_Rand(void)
{
    return rand();
}
```

(3) **Security Library Callback**

Implement the callback function which receives Security Library event.

```
static void app_seclib_callback(SECLIB_EVENT *event)
{
    switch (event->type) {
/* ... skip ... */
    case SECLIB_EVENT_INIT_COMP:
        break;
/* ... skip ... */
    default:
        break;
    }
}
```

(4) **Replace RBLE_GAP_Reset with SecLib_Init function**

Call SecLib_Init function instead of RBLE_GAP_Reset function. If existing Application is using RBLE_GAP_Reset function, replace it with SecLib_Init function. The event to report the completion of initialization is also changed from RBLE_GAP_EVENT_RESET_RESULT event to SECLIB_EVENT_INIT_COMP event.

(5) **Other Security Library functions call**

Call Security Library functions by referring section 2 and 3.

4.1.3 Data Flash Library settings

For the preparation to save bonding information into Data Flash, add following data structures. On Modem Configuration, these are defined in RL78/G1D side.

(1) **renesas/src/driver/dataflash/eel_descriptor_t02.h**

Edit the file as the following.

- Adding EEL_ID_MD, EEL_ID_LD_IRK and EEL_ID_BOND_1 are mandatory.
- The number of EEL_ID_BOND_* (including EEL_ID_BOND_1) shall be greater or equal to CFG_SECLIB_BOND_NUM. The following is the case of CFG_SECLIB_BOND_NUM=8.
- Since the number of EEL_ID_BOND_* can be greater or equal to CFG_SECLIB_BOND_NUM, the following setting will also work with CFG_SECLIB_BOND_NUM=4. But memory region for 4 bonding information is wasted.

```
enum
{
    EEL_ID_BDA    = 0x01,
    EEL_ID_MD     = 0x02,   // <- Add (mandatory)
    EEL_ID_LD_IRK = 0x03,   // <- Add (mandatory)
    EEL_ID_BOND_1 = 0x04,   // <- Add (mandatory)
    EEL_ID_BOND_2 = 0x05,   // <- Add
    EEL_ID_BOND_3 = 0x06,   // <- Add
    EEL_ID_BOND_4 = 0x07,   // <- Add
    EEL_ID_BOND_5 = 0x08,   // <- Add
    EEL_ID_BOND_6 = 0x09,   // <- Add
    EEL_ID_BOND_7 = 0x0A,   // <- Add
    EEL_ID_BOND_8 = 0x0B,   // <- Add
    EEL_ID_END
};
```

(2) **renesas/src/driver/dataflash/eel_descriptor_t02.c**

Edit the file as the following.

- Include secdb.h header file.
- Adding SECDB_MD, SECDB_IRK are mandatory.
- Add SECDB_BONDs. The number of SECDB_BOND shall be equal to the number of EEL_ID_BOND_*. The following example is the case of CFG_SECLIB_BOND_NUM=8.

```
#include "secdb.h" <- Add (mandatory)

/* ... skip ... */

_EEL_CNST _EVENTfar const eel_u08 eel_descriptor[EEL_VAR_NO+2] =
{
  (eel_u08)(EEL_VAR_NO),          /* variable count   */
  (eel_u08)(BD_ADDR_LEN),         /* id=1: EEL_ID_BDA */
  (eel_u08)(sizeof(SECDB_MD)),    // <- Add (mandatory)
  (eel_u08)(sizeof(SECDB_IRK)),   // <- Add (mandatory)
  (eel_u08)(sizeof(SECDB_BOND)),  // <- Add (mandatory)
  (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
  (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
  (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
  (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
  (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
  (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
  (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
  (eel_u08)(0x00),                /* zero terminator  */
};
```

5. Internal Behavior Sequence

5.1 SecLib_Init

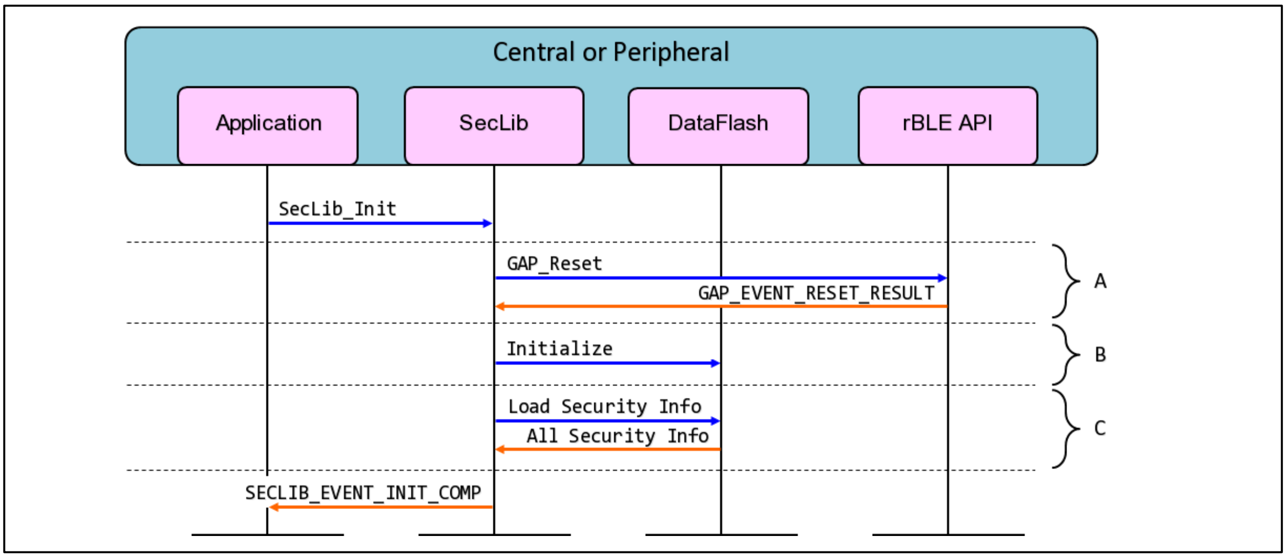


Figure 5-1 Initialization sequence

A) Reset GAP.

B) Initialize Data Flash.

C) Load security information from Data Flash and retain it on SRAM.

5.2 SecLib_Set_Param

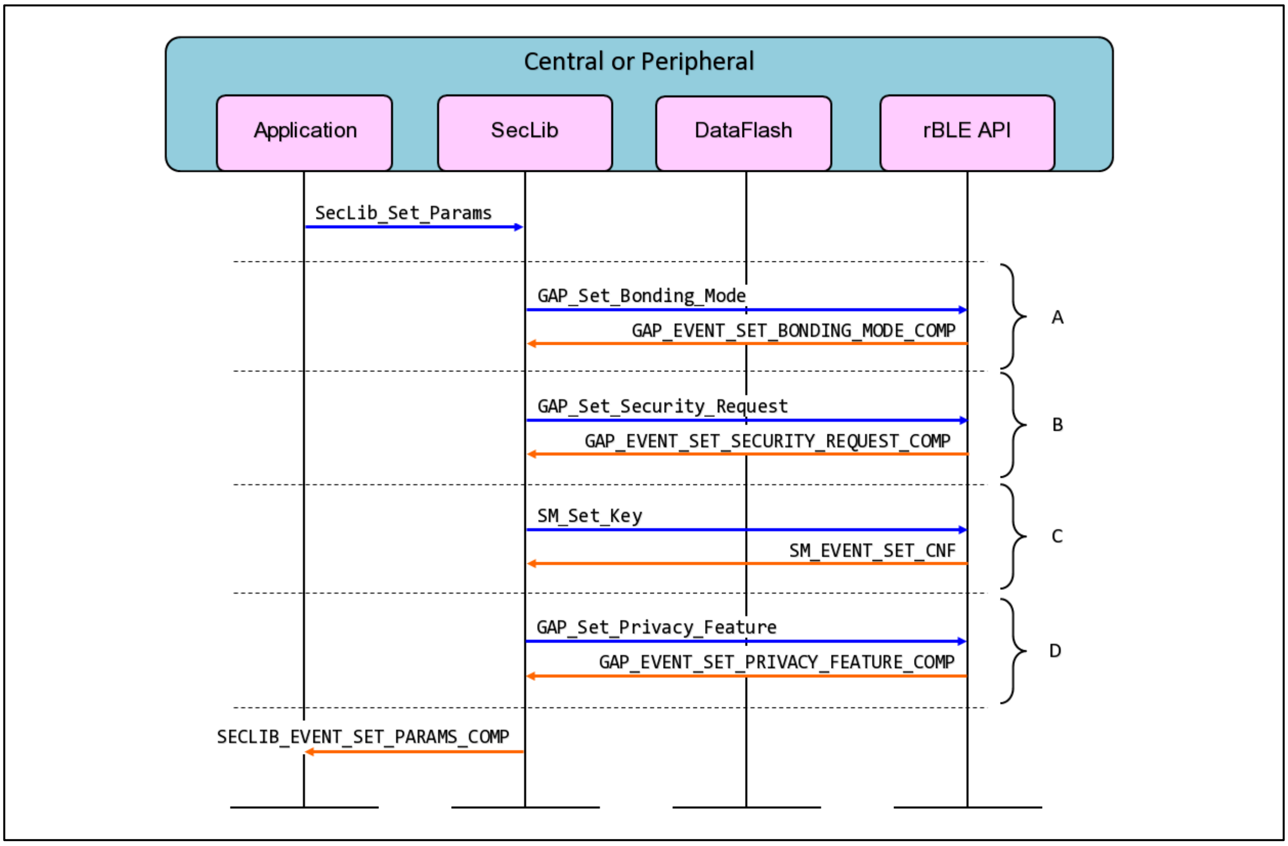


Figure 5-2 Set security parameters sequence

A) Enable bonding mode.

B) Set Authentication Requirement per “auth_req”.

C) Generate IRK if “rpa_generate” is TRUE.

D) Set Privacy Feature.

5.3 SecLib_Start_Encryption (Central Initiated, Pairing is not completed)

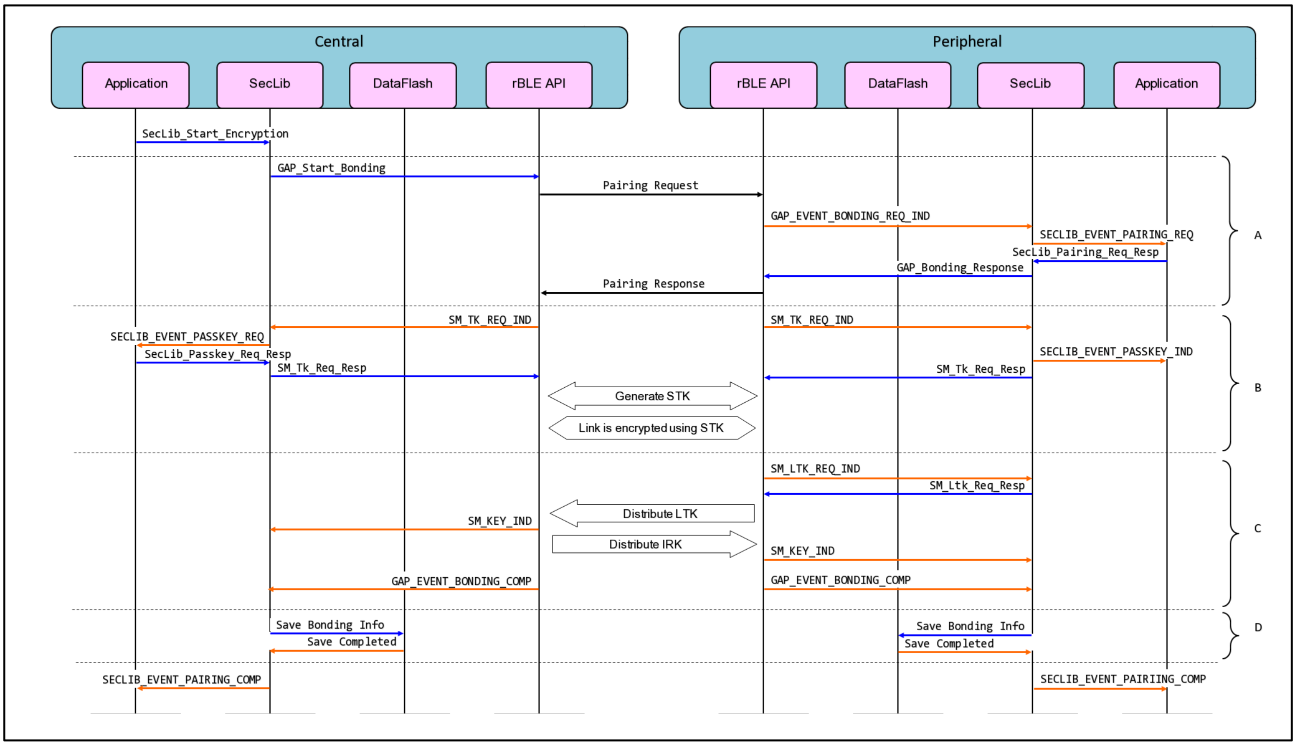


Figure 5-3 Central initiated Pairing sequence

A) Pairing Phase 1: Central requests pairing and Peripheral responds to the request.

B) Pairing Phase 2: Authenticate devices with Passkey Entry.

C) Pairing Phase 3: Exchange security information between Central / Peripheral.

D) After completing pairing, save bonding information.

5.4 SecLib_Start_Encryption (Peripheral initiated, Pairing is not completed)

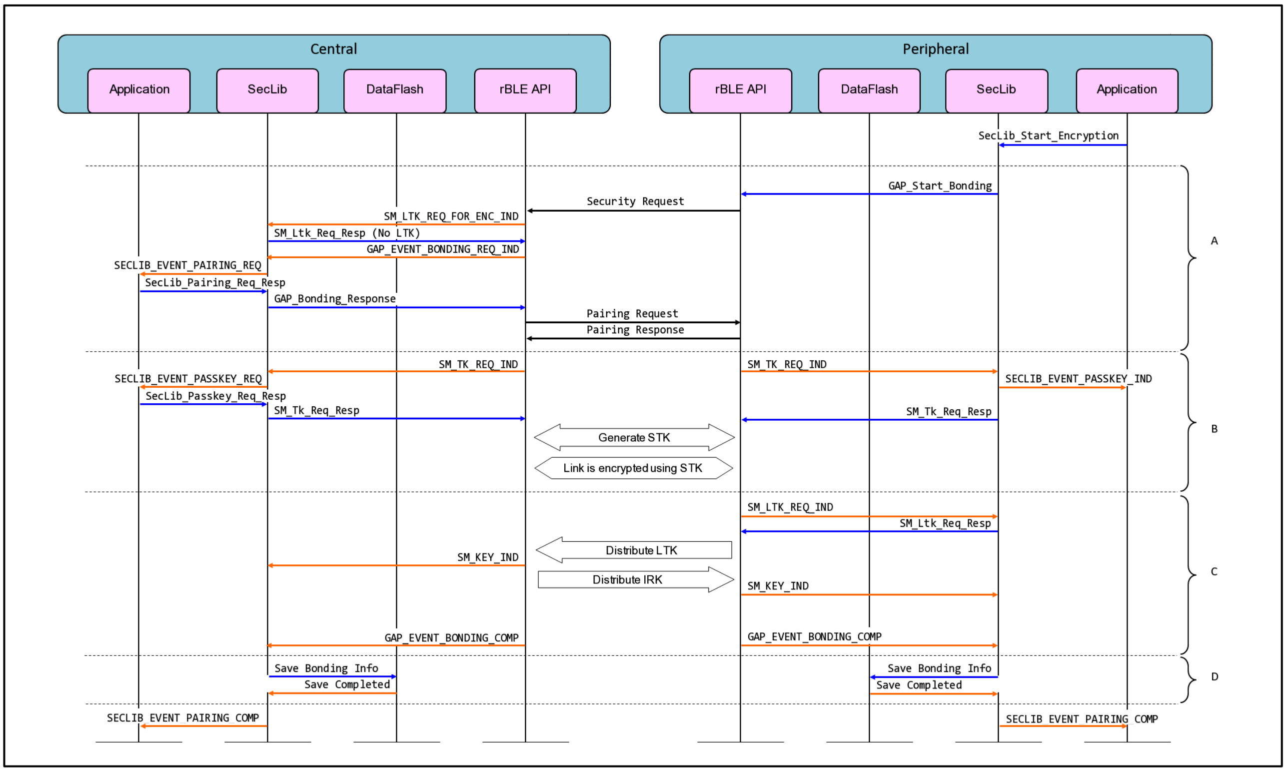


Figure 5-4 Peripheral initiated Pairing sequence

A) Pairing Phase 1: Peripheral requests to establish security to Central and Central starts pairing.

B) Pairing Phase 2: Authenticate devices with Passkey Entry.

C) Pairing Phase 3: Exchange security information between Central / Peripheral.

D) After completing pairing, save bonding information.

5.5 SecLib_Start_Encryption (Central initiated, Pairing is completed)

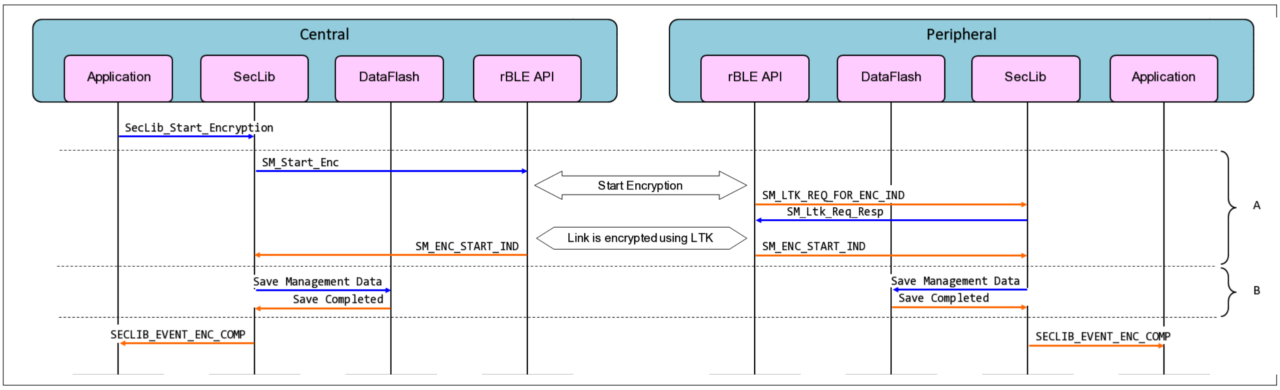


Figure 5-5 Central initiated Encryption sequence

A) Central starts encryption.

B) After encryption is completed, update Management Data used by LRU.

5.6 SecLib_Start_Encryption (Peripheral initiated, Pairing is completed)

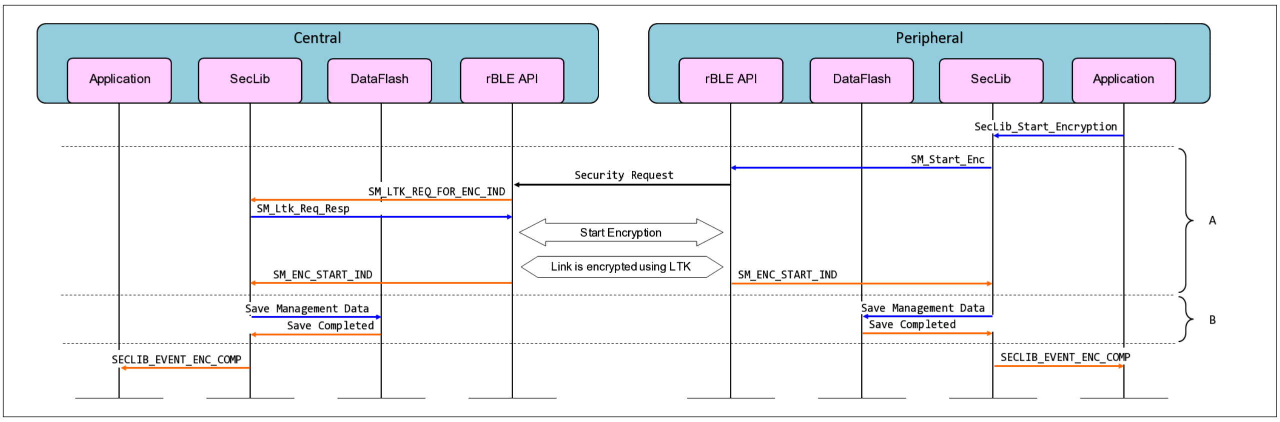
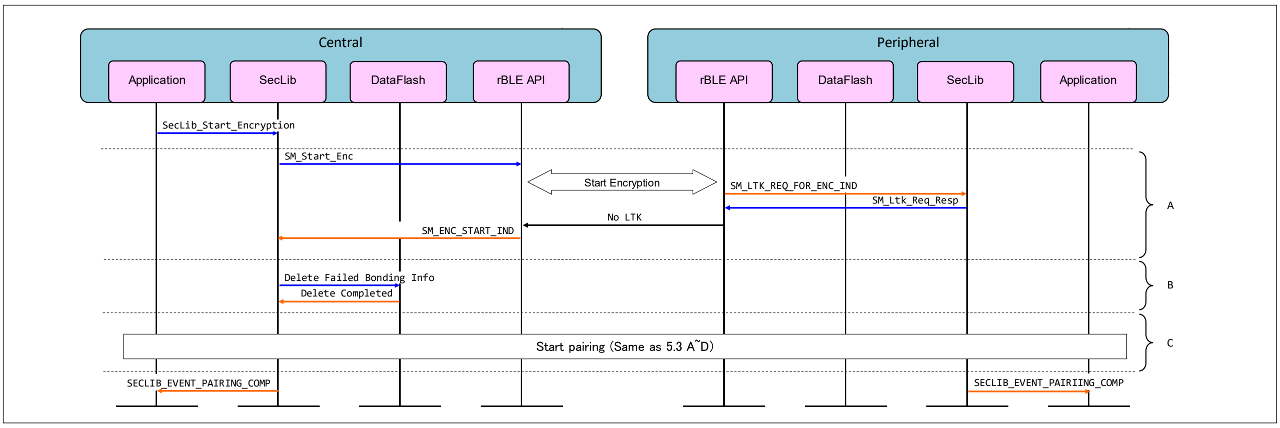


Figure 5-6 Central initiated Encryption sequence

A) Peripheral requests to establish security to Central. Central starts encryption by using keys exchanged during pairing with the Peripheral.

B) After encryption is completed, update Management Data used by LRU.

5.7 SecLib_Start_Encryption (Central initiated, Pairing is completed but Perpheral losts the bonding information)



A) Central starts pairing but Peripheral lost the bonding information, the encryption is failed.

B) Central deletes the bonding information just failed.[Note 1]

C) Central starts pairing. [Note 1]

Note 1: If you connect to a malicious device with the same BD address as the paired device, the encryption will fail because the malicious device does not have the encryption key. The included Security Library automatically starts pairing when encryption fails, and pairing with the malicious device may be established. To avoid this vulnerability, it is recommended to follow Section 6.2 to prevent pairing from starting automatically when encryption fails.

5.8 SecLib_Start_Encryption (Peripheral initiated, Pairing is completed but Central lost the bonding information)

Same sequence with section 5.4.

6. Appendix

6.1 ROM size, RAM size

Table 6-1 shows Security Library ROM / RAM usage. The value is changed depends on CFG_SECLIB_BOND_NUM. The following shows CFG_SECLIB_BOND_NUM=4 and 7 settings.

Table 6-1 ROM, RAM usage

Compiler	Central (4/7)		Peripheral (4/7)	
	ROM	RAM	ROM	RAM
CA78K0R V1.72	7,606 7,602	552 858	7,528 7,592	500 756
CC-RL V1.03.00	5,229 5,228	556 868	5,155 5,164	502 758
IAR EW V1.40.6	5,165 5,170	556 866	4,720 4,720	502 758
IAR EW V2.20.1	5,111 5,111	556 866	4,670 4,670	502 758

6.2 Vulnerability Avoidance Measures in the Event of Encryption Failure

If you connect to a malicious device with the same BD address as the paired device, the encryption will fail because the malicious device does not have the encryption key. The included Security Library automatically starts pairing when encryption fails, and pairing with the malicious device may be established. To avoid this vulnerability, it is recommended to prevent pairing from starting automatically when encryption fails below.

6.2.1 Sequence

Figure 6-1 shows the vulnerable sequence.

The problem is that if Central, which uses the original Security Library, connects to a malicious device that somehow obtains the same BD address as the paired Peripheral and pairs it with Just Works, the bonding information of the paired Peripheral will be automatically deleted.

To work around this problem, you need to make a change to notify the application of an event when encryption fails, as shown in Figure 6-2.

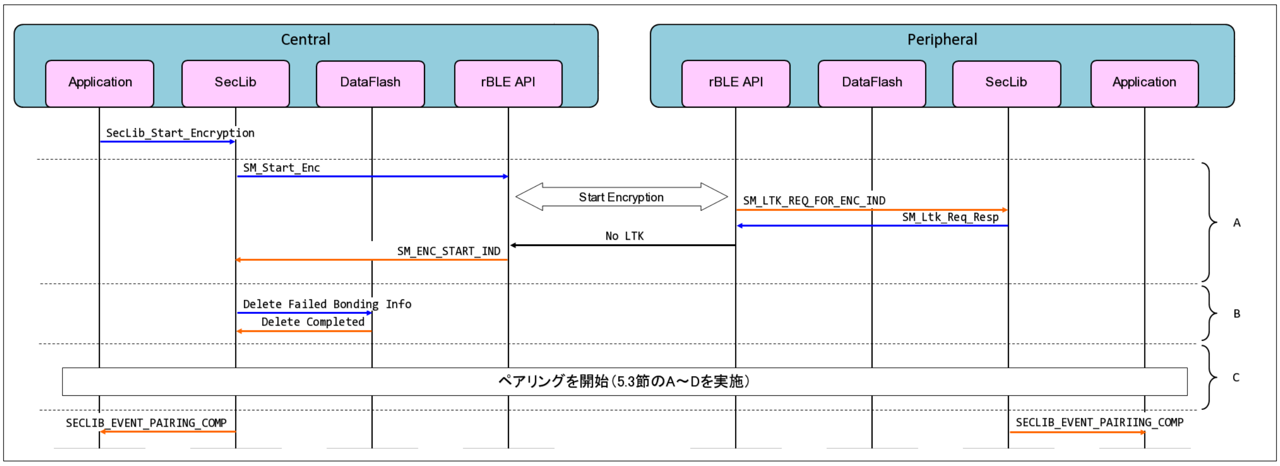


Figure 6-1 Vulnerable sequence

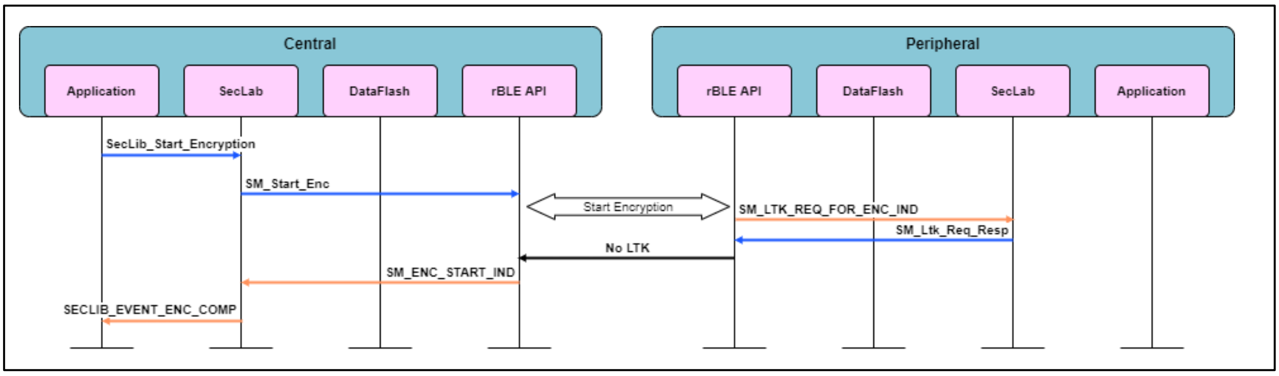


Figure 6-2 Sequence to avoid vulnerability

6.2.2 Vulnerability Avoidance Measures

To avoid vulnerability, change the application to notify an event when encryption fails.

The Security Library function that performs this process is seclib_enc_failed() on seclib.c, shown in Figure 6-3. Change the implementation of this function as shown in Figure 6-4.

```
/* Encryption has finished with error. */
static void seclib_enc_failed(uint16_t cidx)
{
    RBLE_STATUS status = RBLE_OK;
    uint16_t bidx = CON(cidx).bidx;
    int i;

    SECLIB_ERR("SecLib_Start_Encryption: Failed (encryption)", CON(cidx).con_state);

    CON(cidx).con_state &= ~SECLIB_CON_STATE_ENC_COMP;

    /* Delete failed bond info. */
    memset(&BOND(bidx), 0x00, sizeof(BOND(bidx)));

    /* Check whether other pairing is ongoing. */
    for (i = 0; i < CFG_CON; i++) {
        if (CON_STATE_DONE(SECLIB_CON_STATE_IN_PAIRING, i)) {
            break;
        }
    }

    /* If other pairing is not ongoing, delete bonding information. */
    if (i == CFG_CON) {
        info.db_cidx          = cidx;
        info.db_context       = SECLIB_DB_CONTEXT_ENC_DELTETE_COMP;
        info.db.md.data[bidx] = 0;
        status = SecDb_Save(&info.db, ((uint16_t)1 << bidx), FALSE, TRUE);
    }

    /* Event if failed to delete md, go ahead. */
    if ((i != CFG_CON) || (status != RBLE_OK)) {
        seclib_send_enc_comp_event(RBLE_ERR, CON(cidx).conhdl, RBLE_SMP_KSEC_NONE);
    }
}
```

Figure 6-3 Processing when encryption fails (no notification to the application, deletion of bond information)

```
/* Encryption has finished with error. */
static void seclib_enc_failed(uint16_t cidx)
{
    SECLIB_ERR("SecLib_Start_Encryption: Failed (encryption)", CON(cidx).con_state);

    CON(cidx).con_state &= ~SECLIB_CON_STATE_ENC_COMP;

    seclib_send_enc_comp_event(RBLE_ERR, CON(cidx).conhdl, RBLE_SMP_KSEC_NONE);
}
```

Figure 6-4 Processing when encryption fails (notifying the application)

Revision History

Rev.	Date	Description	
		Page	Summary
V1.00	Mar 1, 2017	-	Initial version
V1.01	Apr 28, 2025	40	Described how to avoid the vulnerability.
		17, 19, 38	Added note about vulnerabilities and added link to section 6.2

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. 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.
6. 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.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. 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.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by Renesas Electronics Corporation is under license. Other trademarks and registered trademarks are the property of their respective owners.

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/.