Bluetooth® Low Energy プロトコルスタック

Security Library

要旨

Security Library は、BLE プロトコルスタックが提供するセキュリティ機能を容易に使用するための API を提供します。Security Library は、BLE プロトコルスタック V1.20 以降と組み合わせて使用します。Security Library は、Central / Peripheral のどちらからでも使用可能です。また Embedded 構成 / Modem 構成のどちらからでも使用可能です。

動作確認デバイス

RL78/G1D

目次

1. 概要 4

2. Security Library の使用方法 5
2.1 Security Library の初期化 6
2.2 セキュリティパラメータの設定 7
2.2.1 ロールの設定 8
2.2.2 Authentication Requirement の決定 8
2.2.3 IO Capabilities の設定 8
2.2.4 プライバシーの設定 9
2.3 ペアリング/暗号化の実施 10
2.3.1 ペアリングの実施 11
2.3.2 暗号化の実施 16
2.4 サービス要求エラーへの対応 18
2.5 セキュリティ情報の管理 20
2.5.1 セキュリティ情報の保存 20
2.5.2 セキュリティ情報の削除 20

3. インタフェース定義 21
3.1 関数 21
3.1.1 SecLib_Init 21
3.1.2 SecLib_Set_Param 21
3.1.3 SecLib_Start_Encryption 22
3.1.4 SecLib_Pairing_Req_Resp 22
3.1.5 SecLib_Passkey_Req_Resp 22
3.1.6 SecLib_SrvcReq_Error_Resp 23
3.1.7 SecLib_Delete_Bonding_Info 23
3.1.8 SecLib_Rand 24
3.2 イベント 25
3.2.1 SECLIB_EVENT_INIT_COMP 25

3.2.2 SECLIB_EVENT_SET_PARAM_COMP 25
3.2.3 SECLIB_EVENT_PAIRING_COMP 26
3.2.4 SECLIB_EVENT_ENC_COMP 26
3.2.5 SECLIB_EVENT_PAIRING_REQ 27
3.2.6 SECLIB_EVENT_PASSKEY_IND 27
3.2.7 SECLIB_EVENT_PASSKEY_REQ 27
3.2.8 SECLIB_EVENT_CHK_ADDR_COMP 27
3.2.9 SECLIB_EVENT_DELETE_BONDING_INFO_COMP 27
3.3 定義 28
3.3.1 SECLIB_EVENT_TYPE 28
3.3.2 SECLIB_EVENT 28
3.3.3 SECLIB_PARAM 29
3.3.4 SECLIB_DELETE_TARGET 29
3.3.5 SECLIB_EVENT_HANDLER 29
3.3.6 CFG_SECLIB_DEBUG 29
3.3.7 CFG_SECLIB_BOND_NUM 29

4. プロジェクトへの組み込み方法 30
4.1 手順 30
4.1.1 プロジェクトの設定 30
4.1.2 Security Library の呼び出し部の実装 31
4.1.3 Data Flash Library の設定 32

5. 内部動作シーケンス 34
5.1 SecLib_Init 34
5.2 SecLib_Set_Param 35
5.3 SecLib_Start_Encryption（Central 開始，ペアリング未実施時） 36
5.4 SecLib_Start_Encryption（Peripheral 開始，ペアリング未実施時） 37
5.5 SecLib_Start_Encryption（Central 開始，ペアリング実施済） 38
5.6 SecLib_Start_Encryption（Peripheral 開始，ペアリング実施済） 38
5.7 SecLib_Start_Encryption（Central 開始、ペアリング実施済、Peripheral がボンディング情報を削除） 39
5.8 SecLib_Start_Encryption（Peripheral 開始、ペアリング実施済、Central がボンディング情報を削除） 39

6. Appendix 40
6.1 ROM / RAM 使用量 40
6.2 暗号化失敗時の脆弱性回避対策 41
6.2.1 シーケンス 41
6.2.2 脆弱性回避対策 42

用語

Term	Description
Role	ロール。それぞれのデバイスがプロファイルやサービスで規定される役割を持ちます。
Advertising	アドバタイジング。自デバイスの存在を通知するパケットをブロードキャスト送信する動作です。
Scanning	スキャニング。アドバタイジング中のデバイスを発見するための動作です。
Central	リンクレイヤと GAP で定義されているロールの一種。接続前にスキャニングや接続の開始を行っていたデバイスが接続後に Central になります。接続中は通信のタイミングを生成します。
Peripheral	リンクレイヤと GAP で定義されているロールの一種。接続前にアドバタイジングを実施していたデバイスが接続後に Peripheral になります。
MITM attack	Man In The Middle attack。中間者攻撃。自デバイスと対向デバイスの間に、悪意のある攻撃デバイスが介在し、通信データの盗聴や改ざんを行うこと。
Authentication	認証。セキュリティを確立するため実行されます。
Just Works	ペアリング方法のひとつ。キーボードやディスプレイなどの入出力装置を持たないデバイスや、高いセキュリティを必要としないデバイス間で行われるペアリング方法。ペアリングの過程において MITM から保護されません。ペアリングの過程で攻撃者が存在しない場合、機密性が確保できます。Unauthenticated Pairing。
Passkey Entry	ペアリング方法のひとつ。キーボードやディスプレイなどの入出力装置を持つデバイスが、互いに同じ 6 桁の数値を表示・入力するペアリング方法。MITM から保護されます。Authenticated Pairing。
OOB	Out Of Band。ペアリング方法のひとつ。Bluetooth 以外の（通信）方法により、認証に必要なデータを交換します。
RPA	Resolvable Private Address。デバイスを識別するデバイスアドレスの一種。悪意のあるデバイスからの追跡を困難にするために、定期的に変更されます。
IRK	Identity Resolving Key。RPA の生成に使用されます。RPA を使用するデバイスはペアリング時に対向デバイスに IRK を配布することで、対向デバイスはその IRK を使用してデバイスを特定することが可能です。
LTK	Long Term Key。暗号化に使用される秘密鍵です。ペアリング時に Slave から配布された LTK が暗号化に使用されます。
GAP	Generic Access Profile。2 つのデバイスが互いを発見し、接続を行うための機能を定義するプロファイルです。
GATT	Generic Attribute Profile。サービス（データベース）とサービスにアクセスするための手順を定義するプロファイルです。サーバロールとクライアントロールがあり、サーバロールがサービスを提供し、クライアントロールがサービスの検索やサービスへの書き込み要求などのアクセスを行います。

1. 概要

Security Library は、BLE プロトコルスタックが提供するセキュリティ機能を容易に使用するための API を提供します。Security Library は、BLE プロトコルスタック V1.20 以降と組み合わせて使用します。Central / Peripheral のどちらからでも使用可能です。また、Embedded 構成 / Modem 構成のどちらからでも使用可能です。

Security Library の以下の特徴により rBLE API を使用するよりも容易にセキュリティ機能を使用することができます。

- パスキー / LTK / IRK を自動的に生成
- セキュリティ情報の管理（保存・削除）を自動的に実施
- プライバシーに関連する処理を自動的に実施
- 対向デバイスとのセキュリティ状態やサービス要求時のエラー内容に応じて適切な処理を実施

Security Library が提供するセキュリティ機能の概要を表 1-1 に示します。

表 1-1 セキュリティ機能

Function	Description
ペアリング	ペアリングは、暗号化やプライバシーに使用する鍵などのセキュリティ情報を交換するために実行されます。ペアリング完了後、通信は暗号化されます。ペアリングには、以下の 2 種類があります。 • MITM から保護されない Unauthenticated Pairing • MITM から保護される Authenticated Pairing
暗号化	通信を暗号化します。これにより、不正なデバイスが BLE 上の通信を盗聴することを防ぎます。暗号化の実施前に、ペアリングによりセキュリティ情報を交換しておく必要があります。
ボンディング	ペアリング時に交換したセキュリティ情報を保存する機能です。セキュリティ情報を保存し、以降の暗号化やプライバシーを実施する際に使用します。
プライバシー	RPA を使用することで、悪意のあるデバイスからの追跡を困難にする機能です。 RPA を使用しているデバイスであっても、ペアリング時に対向デバイスとセキュリティ情報を交換することで、対向デバイスはそのデバイスを特定することができます。

本書の構成を以下に示します。

2 章は、Security Library の使用方法について記載しています。Security Library を使用する前に通読してください。3 章は、API のインタフェース仕様について記載しています。4 章は、プロジェクトに Security Library を組み込む際の手順について記載しています。5 章は、API の動作シーケンスについて記載しています。Security Library の内部動作を把握したい場合に参照してください。

2. Security Library の使用方法

本章は、Security Library の使用方法について記載します。Security Library の API には、実行が必須な API と、必要に応じて実行する API があります。各 API の使用方法は、下記の表に示した章を参照してください。

表 2-1 は、Security Library を使用する際に実行が必須な API です。

表 2-1 実行が必須な API

処理内容	API		参照
Security Library の初期化	関数	SecLib_Init	2.1
	イベント	SECLIB_EVENT_INIT_COMP	
セキュリティパラメータの設定	関数	SecLib_Set_Param	2.2
	イベント	SECLIB_EVENT_SET_PARAM_COMP	

表 2-2 に必要に応じて実行する API のリストを示します。

表 2-2 必要に応じて実行する API

処理内容	API		参照
ペアリング/暗号化の実施	関数	SecLib_Start_Encryption SecLib_Pairing_Req_Resp SecLib_Passkey_Req_Resp	2.3
	イベント	SECLIB_EVENT_PAIRING_COMP SECLIB_EVENT_ENC_COMP SECLIB_EVENT_PAIRING_REQ SECLIB_EVENT_PASSKEY_IND SECLIB_EVENT_PASSKEY_REQ	
サービス要求エラーへの対応	関数	SecLib_SrvcReq_Error_Resp	2.4
	イベント	-	
ボンディング情報の管理	関数	SecLib_Delete_Bonding_Info	2.5
	イベント	SECLIB_EVENT_DELETE_BONDING_INFO_COMP	

Security Library の API は、rBLE API と同様の動作を行います。関数を実行後、実行結果がイベントとしてコールバック関数に通知されます。関数実行後は、実行結果がコールバック関数に通知されるまで、他の関数は実行しないでください。

Security Library の API は、rBLE API を使用して実装されています。Security Library の各関数は、複数の rBLE 関数を実行して、指定されたセキュリティ機能を実現しています。また、Security Library 内に rBLE イベントを処理するコールバック関数を持ち、セキュリティ関連のイベントが発生した場合はそのコールバック関数で処理を行い、セキュリティ関連以外のイベントが発生した場合はアプリケーションにそのイベントを通知します。

2.1 Security Library の初期化

Security Library を使用するためには、まず Security Library の初期化を行う必要があります。初期化は、SecLib_Init 関数により行います。実行結果は、SECLIB_EVENT_INIT_COMP により通知されます。

通常、RL78/G1D の初期化は、RBLE_GAP_Reset 関数により行いますが、Security Library を使用する場合は、代わりに SecLib_Init 関数を使用してください。RBLE_GAP_Reset 関数は、SecLib_Init 関数内部で呼び出されます。

図 2-1 に SecLib_Init 関数の使用例を示します。SecLib_Init 関数は、rBLE Mode が RBLE_MODE_ACTIVE に遷移した後に実行する必要があります。

```
1.  static void app_seclib_callback(SECLIB_EVENT *event)
2.  {
3.      switch (event->type) {
4.  /* ... skip ... */
5.      case SECLIB_EVENT_INIT_COMP:
6.          if (event->param.status == RBLE_OK) {
7.              /* SecLib_Init has finshed with OK. */
8.          }
9.          else {
10.             /* SecLib_Init has finshed with Error. */
11.         }
12.         break;
13. /* ... skip ... */
14.     }
15. }
16.
17. static void app_callabck(RBLE_MODE mode)
18. {
19.     switch (mode) {
20. /* ... skip ... */
21.     case RBLE_MODE_ACTIVE:
22.         SecLib_Init(&app_gap_callback, NULL,  &app_seclib_callback);
23.         break;
24. /* ... skip ... */
25.     }
26. }
27.
28. BOOL RBLE_App_Init(void)
29. {
30. /* ... skip ... */
31.     RBLE_Init(&app_callback);
32. /* ... skip ... */
33. }
```

図 2-1 SecLib_Init の使用例

2.2 セキュリティパラメータの設定

Security Library の初期化完了後、ペアリングやプライバシーの実施に使用するセキュリティパラメータの設定を行います。セキュリティパラメータの設定は、SecLib_Set_Param 関数により実行します。実行結果は、SECLIB_EVENT_SET_PARAM_COMP により通知されます。

SecLib_Set_Param 関数は、一度設定したセキュリティパラメータを変更する際にも使用できます。

図 2-2 に SecLib_Set_Param 関数の使用例を示します。SecLib_Set_Param 関数は、SECLIB_EVENT_INIT_COMP 発生以降に実行する必要があります。

```
1.  static SECLIB_PARAM app_sec_param = {
2.      RBLE_MASTER,                /* role */
3.      RBLE_AUTH_REQ_MITM_BOND,    /* auth_req */
4.      RBLE_IO_CAP_KB_ONLY,        /* iocap */
5.      TRUE,                       /* rpa_generate */
6.  };
7.
8.  static void app_seclib_callback(SECLIB_EVENT *event)
9.  {
10.     switch (event->type) {
11. /* ... skip ... */
12.     case SECLIB_EVENT_INIT_COMP:
13.         if (event->param.status == RBLE_OK) {
14.             SecLib_Set_Param(&app_sec_param);
15.         }
16.         else {
17.             /* SecLib_Init has finished with Error. */
18.         }
19.         break;
20.
21.     case SECLIB_EVENT_SET_PARAM_COMP:
22.         if (event->param.status == RBLE_OK) {
23.             /* SecLib_Set_Param has finished with OK. */
24.         }
25.         else {
26.             /* SecLib_Set_Param has finished with Error. */
27.         }
28.         break;
29. /* ... skip ... */
30.     }
31. }
```

図 2-2 SecLib_Set_Param の使用例

表 2-3 にセキュリティパラメータのリストを示します。セキュリティパラメータは、SECLIB_PARAM 構造体により表されます。

表 2-3 セキュリティパラメータ

Parameters	Description	Refer
role	ロールの設定	2.2.1
auth_req	Authentication Requirement の設定	2.2.2
iocap	IO Capabilities の設定	2.2.3
rpa_generate	プライバシーの設定	2.2.4

2.2.1 ロールの設定

role には、デバイスのロールに応じて表 2-4 のいずれかの値を設定して下さい。

表 2-4 ロール設定

Settings	Description
RBLE_MASTER	デバイスは Central として動作
RBLE_SLAVE	デバイスは Peripheral として動作

2.2.2 Authentication Requirement の決定

auth_req には、デバイスが必要とするセキュリティの強度に応じて、表 2-5 に示すいずれかの値を設定してください。本設定は、2.3.1 節で使用されます。

表 2-5 Authenticate Requirement

Settings	Description
RBLE_AUTH_MITM_BOND	Authenticated Pairing を要求 （MITM からの保護を含む Passkey Entry によるペアリング）
RBLE_AUTH_NO_MITM_BOND	Unauthenticated Pairing を要求 （MITM からの保護を含まない Just Works によるペアリング）

2.2.3 IO Capabilities の設定

iocap には、デバイスが保持する入出力装置に応じて、表 2-6 に示すいずれかの値を設定してください。本設定は、2.3.1 節で使用されます。

表 2-6 IO Capabilities

Settings	Description
RBLE_IO_CAP_DISPLAY_ONLY	6 桁数値を表示できる出力装置（ディスプレイなど）
RBLE_IO_CAP_DISPLAY_YES_NO	6 桁数値を表示できる出力装置（ディスプレイなど）および Yes/No を指示できる入力装置（ボタンなど）
RBLE_IO_CAP_KB_ONLY	0~9 の数値を入力できる入力装置（キーボードなど）
RBLE_IO_CAP_NO_INPUT_NO_OUTPUT	入出力装置を持たない
RBLE_IO_CAP_KB_DISPLAY	0~9 の数値を入力できる入力装置（キーボードなど）および 6 桁数値を表示できる出力装置（ディスプレイなど）

2.2.4 プライバシーの設定

rpa_generate には、デバイスのプライバシー機能への要求に従い、表 2-7 に示すいずれかの値を設定してください。本設定を TRUE に設定すると、デバイスアドレスに RPA が使用され、デバイスアドレスが定期的に変更されるようになります。変更間隔は、GAP_RESOLVABLE_PRIVATE_ADDR_INTV に従います。

表 2-7 プライバシー機能の設定

Settings	Description
TRUE	RPA を使用して、デバイスアドレスを定期的に変更する
FALSE	デバイスアドレスを変更しない

プライバシー機能を有効化した RL78/G1D を Peripheral と、iOS 端末を Central としてペアリングを実施すると、ペアリングに失敗します。iOS 端末と接続する Peripheral は、本機能を無効化してください。

2.3 ペアリング/暗号化の実施

ペアリング/暗号化の実施は、SecLib_Start_Encryption 関数により行います。SecLib_Start_Encryption 関数は、対向デバイスとのセキュリティ状態に応じて、以下のように動作します。

- 対向デバイスとペアリングが完了していない場合、ペアリングを実施します。ペアリングの完了は、SECLIB_EVENT_PAIRING_COMP により通知されます。ペアリング完了後、通信は暗号化されています。
- 対向デバイスとのペアリングが完了している場合、ペアリング時に決定した鍵を使用して、暗号化を行います。暗号化の完了は、SECLIB_EVENT_ENC_COMP により通知されます。

対向デバイスとのペアリングが完了・未完了の確認は、接続後 Security Library が実施します。確認結果は、SECLIB_EVENT_CHK_ADDR_COMP により通知されます。

ペアリングおよび暗号化のいずれを実施した場合であっても、完了後は通信が暗号化された状態になります。ただし、ペアリング完了時に実施されている暗号化は、ペアリング時に決定した鍵を使用した暗号化ではなく、一時鍵を使用した暗号化です。ペアリング時に決定した鍵を使用した暗号化を実施するためには、ペアリング完了後、再度 SecLib_Start_Encryption 関数を実行する必要があります。

2.3.1 ペアリングの実施

対向デバイスとのペアリングが完了していない状態で、SecLib_Start_Encryption 関数を実行した場合、ペアリングが実施されます。ペアリングの動作シーケンスを図 2-3 に示します。ペアリングは、以下の処理を行います。

- ペアリング要求・応答（図 2-3-A）
- Passkey Entry / Just Works によるペアリングの実施（図 2-3-B）
- セキュリティ情報の交換・保存（図 2-3-C）
- ペアリングの完了通知（図 2-3-D）

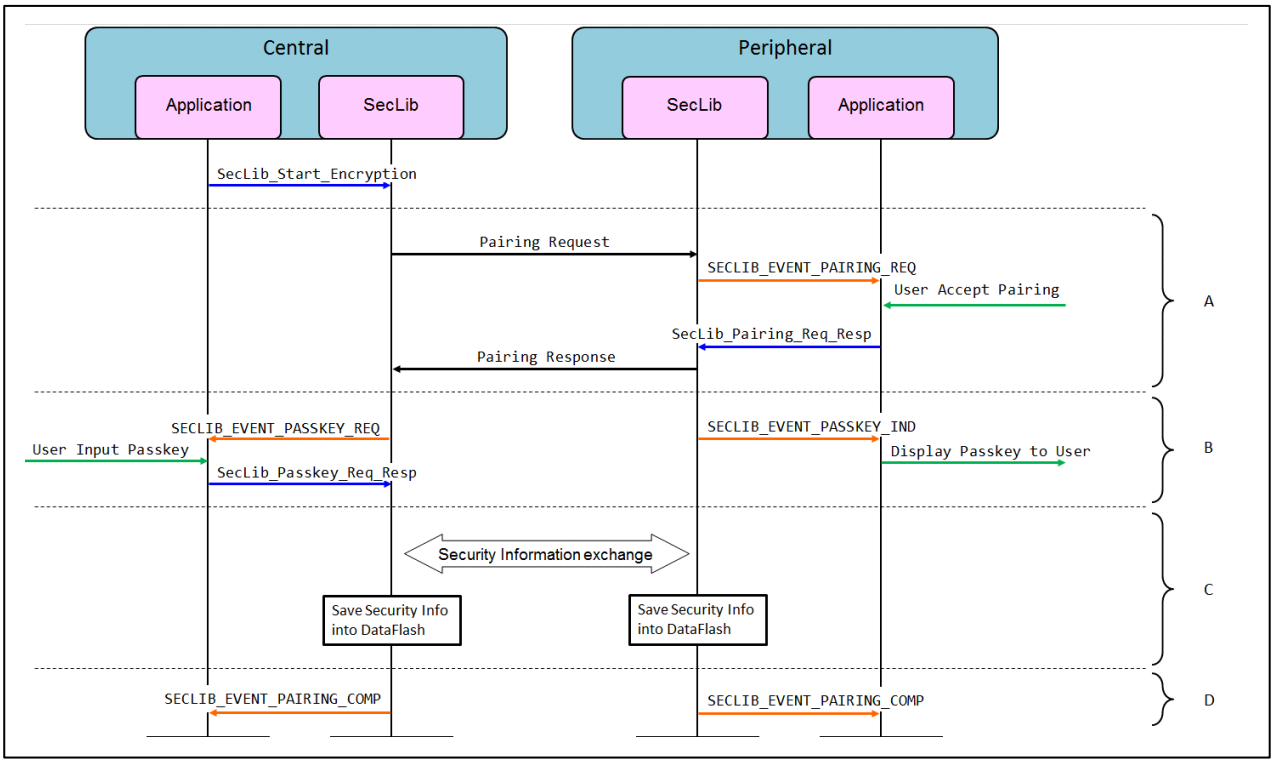


図 2-3 ペアリングの動作シーケンス

図 2-3 は、ペアリングの動作シーケンスの一例です。セキュリティパラメータの設定により、シーケンスは変化します。

(1) ペアリング要求・応答

ペアリングは、ペアリング要求を送信することから始まります。ペアリング要求を受け取ったデバイスは、ペアリングを受諾するか・拒否するかを応答します。

ペアリング要求

対向デバイスとのペアリングが未完了の場合、SecLib_Start_Encryption 関数を実行すると、ペアリング要求が送信されます。

ペアリング応答

ペアリング要求の受信は、SECLIB_EVENT_PAIRING_REQ として通知されます。アプリケーションはペアリングの受諾・拒否のいずれかを SecLib_Pairing_Req_Resp 関数により応答します。ペアリングの受諾・拒否は、必要に応じてユーザに提示して判断させます。

図 2-4 に SecLib_Pairing_Req_Resp 関数の使用例を示します。例では、SECLIB_EVENT_PAIRING_REQ 発生時にユーザに確認を促し、ユーザの指示に従って受諾・拒否の応答をしています。

```
1.  static uint16_t conhdl;
2.
3.  /* This is pseudo function which receives user input (yes/no). */
4.  static void app_user_input_yes_no(BOOL yes_no)
5.  {
6.      SecLib_Pairing_Req_Resp(conhdl, yes_no);
7.  }
8.
9.  static void app_seclib_callback(SECLIB_EVENT *event)
10. {
11.     switch (event->type) {
12. /* ... skip ... */
13.     case SECLIB_EVENT_PAIRING_REQ:
14.         /* Confirm to user whether accept the pairing request or not. */
15.         Printf("Accept pairing request?: (yes/no)¥n");
16.         /* Save conhdl to use for SecLib_Pairing_Req_Resp argument. */
17.         conhdl = event->param.pairing_req.conhdl;
18.         break;
19. /* ... skip ... */
20.     }
21. }
```

図 2-4 SecLib_Pairing_Req_Resp の使用例

(2) Passkey Entry / Just Works によるペアリングの実施

Security Library は、ペアリング方法として Passkey Entry と Just Works の 2 つをサポートしています（OOB には対応していません）。どちらが使用されるのかは、自デバイスおよび対向デバイスの Authentication Requirement / IO Capabilities の組み合わせによって決定されます。表 2-8 および表 2-9 は、Central および Peripheral の Authentication Requirement / IO Capabilities の組み合わせに応じた決定方法を示します。

表 2-8 Authentication Requirement による決定

Peripheral ＼ Central	MITM_BOND	NO_MITM_BOND
MITM_BOND	表 2-9 参照	表 2-9 参照
NO_MITM_BOND	表 2-9 参照	Unauthenticated *Just Works*

表 2-9 IO Capabilities による決定

Central ＼ Peripheral	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_ NO_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 の場合

アプリケーションが行う処理はありません（図 2-3-B の処理は実施されません）。

Passkey Entry の場合

アプリケーションは、SecLib_Set_Param 関数で設定した IO Capabilities の設定にもとづき、ユーザへのパスキーの表示や、ユーザからパスキーの入力を受け付ける処理を実装する必要があります。

- パスキーを表示する側のデバイスには SECLIB_EVENT_PASSKEY_IND が通知されます。出力装置に通知されたパスキーを表示し、ユーザに提示します。
  図 2-4 に SECLIB_EVENT_PASSKEY_IND の使用例を示します。

```
1.  static void app_seclib_callback(SECLIB_EVENT *event)
2.  {
3.      switch (event->type) {
4.  /* ... skip ... */
5.      case SECLIB_EVENT_PASSKEY_IND:
6.          /* Display passkey to user. */
7.          Printf("Passkey: %06ld¥n", event->param.passkey_ind.passkey);
8.          break;
9.  /* ... skip ... */
10.     }
11. }
```

図 2-5 SECLIB_EVENT_PASSKEY_IND の使用例

- パスキーを入力する側のデバイスには SECLIB_EVENT_PASSKEY_REQ が通知されます。ユーザにパスキーの入力を要求します。ユーザが入力したパスキーは、SecLib_Passkey_Req_Resp 関数により Security Library に引き渡します。
  図 2-6 に SecLib_Passkey_Req_Resp 関数の使用例を示します。

```
1.  static uint16_t conhdl;
2.
3.  /* This is pseudo function which receives user input (passkey). */
4.  static void app_user_input_passkey(uint32_t passkey)
5.  {
6.      SecLib_Passkey_Req_Resp(conhdl, passkey);
7.  }
8.
9.  static void app_seclib_callback(SECLIB_EVENT *event)
10. {
11.     switch (event->type) {
12. /* ... skip ... */
13.     case SECLIB_EVENT_PASSKEY_REQ:
14.         /* Request user to input passkey displayed on Peer Device. */
15.         Printf("Input passkey:¥n");
16.         /* Save conhdl to use for SecLib_Passkey_Req_Resp argument. */
17.         conhdl = event->param.passkey_req.conhdl;
18.         break;
19. /* ... skip ... */
20.     }
21. }
```

図 2-6 SecLib_Passkey_Req_Resp の使用例

(3) セキュリティ情報の交換・保存

暗号化やプライバシーを有効化する際に必要となるセキュリティ情報を交換します。セキュリティ情報は、ボンディングにより RL78/G1D の内蔵データフラッシュに保存されます。以降は、保存した情報を使用して、暗号化やプライバシーの有効化を実施できます。

(4) ペアリング完了通知

ペアリングが完了すると、SECLIB_EVENT_PAIRING_COMP が通知されます。図 2-7 に SECLIB_EVENT_PAIRING_REQ の使用例を示します。

```
1.  static void app_seclib_callback(SECLIB_EVENT *event)
2.  {
3.      switch (event->type) {
4.  /* ... */
5.      case SECLIB_EVENT_PAIRING_COMP:
6.          if (event->param.pairing_comp.status == RBLE_OK) {
7.              /* Pairing has finished with OK. */
8.          }
9.          else {
10.             /* Pairing has finished with Error. */
11.         }
12.         break;
13. /* ... */
14.     }
15. }
```

図 2-7 SECLIB_EVENT_PAIRING_COMP の使用例

(5) ペアリングの失敗

ペアリングは以下のような場合に失敗します。ペアリングが失敗した場合、再度ペアリングを実施するためには、切断・再接続した後、ペアリングを開始する必要があります。

- ペアリング応答が拒否の場合、SECLIB_EVENT_PAIRING_COMP の status が RBLE_SM_PAIR_ERR_PAIRING_NOT_SUPPORTED として通知されます。
- ペアリングが 30 秒以内に完了しない場合、SECLIB_EVENT_PAIRING_COMP の status が RBLE_ERR として通知されます。
- 誤ったパスキーを入力した場合、SECLIB_EVENT_PAIRING_COMP の status が RBLE_SM_PAIR_ERR_CFM_VAL_FAILED として通知されます。
- Central / Peripheral の片方もしくは両方のデバイスが Authenticated Pairing（Passkey Entry）を要求しているが、IO Capabilities の組み合わせによって、Unauthenticated Pairing（Just Works）が選択された場合、そのペアリングは失敗し、SECLIB_EVENT_PAIRING_COMP の status が RBLE_SM_PAIR_ERR_AUTH_REQUIREMENTS として通知されます。

2.3.2 暗号化の実施

対向デバイスとのペアリングが完了している状態で、SecLib_Start_Encryption 関数を実行した場合、暗号化が実施されます。暗号化の動作シーケンスを図 2-8 に示します。暗号化は、以下の処理を行います。

- 暗号化の開始（図 2-8-A）
- 管理データの保存（図 2-8-B）
- 暗号化の完了通知（図 2-8-C）

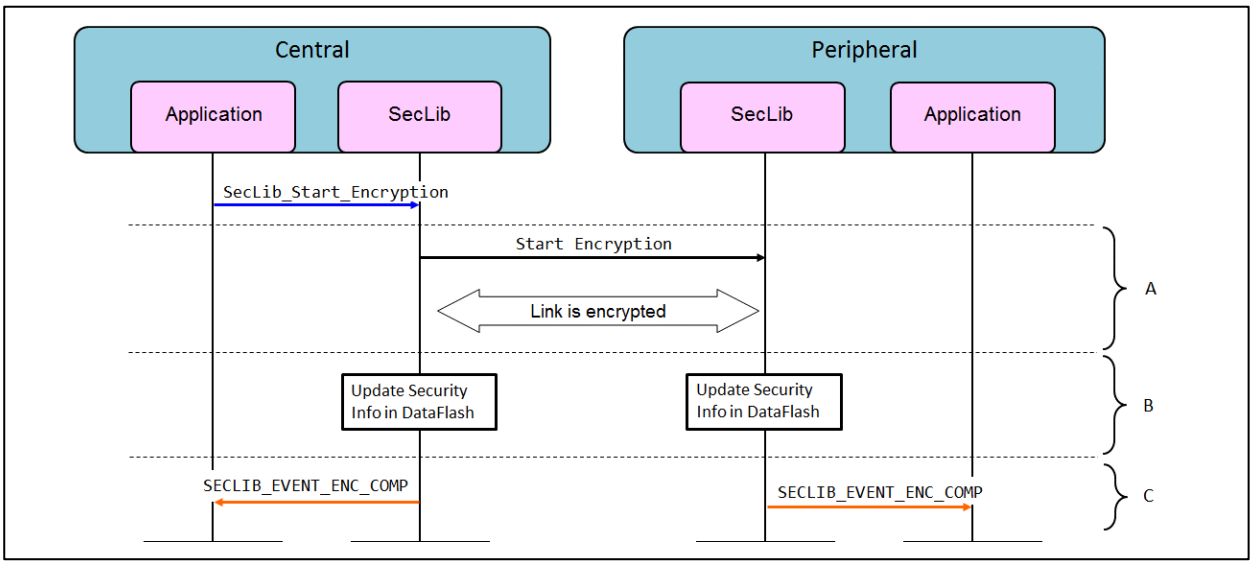


図 2-8 暗号化の動作シーケンス

(1) 暗号化の開始

暗号化は、ペアリング時に決定した鍵を使用して実施します。

(2) 管理データの保存

2.5.1 項に示す LRU に使用する情報を更新します。

(3) 暗号化完了通知

暗号化が完了すると、SECLIB_EVENT_ENC_COMP が通知されます。図 2-9 に SECLIB_EVENT_ENC_COMP の使用例を示します。暗号化の実行結果を取得することができます。

```
1.  static void app_seclib_callback(SECLIB_EVENT *event)
2.  {
3.      switch (event->type) {
4.  /* ... */
5.      case SECLIB_EVENT_ENC_COMP:
6.          if (event->param.enc_comp.status == RBLE_OK) {
7.              /* Encryption has finished with OK. */
8.          }
9.          else {
10.             /* Encryption has finished with Error. */
11.         }
12.         break;
13. /* ... */
14.     }
15. }
```

図 2-9 SECLIB_EVENT_ENC_COMP の使用例

2.4 サービス要求エラーへの対応

対向デバイスに対して、GATT 書き込み・読み込み要求などのサービス要求を実行した際に、サービス要求先の特性値のパーミッション設定や、対向デバイスとのセキュリティ状態によって、サービス要求エラーが発生する場合があります。

SecLib_SrvcReq_Error_Resp 関数は、表 2-10 に示すように、サービス要求エラーが発生した際に、セキュリティ状態に応じて、ペアリングや暗号化を開始し、2.3.1 項や 2.3.2 項に示す処理が実行されます。また、サービス要求エラーに対応できない状態の場合は、エラーを返却します。以下に、サンプルプログラムに同梱されいる Security Library の各ペアリング/暗号化状態における処理内容を示します。

表 2-10 サービス要求エラーとセキュリティ状態に応じた処理

サービス要求エラー	ペアリング	暗号化	処理（アプリケーションが実施する処理）
Insufficient Authentication	None	Disabled	ペアリングを開始 （図 2-3 の A~D を実施）
	None	Enabled	RBLE_STATUS_ERROR を返却 （通常発生しない）
	Unauthenticated	Disable	暗号化を開始、失敗した場合ペアリングを開始[Note 1] （図 2-8 の A~C を実施、暗号化に失敗した場合、図 2-8 の C は通知されず、図 2-3 の A~D を実施）
	Unauthenticated	Enabled	MITM 保護つきのペアリングを開始 （図 2-3 の A~D を実施）
	Authenticated	Disabled	暗号化を開始、失敗した場合ペアリングを開始[Note 1] （図 2-8 の A~C を実施、暗号化に失敗した場合、図 2-8 の C は通知されず、図 2-3 の A~D を実施）
	Authenticated	Enabled	RBLE_ERR を返却 （RL78/G1D がサポートしていない LE Secure Connection を対向デバイスが要求した場合）
Insufficient Encryption	None	Disabled	ペアリングを開始 （図 2-3 の A~D を実施）
	None	Enabled	RBLE_STATUS_ERROR を返却 （通常発生しない）
	Unauthenticated	Disable	暗号化を開始、失敗した場合ペアリングを開始[Note 1] （図 2-8 の A~C を実施、暗号化に失敗した場合、図 2-8 の C は通知されず、図 2-3 の A~D を実施）
	Unauthenticated	Enabled	RBLE_STATUS_ERROR を返却 （通常発生しない）
	Authenticated	Disabled	暗号化を開始、失敗した場合ペアリングを開始[Note 1] （図 2-8 の A~C を実施、暗号化に失敗した場合、図 2-8 の C は通知されず、図 2-3 の A~D を実施）
	Authenticated	Enabled	RBLE_STATUS_ERROR を返却 （通常発生しない）

Note 1: ペアリング済みのデバイスと同じ BD アドレスをもつ悪意あるデバイスと接続した場合には、その悪意あるデバイスが暗号鍵を持たないため暗号化が失敗します。同梱している Security Library は暗号化失敗時に自動的にペアリングを開始し、悪意あるデバイスとのペアリングが成立する可能性があります。この問題を回避するために、6.2 節にしたがって暗号化失敗時に自動的にペアリングを開始しないように対策することを推奨します。

図 2-10 に SecLib_SrvcReq_Error_Resp 関数の使用例を示します。引数には、GATT 書き込み要求に対する応答を渡しています。

```
1.  static void app_gatt_callback(RBLE_GATT_EVENT *event)
2.  {
3.      switch (event->type) {
4.  /* ... skip ... */
5.      case RBLE_GATT_EVENT_WRITE_CHAR_RESP:
6.          SecLib_SrvcReq_Error_Resp(conhdl, event->param.write_char_resp.att_code);
7.          break;
8.  /* ... skip ... */
9.      }
10. }
```

図 2-10 SecLib_SrvcReq_Error_Resp の使用例

2.5 セキュリティ情報の管理

セキュリティ情報は、ペアリング/暗号化を実施時に RL78/G1D の内蔵データフラッシュに保存されます。また、アプリケーションの指示に応じて、セキュリティ情報を削除することができます。

2.5.1 セキュリティ情報の保存

セキュリティ情報の保存は、ペアリング/暗号化完了時に Security Library が行います。表 2-11 にセキュリティ情報として保存する項目を示します。

CFG_SECLIB_BOND_NUM により設定したボンディング情報の個数を超えてボンディング情報を保存しようとした場合、LRU（Least Recently Used）にもとづいて最も長い期間使用されていないボンディング情報が、新たなボンディング情報により上書きされます。LRU 実施のため、ペアリング/暗号化完了時に、各ボンディング情報をアクセスした順序を保存します。

表 2-11 セキュリティ情報

Data	Description
自デバイス IRK	自デバイスが生成した IRK
管理データ	LRU 実施に使用する各ボンディング情報のアクセス順序の記録
ボンディング情報	対向デバイス毎に保持する情報（表 2-12 参照） （CFG_SECLIB_BOND_NUM 個分）

表 2-12 ボンディング情報

Data	Description
セキュリティプロパティ	対向デバイスとのセキュリティレベル
鍵長	対向デバイスから受け取った LTK のサイズ
対向デバイスアドレス	対向デバイスのデバイス アドレス
対向デバイスアドレスタイプ	対向デバイスのデバイス アドレス Type
対向デバイス IRK	対向デバイスから受け取った IRK
対向デバイス LTK	対向デバイスから受け取った LTK
自デバイス LTK	自デバイスが生成した LTK

2.5.2 セキュリティ情報の削除

以下の場合に、セキュリティ情報やボンディング情報が削除されます。

- アプリケーションが SecLib_Delete_Bonding_Info 関数を実行した場合、指定したボンディング情報が削除されます。本関数の実行結果は、SECLIB_EVENT_DELETE_BONDING_INFO_COMP により通知されます。
- CFG_SECLIB_BOND_NUM により設定したボンディング情報の個数を超えてボンディング情報を保存しようとした場合、古いボンディング情報が新しいボンディング情報により上書きされます。
- 暗号化に失敗した場合、暗号化に失敗したボンディング情報は削除[Note 1]されます。暗号化に失敗した際には、「Bluetooth Core Specification 4.2 Vol 3 10.6 Encryption Procedure」の記載に従い、アプリケーションは、SecLib_Set_Param 関数の”rpa_generate”に TRUE を設定して実行し、RPA を変更する必要があります。
  Note 1: 6.2 節に従った場合は Security Library はボンディング情報を削除しません。
- セキュリティ情報に含まれる自デバイスの IRK は、すべてのボンディング情報が削除された場合にのみ削除されます。

3. インタフェース定義

3.1 関数

3.1.1 SecLib_Init

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

- Security Library の初期化を行います。
- 本関数の実行完了は、SECLIB_EVENT_INIT_COMP により通知されます。
- 本関数は rBLE モードが RBLE_MODE_ACTIVE になった後に実行してください。
- 本関数を RBLE_GAP_Reset 関数の代わりに実行してください。RBLE_GAP_Reset 関数は、本関数内で呼び出すため、アプリケーションで使用しないでください。
- RBLE_VS_Enable 関数は、本関数内で呼び出すためアプリケーションから使用しないでください。また、Vendor Specific 関数の実行は、SECLIB_EVENT_INIT_COMP が通知された後に行ってください。

Parameter:	
gap_callback	GAP を通知するコールバック関数
vs_callback	Vendor Specific を通知するコールバック関数
seclib_callback	Security Library を通知するコールバック関数
Return:	
RBLE_OK	正常終了
RBLE_PARAM_ERR	gap_callback または seclib_callback が NULL

3.1.2 SecLib_Set_Param

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

- セキュリティパラメータの設定を行います。
- 本関数の実行結果は、SECLIB_EVENT_SET_PARAM_COMP により通知されます。
- 本関数は、SECLIB_EVENT_INIT_COMP 以降に実行してください。
- 本関数は、セキュリティパラメータの変更時にも使用することができます。
- 本関数は、対向デバイスとの接続中は実行できません。

Parameter:	
param	セキュリティパラメータ
Return:	
RBLE_OK	正常終了
RBLE_PARAM_ERR	param が NULL
RBLE_STATUS_ERROR	SECLIB_EVENT_INIT_COMP 前に本関数を実行 対向デバイスと接続中

3.1.3 SecLib_Start_Encryption

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

- 対向デバイスとのペアリングが未完了の場合は、ペアリングを行います。この場合、実行結果は、SECLIB_EVENT_PAIRING_COMP により通知されます。
- ペアリングが完了している場合は、ペアリング時に決定した鍵を使用して暗号化を行います。この場合、実行結果は SECLIB_EVENT_ENC_COMP により通知されます。

Parameter:	
conhdl	コネクションハンドル
Return:	
RBLE_OK	正常終了
RBLE_PARAM_ERR	conhdl が不正
RBLE_STATUS_ERROR	SECLIB_EVENT_SET_PARAM_COMP 前に本関数を実行 SECLIB_EVENT_CHK_ADDR_COMP 前に本関数を実行

3.1.4 SecLib_Pairing_Req_Resp

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

- 対向デバイスからのペアリング要求に応答します。
- 本関数は、SECLIB_EVENT_PAIRING_REQ 発生時にのみ実行してください。それ以外の場面では使用しないでください。

Parameter:	
conhdl	コネクションハンドル
accept	ペアリング要求を受諾（TRUE）もしくは拒否（FALSE）
Return:	
RBLE_OK	正常終了
RBLE_PARAM_ERR	conhdl が不正

3.1.5 SecLib_Passkey_Req_Resp

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

- Passkey Entry によるペアリングを実施時に、パスキーの入力要求に応答します。
- 本関数は、SECLIB_EVENT_PASSKEY_REQ 発生時にのみ実行してください。それ以外の場面では使用しないでください。

Parameter:	
conhdl	コネクションハンドル
passkey	パスキー
Return:	
RBLE_OK	正常終了
RBLE_PARAM_ERR	conhdl が不正

3.1.6 SecLib_SrvcReq_Error_Resp

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

- サービス要求エラーへの応答として、ペアリング/暗号化などを開始します。
- 本関数で応答可能なエラーは、Insufficient Authentication、Insufficient Encryption です。att_err に発生したエラーを指定します。

Parameter:

conhdl	コネクションハンドル	
att_err	RBLE_ATT_ERR_INSUFF_AUTHEN	Insufficient Authentication の処理を行います。
	RBLE_ATT_ERR_INSUFF_ENC	Insufficient Encryption の処理を行います。
	RBLE_ATT_ERR_NO_ERROR	何もしません。

Return:

RBLE_OK	正常終了
RBLE_PARAM_ERR	att_err が不正 conhdl が不正
RBLE_STATUS_ERROR	SECLIB_EVENT_SET_PARAM_COMP 前に本関数を実行 SECLIB_EVENT_CHK_ADDR_COMP 前に本関数を実行 不正なペアリング/暗号化状態
RBLE_ERR	対向デバイスが Authenticated Pairing を要求しているが、自デバイスの IO Capabilities が RBLE_IO_CAP_NO_INPUT_NO_OUTPUT の場合 Authenticated Pairing を実施している状態で Insufficient Authentication が発生した場合（対向デバイスが LE Secure Connection を要求している場合）

3.1.7 SecLib_Delete_Bonding_Info

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

- ボンディング情報を RL78/G1D の内蔵データフラッシュから削除します。
- 実行結果は、SECLIB_EVENT_DELETE_BONDING_INFO_COMP により通知されます。

Parameter:

target	SECLIB_DELETE_ALL_BONDS	全ボンディング情報を削除
	SECLIB_DELETE_ALL_BUT_ACTIVE_BOND	接続中の対向デバイスとのボンディング情報以外を削除

Return:

RBLE_OK	正常終了
RBLE_PARAM_ERR	target が不正
RBLE_STATUS_ERROR	SECLIB_EVENT_INIT_COMP 前に本関数を実行
RBLE_BUSY	RL78/G1D の内蔵データフラッシュへのアクセスに失敗

3.1.8 SecLib_Rand

```
uint16_t SecLib_Rand(void)
```

- 16-bits の擬似乱数を生成する関数です。
- Security Library がパスキー / LTK / IRK を生成する際に使用します。
- 本関数の定義は、アプリケーションが行う必要があります。

3.2 イベント

3.2.1 SECLIB_EVENT_INIT_COMP

SECLIB_EVENT_INIT_COMP

RBLE_STATUS status

SecLib_Init 関数の実行結果です。

Value	Description
RBLE_OK	実行成功
RBLE_ERR	実行失敗

3.2.2 SECLIB_EVENT_SET_PARAM_COMP

SECLIB_EVENTSET_PARAM_COMP

RBLE_STATUS status

SecLib_Set_Param 関数の実行結果です。

Value	Description
RBLE_OK	実行成功
RBLE_ERR	実行失敗

3.2.3 SECLIB_EVENT_PAIRING_COMP

SECLIB_EVENT_PAIRING_COMP

RBLE_STATUS status

ペアリングの実行結果です。

値	説明
RBLE_OK	実行成功
RBLE_ERR	セキュリティ情報の保存に失敗 30 秒以内にペアリングが完了しない場合
RBLE_SM_PAIR_ERR_CFM_VAL_FAILED	誤ったパスキーを入力
RBLE_SM_PAIR_ERR_AUTH_REQUIREMENTS	認証条件が満たされなかった場合
RBLE_SM_PAIR_ERR_PAIRING_NOT_SUPPORTED	ペアリング要求が拒否された場合

uint16_t conhdl

コネクションハンドル

uint16_t sec_prop

完了したペアリングのセキュリティプロパティ

値	説明
RBLE_SMP_SEC_NONE	ペアリング失敗
RBLE_SMP_UNAUTHENTICATED	Unauthenticated Pairing を実施
RBLE_SMP_AUTHENTICATED	Authenticated Pairing を実施

3.2.4 SECLIB_EVENT_ENC_COMP

SECLIB_EVENT_ENC_COMP

RBLE_STATUS status

暗号化の実行結果です。

値	説明
RBLE_OK	実行成功
RBLE_ERR	実行失敗

uint16_t conhdl

コネクションハンドル

uint16_t sec_prop

完了済のペアリングのセキュリティプロパティ

値	説明
RBLE_SMP_SEC_NONE	暗号化失敗
RBLE_SMP_UNAUTHENTICATED	Unauthenticated Pairing を実施
RBLE_SMP_AUTHENTICATED	Authenticated Pairing を実施

3.2.5 SECLIB_EVENT_PAIRING_REQ

SECLIB_EVENT_SET_PARAM_COMP	
uint16_t conhdl	コネクションハンドル
uint8_t auth_req	対向デバイスが要求する Authentication Requirement
uint8_t iocap	対向デバイスの IO Capabilities 本値は、Central がペアリングを開始し、Peripheral が本イベントを受信した場合のみ有効です。

3.2.6 SECLIB_EVENT_PASSKEY_IND

SECLIB_EVENT_PASSKEY_IND	
uint16_t conhdl	コネクションハンドル
uint32_t passkey	パスキー

3.2.7 SECLIB_EVENT_PASSKEY_REQ

SECLIB_EVENT_PASSKEY_REQ	
uint16_t conhdl	コネクションハンドル

3.2.8 SECLIB_EVENT_CHK_ADDR_COMP

SECLIB_EVENTCHK_ADDR_COMP	
uint8_t status	対向デバイスとのペアリングの状態が完了か未完了かの確認結果です。

実行結果	説明
RBLE_OK	ペアリング済みの対向デバイス
RBLE_ERR	ペアリング未完了の対向デバイス

uint16_t conhdl	コネクションハンドル

3.2.9 SECLIB_EVENT_DELETE_BONDING_INFO_COMP

SECLIB_EVENT_DELETE_BONDING_INFO_COMP	
uint8_t status	SecLib_Delete_Bonding_Info 関数の実行結果です。

実行結果	説明
RBLE_OK	実行成功
RBLE_ERR	実行失敗

3.3 定義

3.3.1 SECLIB_EVENT_TYPE

```
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

```
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

定義すると Security Library のデバッグ情報を出力します。

3.3.7 CFG_SECLIB_BOND_NUM

RL78/G1D の内蔵データフラッシュに保存可能なボンディング情報の個数を定義します。CFG_CON 以上の値を設定してください。本設定は、開発環境のプロジェクト設定画面から行います。設定手順は、4.1.1 節を参照してください。

4. プロジェクトへの組み込み方法

4.1 手順

Security Library を既存プロジェクトに追加する手順を記載します。

4.1.1 プロジェクトの設定

(1) CFG_SECLIB_BOND_NUM マクロ定義の追加

RL78/G1D の内蔵データフラッシュに保存可能なボンディング情報の個数を設定します。CFG_SECLIB_BOND_NUM マクロ定義の追加は、開発環境の設定画面から行います。手順は、下記「インクルード・パスの追加」と共に記載します。

(1) Security Library のソースコードを追加

Security Library のソースコード（seclib.c / secdb.c）をプロジェクトに追加します。

(2) インクルード・パスの追加

Security Library のヘッダファイル（seclib.h / secdb.h）が含まれるディレクトリをインクルード・パスに追加します。インクルード・パスの追加は、開発環境の設定画面から行います。以下に各開発環境での手順を示します。

CS+の場合

プロジェクトツリー内「CA78K0R（ビルドツール）」もしくは「CC-RL（ビルドツール）」をダブルクリックします。設定画面が表示されるので、「共通オプション 」タブを選択します。タブ内の「定義マクロ」および「追加のインクルード・パス」を設定します。

e²studio の場合

プロジェクト・エクスプローラ内のプロジェクトを右クリックし、表示されたドロップダウンメニューから「Renesas Tool Settings」を選択します。表示されたウィンドウの「C/C++ビルド」→「設定」→「Compiler」→「ソース」を選択します。「定義マクロ」および「インクルード・ファイル・ディレクトリー」を設定します。

IAR Embedded Workbench の場合

Workspace 内のプロジェクトを右クリックし、表示されたドロップダウンメニューから「オプション(O)…」を選択します。表示されたウィンドウの「C/C++コンパイラ」→「プリプロセッサ」タブを選択します。「シンボル定義」および「追加のインクルードディレクトリ(A):1 行に 1 ディレクトリ」を設定します。

4.1.2 Security Library の呼び出し部の実装

(1) Security Manager 関連部分の削除

Security Library を使用している場合、以下の API は使用できません。既存のプロジェクト内で使用している場合は、該当部分を削除してください。

- すべての SM API（RBLE_SM_から始まる API）
- 以下に示す一部の rBLE GAP / 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 の実装

SecLib_Rand 関数を実装します。デバイスやアプリケーションが提供する乱数生成機能を使用して実装してください。Embedded 構成の場合、標準ライブラリが提供する rand 関数が使用できます。Modem 構成の場合は、ホスト側で乱数生成関数を準備する必要があります。

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

(3) Security Library のコールバック関数の実装

Security Library のイベントを通知するコールバック関数を実装します。Security Library の各イベント発生時に行う処理を実装してください。

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

(4) RBLE_GAP_Reset の置き換え

RBLE_GAP_Reset 関数の代わりに、SecLib_Init 関数を実行します。すでに RBLE_GAP_Reset 関数を使用している場合は、RBLE_GAP_Reset 関数呼び出し部分を削除し、SecLib_Init 関数に置き換えてください。また、実行完了時に発生するイベントも変更になるので注意してください。RBLE_GAP_Reset 関数の実行完了通知は、RBLE_GAP_EVENT_RESET_RESULT ですが、SecLib_Init 関数の完了通知は、SECLIB_EVENT_INIT_COMP です。

(5) その他の Security Library 関数の実行

2 章および 3 章を参照して実装してください。

4.1.3 Data Flash Library の設定

RL78/G1D の内蔵データフラッシュにセキュリティ情報を保存するための前準備として、データ構造の定義を追加します。本設定は、Modem 構成の場合は、RL78/G1D 側に設定してください。

(1) renesas/src/driver/dataflash/eel_descriptor_t02.h の編集方法

以下を参考に、編集してください。

- EEL_ID_MD、EEL_ID_LD_IRK、EEL_ID_BOND_1 は必須です。
- EEL_ID_BOND_*は、EEL_ID_BOND_1 と合わせて CFG_SECLIB_BOND_NUM で指定数以上になるようにします。以下は、CFG_SECLIB_BOND_NUM=8 の場合です。
- EEL_ID_BOND_*の個数は、CFG_SECLIB_BOND_NUM 以上であればよいので、以下の設定は、CFG_SECLIB_BOND_NUM=4 の場合でも動作します。ただしこの場合、ボンディング情報 4 個分の Data Flash 領域が使用されないままになります。

```
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 の編集方法

以下を参考に編集してください。

- secdb.h をインクルードします。
- SECDB_MD、SECDB_IRK の追加は必須です。
- SECDB_BOND 追加します。SECDB_BOND は合計で、EEL_ID_BOND_*と同数にしてください。以下は、CFG_SECLIB_BOND_NUM=8 の場合です。

```
1.  #include "secdb.h" <- Add (mandatory)
2.
3.  /* ... skip ... */
4.
5.  _EEL_CNST _EVENTfar const eel_u08 eel_descriptor[EEL_VAR_NO+2] =
6.  {
7.    (eel_u08)(EEL_VAR_NO),          /* variable count   */
8.    (eel_u08)(BD_ADDR_LEN),         /* id=1: EEL_ID_BDA */
9.    (eel_u08)(sizeof(SECDB_MD)),    // <- Add (mandatory)
10.   (eel_u08)(sizeof(SECDB_IRK)),   // <- Add (mandatory)
11.   (eel_u08)(sizeof(SECDB_BOND)),  // <- Add (mandatory)
12.   (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
13.   (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
14.   (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
15.   (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
16.   (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
17.   (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
18.   (eel_u08)(sizeof(SECDB_BOND)),  // <- Add
19.   (eel_u08)(0x00),                /* zero terminator  */
20. };
```

5. 内部動作シーケンス

5.1 SecLib_Init

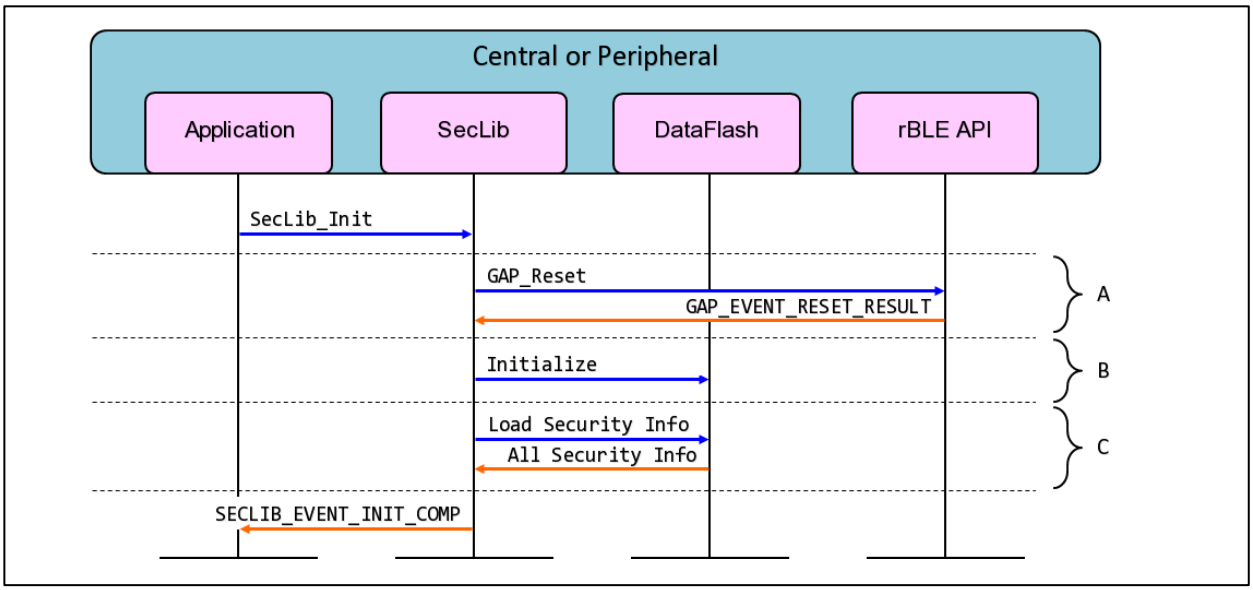


図 5-1 初期化シーケンス

A) GAP の Reset を行います。

B) RL78/G1D の内蔵データフラッシュの初期化を行います。

C) RL78/G1D の内蔵データフラッシュからセキュリティ情報を取得し、SRAM 上に保持します。

5.2 SecLib_Set_Param

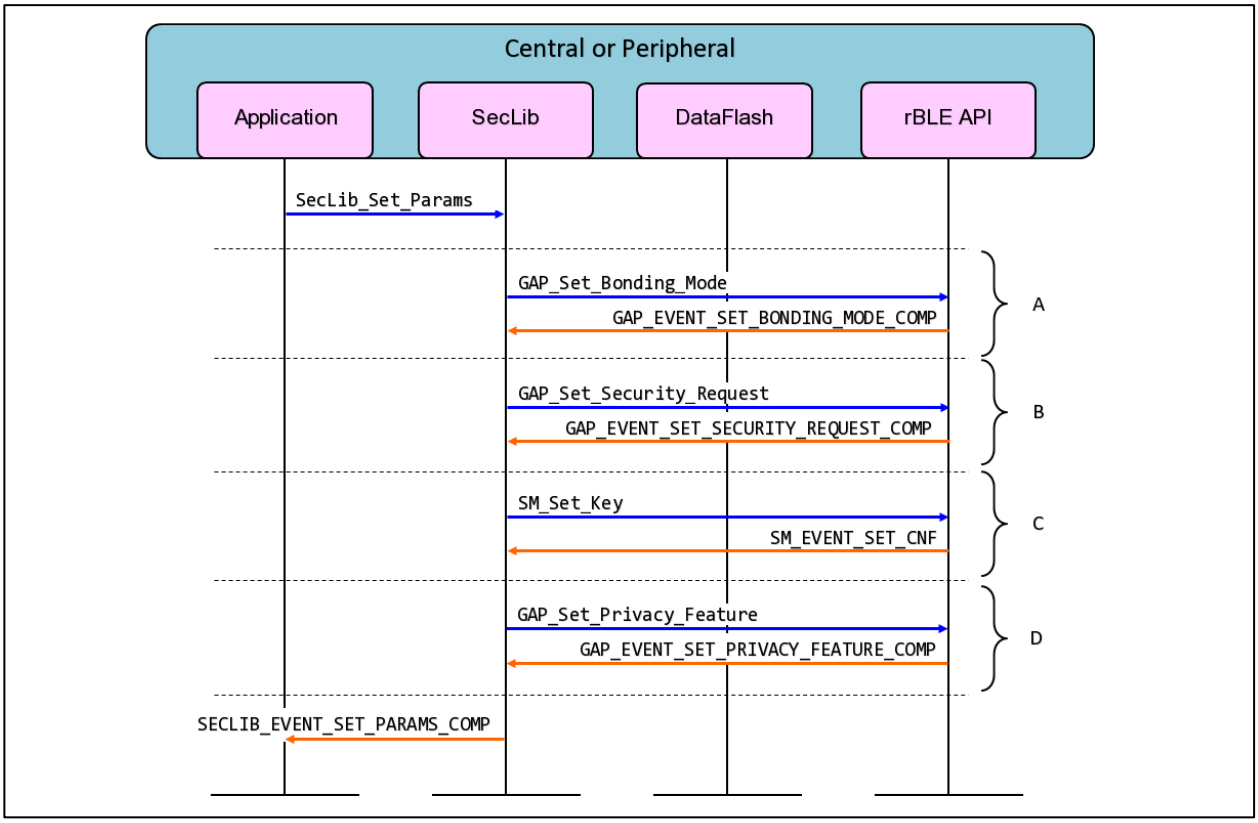


図 5-2 セキュリティパラメータ設定シーケンス

A) Bonding Mode を有効化します。

B) auth_req に従い Authentication Requirement の設定を行います。

C) rpa_generate が TRUE の場合、デバイスが使用する IRK を生成します。

D) プライバシーの設定を行います。

5.3 SecLib_Start_Encryption（Central 開始，ペアリング未実施時）

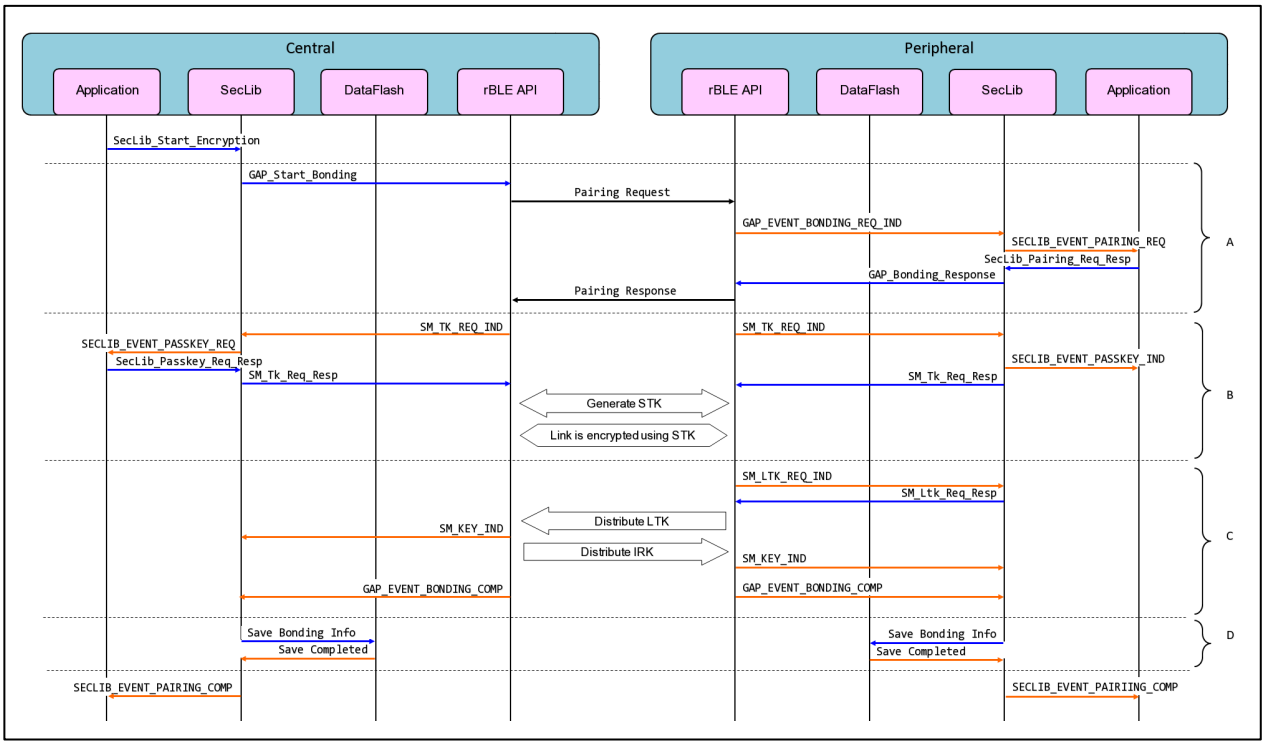


図 5-3 Central 開始ペアリングシーケンス

A) ペアリング Phase 1 : Central がペアリングを開始し、Peripheral がペアリングの開始を受諾します。

B) ペアリング Phase 2 : Passkey Entry によりデバイス間の認証を行います。

C) ペアリング Phase 3 : Central / Peripheral 間でセキュリティ情報の交換をします。

D) ペアリングが成功したため、ボンディング情報を保存します。

5.4 SecLib_Start_Encryption（Peripheral 開始，ペアリング未実施時）

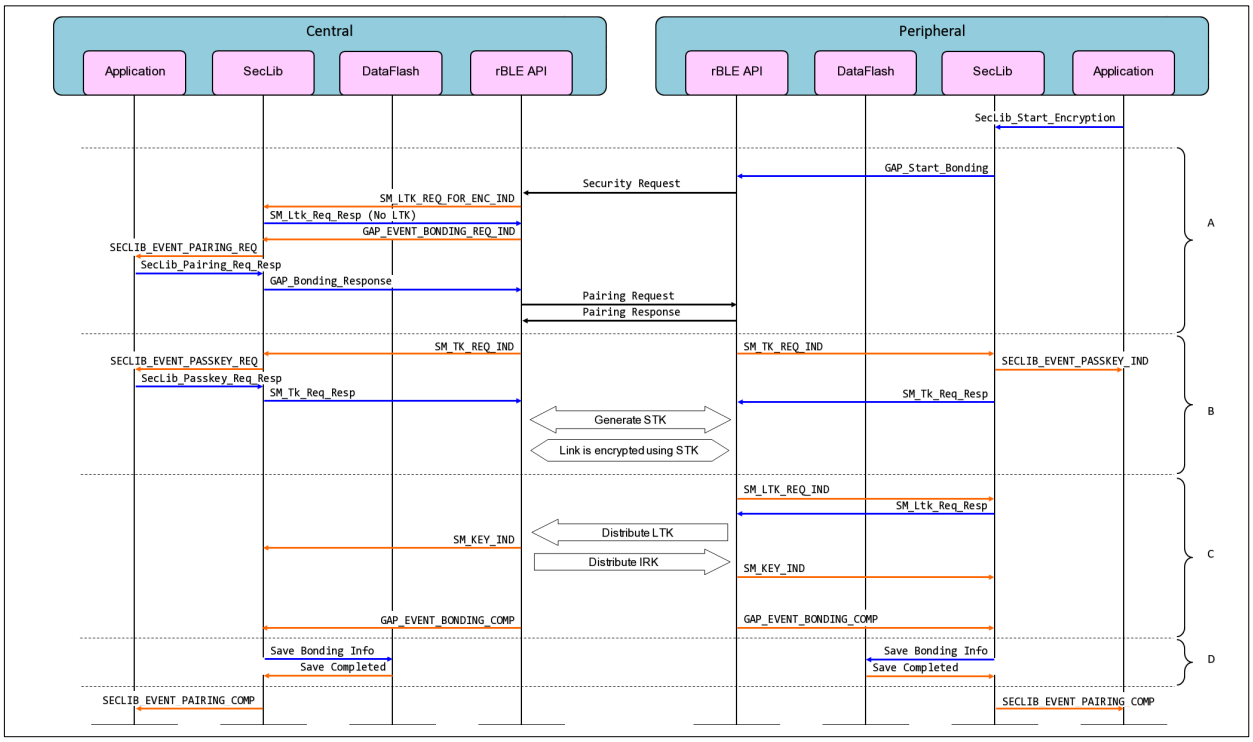


図 5-4 Peripheral 開始ペアリングシーケンス

A) ペアリング Phase 1 : Peripheral がセキュリティの確立を要求し、Central がペアリングを開始します。

B) ペアリング Phase 2 : Passkey Entry によりデバイス間の認証を行います。

C) ペアリング Phase 3 : Central / Peripheral 間でセキュリティ情報の交換をします。

D) ペアリングが成功したため、ボンディング情報を保存します。

5.5 SecLib_Start_Encryption（Central 開始，ペアリング実施済）

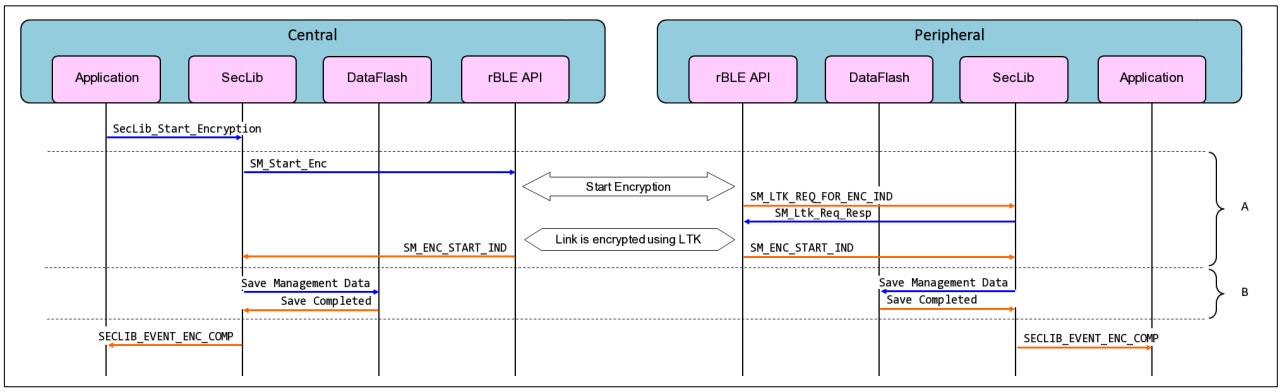


図 5-5 Central 開始暗号化シーケンス

A) Central から暗号化を開始します。

B) 暗号化が成功時に、LRU で使用する管理データを更新します。

5.6 SecLib_Start_Encryption（Peripheral 開始，ペアリング実施済）

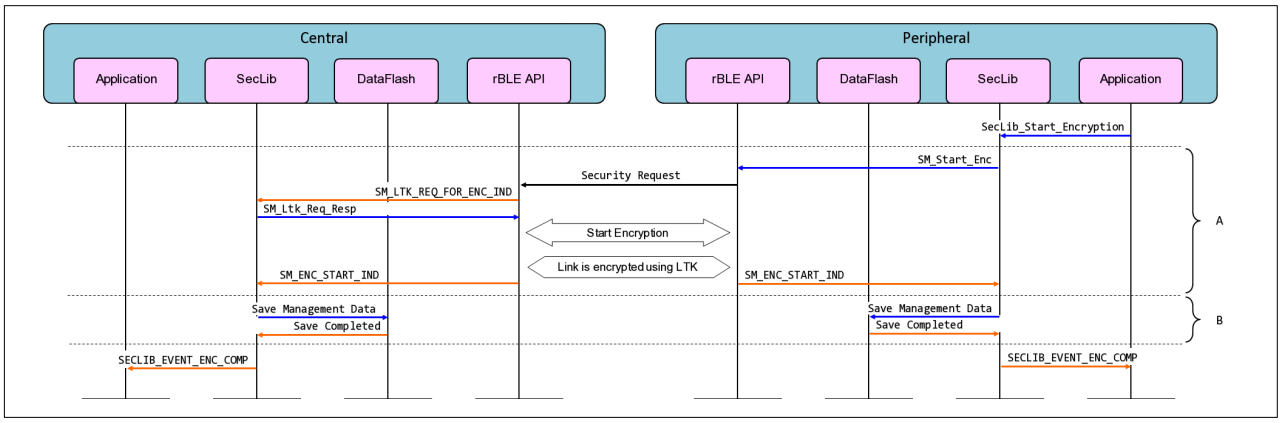
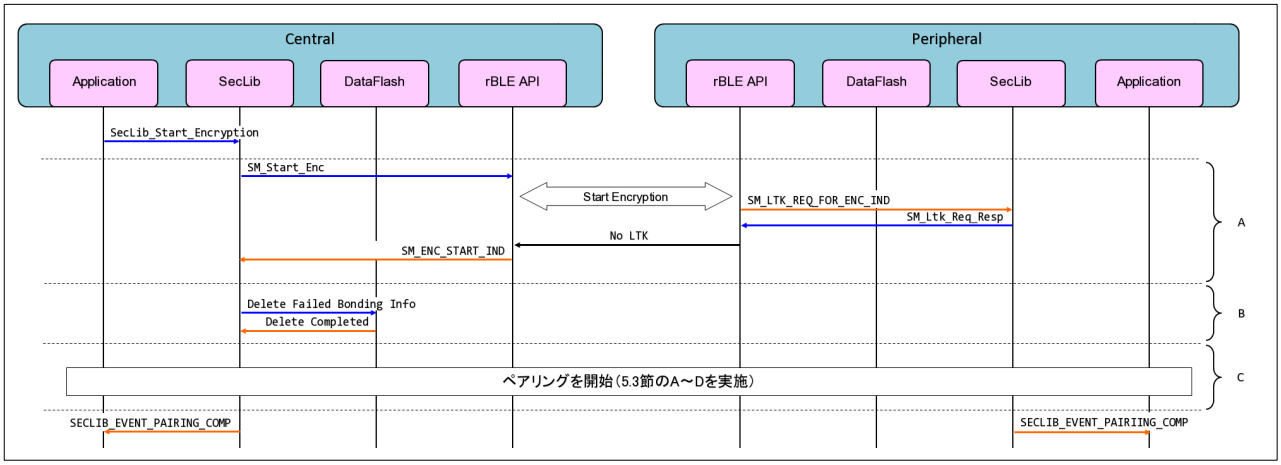


図 5-6 Peripheral 開始暗号化シーケンス

A) Peripheral からセキュリティの確立を要求します。Central は Peripheral とのペアリング時に保存したボンディング情報を使用して暗号化を開始します。

B) 暗号化が成功時に、LRU で使用する管理データを更新します。

5.7 SecLib_Start_Encryption（Central 開始、ペアリング実施済、Peripheral がボンディング情報を削除）



A) Central から暗号化を開始しますが、Peripheral がボンディング情報を削除しているため、暗号化に失敗します。

B) 暗号化に失敗したボンディング情報を削除します。[Note 1]

C) 暗号化に失敗した場合、ペアリングを開始します。[Note 1]

Note 1: ペアリング済みのデバイスと同じ BD アドレスをもつ悪意あるデバイスと接続した場合には、その悪意あるデバイスが暗号鍵を持たないため暗号化が失敗します。同梱している Security Library は暗号化失敗時に自動的にペアリングを開始し、悪意あるデバイスとのペアリングが成立する可能性があります。この問題を回避するために、6.2 節にしたがって暗号化失敗時に自動的にペアリングを開始しないように対策することを推奨します。

5.8 SecLib_Start_Encryption（Peripheral 開始、ペアリング実施済、Central がボンディング情報を削除）

5.4 節と同じシーケンスになります。

6. Appendix

6.1 ROM / RAM 使用量

Security Library が使用する ROM / RAM サイズを以下に示します。サイズは、CFG_SECLIB_BOND_NUM の設定により異なります。CFG_SECLIB_BOND_NUM を 4 または 7 にした場合サイズを示します。

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

(in bytes)

6.2 暗号化失敗時の脆弱性回避対策

ペアリング済みのデバイスと同じ BD アドレスをもつ悪意あるデバイスと接続した場合には、その悪意あるデバイスが暗号鍵を持たないため暗号化が失敗します。同梱している Security Library は暗号化失敗時に自動的にペアリングを開始し、悪意あるデバイスとのペアリングが成立する可能性があります。この問題を回避するために、以下にしたがって暗号化失敗時に自動的にペアリングを開始しないように対策することを推奨します。

6.2.1 シーケンス

図 6-1 に脆弱性のあるシーケンスを示します。

同梱しているオリジナルの Security Library を使用した Central が、何らかの方法でペアリング済みの Peripheral と同じ BD アドレスを取得した悪意あるデバイスに接続して Just Works でペアリングした場合、ペアリング済みの Peripheral のボンディング情報が自動的に削除されることが問題となります。

この問題を回避するためには、図 6-2 のように暗号化失敗時にアプリケーションにイベントを通知する変更が必要です。

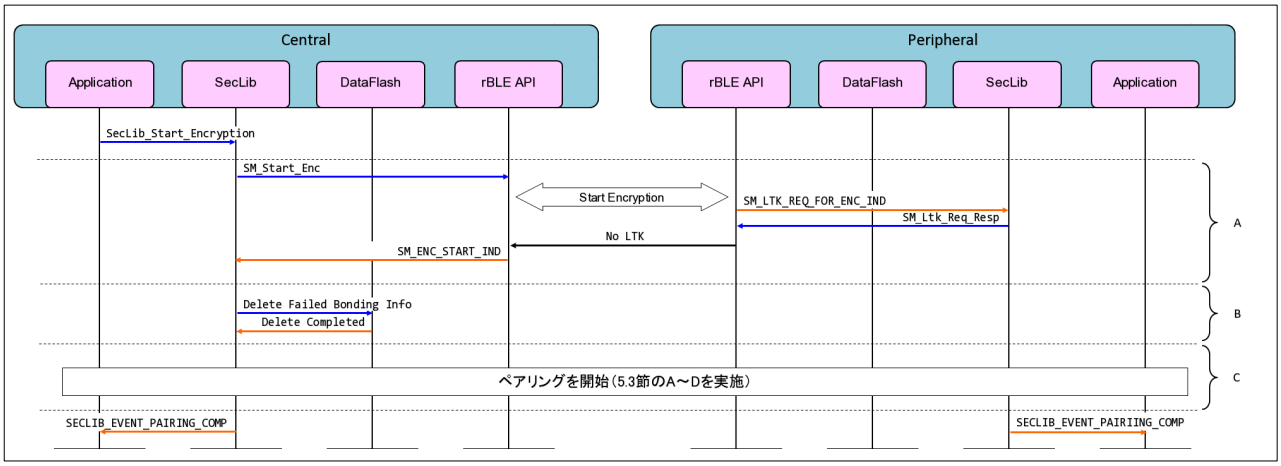


図 6-1 脆弱性のあるシーケンス

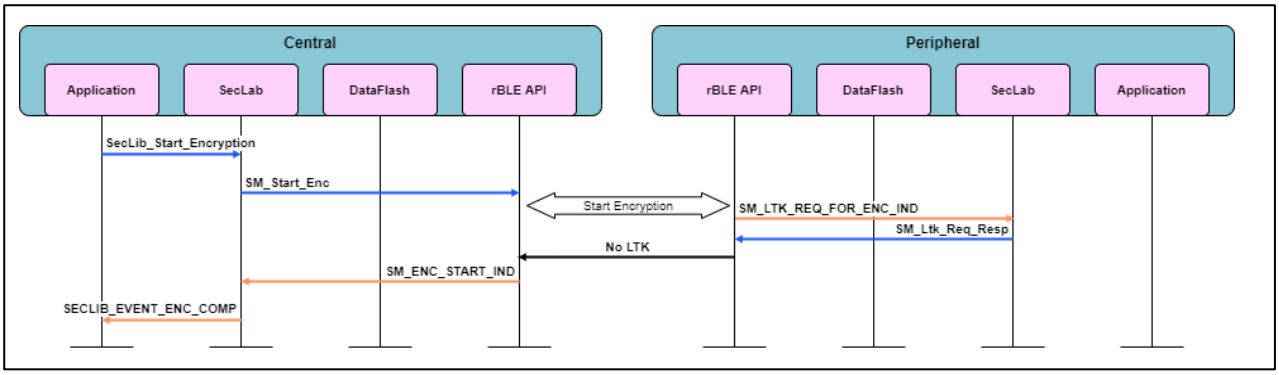


図 6-2 脆弱性を回避するシーケンス

6.2.2 脆弱性回避対策

脆弱性を回避するためには、暗号化失敗時にアプリケーションにイベントを通知するように変更します。

当該処理を実施する Security Library の関数は、図 6-3 に示す seclib.c の seclib_enc_failed()です。この関数の実装を図 6-4 のように変更してください。

```
1.  /* Encryption has finished with error. */
2.  static void seclib_enc_failed(uint16_t cidx)
3.  {
4.      RBLE_STATUS status = RBLE_OK;
5.      uint16_t bidx = CON(cidx).bidx;
6.      int i;
7.
8.      SECLIB_ERR("SecLib_Start_Encryption: Failed (encryption)", CON(cidx).con_state);
9.
10.     CON(cidx).con_state &= ~SECLIB_CON_STATE_ENC_COMP;
11.
12.     /* Delete failed bond info. */
13.     memset(&BOND(bidx), 0x00, sizeof(BOND(bidx)));
14.
15.     /* Check whether other pairing is ongoing. */
16.     for (i = 0; i < CFG_CON; i++) {
17.         if (CON_STATE_DONE(SECLIB_CON_STATE_IN_PAIRING, i)) {
18.             break;
19.         }
20.     }
21.
22.     /* If other pairing is not ongoing, delete bonding information. */
23.     if (i == CFG_CON) {
24.         info.db_cidx          = cidx;
25.         info.db_context       = SECLIB_DB_CONTEXT_ENC_DELTETE_COMP;
26.         info.db.md.data[bidx] = 0;
27.         status = SecDb_Save(&info.db, ((uint16_t)1 << bidx), FALSE, TRUE);
28.     }
29.
30.     /* Event if failed to delete md, go ahead. */
31.     if ((i != CFG_CON) || (status != RBLE_OK)) {
32.         seclib_send_enc_comp_event(RBLE_ERR, CON(cidx).conhdl, RBLE_SMP_KSEC_NONE);
33.     }
34. }
```

図 6-3 暗号化失敗時の処理（アプリケーションへ通知なし、ボンディング情報削除）

```
1.  /* Encryption has finished with error. */
2.  static void seclib_enc_failed(uint16_t cidx)
3.  {
4.      SECLIB_ERR("SecLib_Start_Encryption: Failed (encryption)", CON(cidx).con_state);
5.
6.      CON(cidx).con_state &= ~SECLIB_CON_STATE_ENC_COMP;
7.
8.      seclib_send_enc_comp_event(RBLE_ERR, CON(cidx).conhdl, RBLE_SMP_KSEC_NONE);
9.  }
```

図 6-4 暗号化失敗時の処理（アプリケーションへ通知）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2017.03.01	-	初版
1.01	2025.04.28	41	脆弱性の回避方法について記載
		18, 20, 39	脆弱性に関する注を記載し、6.2 節へのリンクを追加

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。Bluetooth®のワードマークおよびロゴは、Bluetooth SIG Inc. が所有する登録商標であり、ルネサス エレクトロニクス株式会社はこれらのマークをライセンスに基づいて使用しています。その他の商標および登録商標は、それぞれの所有者に帰属します。

© 2025 Renesas Electronics Corporation. All rights reserved.