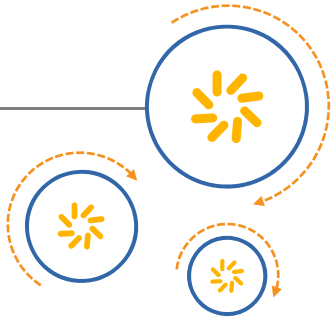
Qualcomm Technologies International, Ltd.



**Confidential and Proprietary – Qualcomm Technologies International, Ltd.**

*(formerly known as Cambridge Silicon Radio Ltd.)*

**NO PUBLIC DISCLOSURE PERMITTED:** Please report postings of this document on public servers or websites to: DocCtrlAgent@qualcomm.com.

**Restricted Distribution:** Not to be distributed to anyone who is not an employee of either Qualcomm Technologies International, Ltd.or its affiliated companies without the express approval of Qualcomm Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies International, Ltd.

Any software provided with this notice is governed by the Qualcomm Technologies International, Ltd. Terms of Supply or the applicable license agreement at https://www.csrsupport.com/CSRTermsandConditions.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. All Qualcomm Incorporated trademarks are used with permission. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer (“export”) laws. Diversion contrary to U.S. and international law is strictly prohibited.



Qualcomm Technologies International, Ltd.
Churchill House
Cambridge Business Park
Cambridge, CB4 0WZ
United Kingdom

CSR

Push every boundary.™

CSR µEnergy®

# Long Term Key (LTK)

## Application Note

Issue 2

# Document History

| Revision | Date | History |
|---|---|---|
| 1 | 10 MAY 13 | Original publication of this document |
| 2 | 03 FEB 14 | Added security request examples. Updated to new CSR branding. |

# Contacts


| | |
|---|---|
| General information | www.csr.com |
| Information on this product | sales@csr.com |
| Customer support for this product | www.csrsupport.com |
| More detail on compliance and standards | product.compliance@csr.com |
| Help with this document | comments@csr.com |

# Trademarks, Patents and Licences

Unless otherwise stated, words and logos marked with ™ or ® are trademarks registered or owned by CSR plc and/or its affiliates.

Bluetooth® and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR.

Other products, services and names used in this document may have been trademarked by their respective owners.

The publication of this information does not imply that any licence is granted under any patent or other rights owned by CSR plc or its affiliates.

CSR reserves the right to make technical changes to its products as part of its development programme.

While every care has been taken to ensure the accuracy of the contents of this document, CSR cannot accept responsibility for any errors.

Use of this document is permissible only in accordance with the applicable CSR licence agreement.

# Safety-critical Applications

CSR's products are not designed for use in safety critical devices or systems such as those relating to: (i) life support; (ii) nuclear power; and/or (iii) civil aviation applications, or other applications where injury or loss of life could be reasonably foreseeable as a result of the failure of a product. The customer agrees not to use CSR's products (or supply CSR's products for use) in such devices or systems.

# Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.

# Contents



# Tables, Figures and Equations

# 1. Introduction

This document describes how to store and retrieve an externally generated Long Term Key (LTK) in the Customer area of the Non-Volatile Memory (NVM) on CSR µEnergy® devices. The LTK is distributed during the pairing process between two Bluetooth® Smart devices. Externally generated LTKs are used to pre-bond devices, avoiding the need to have all devices present and communicating during the pairing process.

Example code fragments are used to illustrate the NVM storage and retrieval functions used during device production and with the device in normal use.

This document assumes that the reader is familiar with Bluetooth Low Energy security as specified in Volume 3 Part H Section 2.4 of the *Bluetooth Core Specification Version 4.1*.

## 1.1. Overview

A Long Term Key is one of the keys that are distributed while pairing two Bluetooth Smart devices. Pairing is performed when a connection between two devices is required to be encrypted for the first time. The final stage of pairing allows for keys including the LTK to be sent to the peer device over the air using an encrypted link. Pairing is not required on future connections if the keys are stored on both devices once they have been paired. All future connections can encrypt the link using the stored LTK.

Rather than pairing two devices over the air, both devices can be pre-bonded by storing the LTK when the devices are first programmed on the production line. The devices may be manufactured in separate locations and at different times but remain bonded.

## 1.2. Advantages of Pre-Bonding

- Devices may be bonded without being in close proximity.
- An Encrypted Diversifier (EDIV) and Random Number (Rand) are no longer required, as the LTK alone is sufficient to provide 128-bit security.
- Pre-bonding results in an LTK that has a 128-bit entropy whereas bonding using the pairing procedure as defined in the *Bluetooth Core Specification version 4.1* results in an LTK that has up to 20 bits of entropy (the key strength of a 6-digit PIN).
- Pre-bonding means there is no over-the-air pairing process that an attacker could monitor and compromise.

## 1.3. Pre-Bonding Devices Example

Table 1.1 lists the key elements required for a pre-bonded car and two associated key fobs:

| Device | Programmed in Production | | Generated in Normal Use |
|---|---|---|---|
| | Stored in CS | Stored in NVM | |
| Car (Master) | ER (Master)<br>IR (Master) | LTK (Slave 1)<br>IRK (Slave 1)<br>LTK (Slave 2)<br>IRK (Slave 2) | IRK (Master) from IR (Master) |
| Key Fob 1 (Slave 1) | ER (Slave 1)<br>IR (Slave 1) | LTK (Slave 1)<br>IRK (Master) | IRK (Slave 1) from IR (Slave 1) |
| Key Fob 2 (Slave 2) | ER (Slave 2)<br>IR (Slave 2) | LTK (Slave 2)<br>IRK (Master) | IRK (Slave 2) from IR (Slave 2) |

**Table 1.1: Pre-Bonding Example**

**Notes:**

1. It is strongly recommended that each Slave should have its own LTK, as shown
2. An API is provided to allow off-chip applications to program the keys into NVM.
3. An API is provided to allow on-chip applications to access the keys in the NVM.

Pre-bonding may be performed through the following process:

1. Generate the LTK to be shared between the Master and Slave devices. This may be an arbitrary 128-bit value, or it could be pre-calculated from the Slave's ER and the Master's DIV.
2. Download the LTK to each device, either:
   - Directly to the customer area of the NVM through a SPI link, see section 2.1, or
   - Indirectly using an on-chip application to receive the LTK over e.g. the UART link and store it in the customer area of NVM, see section 3.1

When the devices are ready to communicate over-the-air using Bluetooth, the LTK may be retrieved from NVM by an on-chip application using the example code in section 3.2.

## 1.4. Example of Link Encryption for Pre-Bonded Devices

Figure 1.1 shows how the pre-programmed link keys may be used by the protocol stack to perform link encryption:

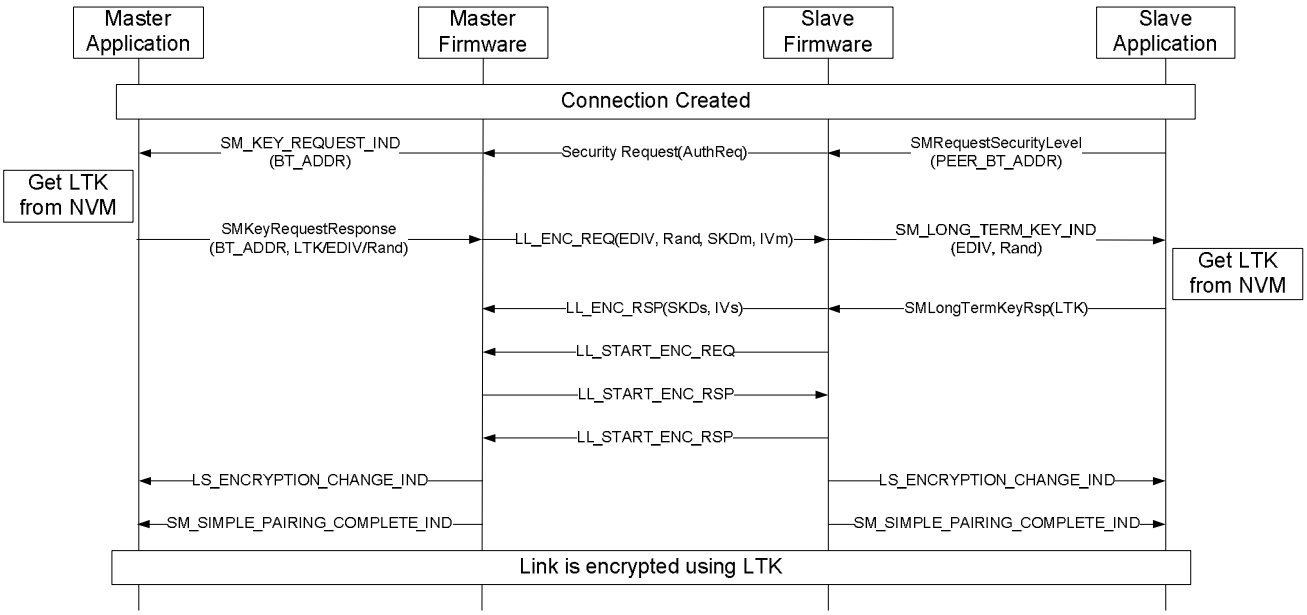



**Figure 1.1: Link Encryption Example for Pre-Bonded Devices**

**Notes:**

1. Events between the Firmware and the application are defined by the Firmware Library.
2. Messages between devices are defined in the *Bluetooth Core Specification Version 4.1*.
3. Only LTK needs to be read from NVM.
4. EDIV and Rand are not required to generate the LTK and may be ignored by the Slave, unless the Slave application wishes to use them for some other purpose.

## 1.5. Storing the LTK on the Slave

### 1.5.1. Where Does the Slave LTK Come From?

Normally the Slave generates the LTK using its ER and the DIV which in turn is generated by the EDIV supplied by the Master when it initiates encryption, see *Bluetooth Core Specification Version 4.1*, Volume 3, Part H, Section 5. However, with pre-bonding the Slave stores a pre-generated LTK in the customer area of NVM instead.

### 1.5.2. How Does the Slave Use a Stored LTK?

When the Slave Security Manager receives an `LL_ENC_REQ` PDU, it sends an `SM_LONG_TERM_KEY_IND` event to the application.

- If the application has been pre-bonded with the peer device, it retrieves the LTK from NVM and calls `SMLongTermKeyRsp()` with the stored LTK.
- If the application has not been pre-bonded with the peer device (and thus does not have an LTK available), it should call `SMLongTermKeyRsp()` with a NULL pointer for the LTK. This causes the Security Manager to regenerate the LTK as normal, hence avoiding the need for the application to store or approve the DIV.

**Notes:**

1. The LTK can be a simple 128-bit random number known by both sides, unrelated to EDIV and Rand, in which case EDIV and Rand could have any value.
2. If the application never uses pre-bonded LTKs then `SMLongTermKeyRsp()` should not be called. When the application is built and linked against the firmware library, a check is made to see whether `SMLongTermKeyRsp()` is called by the application, and if it is not then at runtime the firmware will not send the application any `SM_LONG_TERM_KEY_IND` events. Instead the Security Manager always generates the LTK internally.

## 1.6. Storing the IRK

### 1.6.1. Where Does the IRK Currently Come From?

The IRK is generated on-chip from the Identity Root (IR) which is a secret value provided in a CS Key that remains on-chip, see *Bluetooth Core Specification Version 4.1*, Volume 3, Part H, Section 5. A device distributes its own IRK during pairing if it uses a Resolvable Private Address (RPA); a device that generates the RPA must distribute its own IRK to enable the peer device to resolve the RPA. As the exchange of IRKs is asymmetric, it is not necessary for both devices to distribute their IRKs.

### 1.6.2. How is a Stored IRK Used?

The IRK of each peer device using a RPA may be stored instead of being received over the air during pairing. The IRK is generated from the peer's IR which can be randomly generated. It is expected that the IR for each device using a RPA is created when the LTK is generated during production. The IRs are then used to generate the IRKs using the standard algorithms, see *Bluetooth Core Specification Version 4.1*, Volume 3, Part H, Section 5. The shared LTK, private IR and peer IRK are then programmed into each device. This allows the device to resolve the peer's RPA and to establish an encrypted connection using the shared LTK.

# 2. Accessing the LTK from Off-Chip

It is expected that the LTK will be programmed into devices during the production phase. Functions have been added to `uEnergyTest.dll`, supplied in CSR µEnergy Tools, so that the LTK may be programmed into the Customer area of NVM immediately before or after hardware tests have been performed. See the *CSR µEnergy Tools help* for details on the API.

**Note:**

The location of the LTK stored in NVM is entirely at the application's discretion. The examples place the LTK at the beginning of the NVM, but this location is not mandatory.

## 2.1. Example Code for Writing to the Customer Area in NVM

```
CsrHandle_t handle = 0;
int32 retVal = uetOpen("SPITRANS=USB SPIPORT=0", NULL, &handle);
if (retVal == UET_OK)
{
    // Read CS
    retVal = uetCsCacheReadFromFile(handle, "cstest.keyr");

    // Write LTK to the start of the customer NVM area
    if (retVal == UET_OK)
    {
        uint16 ltk[8] = { 0x0e0f, 0x0c0d, 0x0a0b, 0x0809,
                          0x0607, 0x0405, 0x0203, 0x0001 };
        retVal = uetNvmCustomWrite(handle, UET_NVM_TYPE_EEPROM, 0,
                                   ltk, sizeof(ltk) / sizeof(ltk[0]));
    }

    uetClose(handle);
}
```

## 2.2. Example Code for Reading from the Customer Area in NVM

```
CsrHandle_t handle = 0;
int32 retVal = uetOpen("SPITRANS=USB SPIPORT=0", NULL, &handle);
if (retVal == UET_OK)
{
    // Read CS
    retVal = uetCsCacheReadFromFile(handle, "cstest.keyr");

    // Read LTK from the start of the customer NVM area
    uint16 ltk[8];
    if (retVal == UET_OK)
    {
        retVal = uetNvmCustomRead(handle, UET_NVM_TYPE_EEPROM, 0,
                                  ltk, sizeof(ltk) / sizeof(ltk[0]));
    }

    uetClose(handle);
}
```

# 3. Accessing the LTK from On-Chip

The LTK may be retrieved from the Customer area in NVM by the on-chip application using standard firmware library routines and by including `nvm.h` in the source files. See the *Firmware Library documentation* (NVM in the Memory Management module) included in the CSR µEnergy Software Development Kit (SDK) for further details.

The same functions are used to access NVM for both EEPROM and SPI Flash memory types. The application must inform the NVM manager which type of device is connected by calling either `NvmConfigureI2cEeprom()` or `NvmConfigureSpiFlash()`.

**Note:**

The location of the LTK stored in NVM is entirely at the application's discretion. The examples place the LTK at the beginning of the NVM, but this location is not mandatory.

## 3.1. Example Code tor Writing to the Customer Area in NVM

```
sys_status result = sys_status_success;
uint16 ltk[8] = { 0x0e0f, 0x0c0d, 0x0a0b, 0x0809, 0x0607, 0x0405, 0x0203, 0x0001 };

/* NvmWrite automatically enables the NVM before writing */
result = NvmWrite(ltk, sizeof(ltk)/sizeof(ltk[0]), 0);

/* Disable NVM after writing and power off the storage device */
NvmDisable();
```

## 3.2. Example Code for Reading from the Customer Area in NVM

```
sys_status result = sys_status_success;
uint16 ltk[8];

/* NvmRead automatically enables the NVM before reading */
result = NvmRead(ltk, sizeof(ltk)/sizeof(ltk[0]), 0);

/* Disable NVM after reading and power off the storage device */
NvmDisable();
```

## 3.3. Example Code for a Master Responding to a Security Request

```
bool AppProcessLmEvent(lm_event_code event_code, LM_EVENT_T *p_event_data)
{
    sys_status result = sys_status_success; /* Function status */

    switch (event_code)
    {
        /* The following event indicates that the Security Manager could not
         * find security keys for the peer device in its persistent store.
         */
        case SM_KEY_REQUEST_IND:
        {
            result = MasterLTKRequest((SM_KEY_REQUEST_IND_T *)p_event_data);
        }
        break;
    }

    return TRUE;
}

sys_status MasterLTKRequest(SM_KEY_REQUEST_IND_T *key_request)
{
    sys_status result = sys_status_success; /* Function status */
    SM_KEYSET_T keyset;                     /* Set of security keys */
    uint16 ltk[8];                          /* Long Term Key */

    /* Retrieve the LTK from NVM (see examples) */
    result = MasterLTKRetrieve(ltk);
    if (result == sys_status_success)
    {
        /* Package the LTK in a keyset */
        keyset.keys_present = 1 << SM_KEY_TYPE_ENC_CENTRAL;
                                                /* Peer LTK + EDIV + Rand */
        keyset.encryption_key_size = 16;        /* Key size in octets */
        MemCopy(keyset.ltk, ltk, 8);            /* Long Term Key */
        /* EDIV and Rand need only be supplied if the Slave application wants */
        /* to approve the bond when setting up the encrypted link. */
        keyset.ediv = 0;                        /* EDIV, optional */
        MemSet(keyset.rand, 0, 4);              /* Rand, optional */

        /* Return LTK to the Security Manager */
        SMKeyRequestResponse(&key_request->remote_addr, &keyset);

        /* The Security Manager will use the key set in the LL_ENC_REQ PDU */
        /* sent to the Slave to request an encrypted link. */
    }

    return result;
}
```

## 3.4. Example Code for a Slave Responding to an Encryption Request

```
bool AppProcessLmEvent(lm_event_code event_code, LM_EVENT_T *p_event_data)
{
    sys_status result = sys_status_success; /* Function status */

    switch (event_code)
    {
        /* The following event indicates that the Security Manager has received
         * an encryption request from the peer device, and the application has
         * indicated that it wants to manage some Long Term Keys itself.
         */
        case SM_LONG_TERM_KEY_IND:
        {
            result = SlaveLTKRequest((SM_LONG_TERM_KEY_IND_T *)p_event_data);
        }
        break;
    }

    return TRUE;
}

sys_status SlaveLTKRequest(SM_LONG_TERM_KEY_IND_T *key_ind)
{
    sys_status result = sys_status_success; /* Function status */
    uint16 ltk[8];                          /* Long Term Key */

    /* Retrieve the LTK from NVM (see examples) */
    result = SlaveLTKRetrieve(ltk);
    if (result == sys_status_success)
    {
        /* Provide Security Manager with Slave's stored LTK */
        SMLongTermKeyRsp(key_ind->cid, ltk, TRUE, 16);
    }

    return result;
}
```

# Document References

| Document | Reference |
|---|---|
| *Bluetooth Core Specification Version 4.1* | https://www.bluetooth.org/Technical/Specifications/adopted.htm |
| *Firmware Library documentation* | Supplied with the CSR µEnergy SDK support documentation |
| *CSR µEnergy Tools Help* | Available from the CSR µEnergy Tools supplied with the CSR µEnergy SDK. |

# Terms and Definitions

| | |
|---|---|
| API | Application Program Interface |
| Bluetooth® | Set of technologies providing audio and data transfer over short-range radio connections |
| Bluetooth Smart | Formerly known as Bluetooth low energy |
| CS | Configuration Store |
| CSR | Cambridge Silicon Radio |
| DIV | Diversifier |
| e.g. | *exempli gratia*, for example |
| EDIV | Encrypted Diversifier. A 16-bit stored value used to identify the LTK |
| EEPROM | Electrically Erasable Programmable Read Only Memory |
| ER | Encryption Root |
| etc. | *et cetera*, and the rest, and so forth |
| i.e. | *id est*, that is |
| IR | Identity Root |
| IRK | Identity Resolving Key. A 128-bit key used to generate and resolve random addresses |
| IV | Initialisation Vector. Composed of two parts, IVm (master) and IVs (slave) |
| LTK | Long Term Key. A 128-bit key used to generate the contributory session key for an encrypted connection. |
| NVM | Non-Volatile Memory |
| PDU | Protocol Data Unit |
| PIN | Personal Identification Number |
| Rand | Random Number. A 64-bit value used to identify the LTK |
| RPA | Resolvable Private Address |
| SDK | Software Development Kit |
| SKD | Session Key Diversifier. Composed of two parts, SKDm (master) and SKDs (slave) |
| SPI | Serial Peripheral Interface |