Application note

DA14580 Resolvable private address guidelines

AN-B-022

Abstract

Bluetooth Low Energy devices are identified using a device address. Device addresses may be either a public device address or a random device address. A device with a random resolvable private address can be identified by bonded peers in subsequent advertising or connection sessions even when it uses a different address. Guidelines and methods of implementing devices using and resolving resolvable private addresses are described in this document.

Contents

Figures

1 Terms and definitions

IRK	Identity Resolving Key
LTK	Long Term Key
MSB	Most Significant Bits
MITM	Man In The Middle

2 References

1. Bluetooth Specification version 4.0
2. RW-BLE-GAP-IS, GAP Interface Specification, Riviera Waves

3 Introduction

This document describes the procedures that must be followed by a DA14580 host application in order to:

- Configure its local device address as a resolvable private address.
- Identify a bonded peer device using a resolvable private address in subsequent scanning or connection operations.

4 Random resolvable address mode

4.1 Set up device configuration

A resolvable private address is generated from an Identity Resolving Key (IRK). The IRK must be generated by the host application and provided to the BLE stack in the corresponding field of a GAPM_SET_DEV_CONFIG_CMD message.

If the device role is set to peripheral role (GAP_PERIPHERAL_SLV), the privacy bit in the flag field (GAPM_CFG_PRIVACY_EN) must be set.

4.2 Advertising

In order to use a resolvable random address during advertising, the host application must set *op.addr_src* to GAPM_GEN_RSLV_ADDR in the GAPM_START_ADVERTISE_CMD command. Field *op.renew_dur* controls the duration of the resolvable address before it gets regenerated and it is counted in units of 10 ms. Minimum valid value for *renew_dur* is 15000 (150 s). If the value of *renew_dur* is < 15000, the BLE stack will automatically set it to 15000.

The BLE stack will send a GAPM_DEV_BDADDR_IND event containing the generated random resolvable address to the host application upon receiving the GAPM_START_ADVERTISE_CMD command and each time a new resolvable random address is generated. The host application may store the generated random address for future usage. The random address generation process is active until the GAPM_START_ADVERTISE_CMD command is completed, i.e. until a peer connects to the device or the host application cancels advertising with a GAPM_CANCEL_CMD command.

A GAPM_START_ADVERTISE_CMD command with *op.addr_src* = GAPM_GEN_RSLV_ADDR always starts by generating a new resolvable random address. If a host application has a stored address and wishes to reuse the last generated resolvable random address in subsequent advertising commands, it can set the *op.addr_src* field to GAPM_PROVIDED_RND_ADDR and copy the stored address in *op.addr* to force the BLE stack to use this address.

4.3 Scan/connection

Similar to the advertising procedure, the host application in central role devices can set *op.addr_src* to GAPM_GEN_RSLV_ADDR in commands GAPM_START_SCAN_CMD and GAPM_START_CONNECTION_CMD to enable random resolvable private addressing. The address is generated by the BLE stack and returned to the host application in a GAPM_DEV_BDADDR_IND message.

The application can set *op.addr_src* to GAPM_PROVIDED_RND_ADDR in these two commands to reuse a previously generated random address.

5 Bonding procedure

A device using a resolvable private address should be able to distribute its local IRK to peer devices during the bonding procedure. Otherwise it will not be possible for its peer devices to identify the device when it updates its resolvable private random address.

The bonding procedure must be initiated by the central device with a GAPC_BOND_CMD command.

5.1 IRK distribution from a central device perspective

The host application initiates the bonding procedure using a GAPC_BOND_CMD command with:

- Set GAP_KDIST_IDKEY bit of *pairing.ikey_dist* to enable distribution of its own IRK
- Set GAP_KDIST_IDKEY bit of *pairing.rkey_dist* in GAPC_BOND_CMD to request peer IRK distribution

During the bonding procedure the central host application will receive a GAPC_BOND_IND message with *info* = GAPC_IRK_EXCH containing the peer's IRK in *data.irk.*

5.2 IRK distribution from a peripheral device perspective

During the bonding procedure initiated by the peer central device, the host application of the peripheral device receives a GAPC_BOND_REQ_IND with *request* = GAPC_PAIRING_REQ.

The application will respond with a GAPC_BOND_CFM with *request* = GAPC_PAIRING_RSP and

- Set GAP_KDIST_IDKEY bit of *data.pairing.rkey_dist* to enable the distribution of its own IRK
- Set GAP_KDIST_IDKEY bit of *data.pairing.ikey_dist* to request peer IRK distribution

The BLE stack will handle the distribution of the peripheral's IRK to the peer central device.

Later during the bonding procedure, the peripheral host application will receive a GAPC_BOND_IND message with *info* = GAPC_IRK_EXCH containing the peer's IRK in *data.irk*.

6 Resolving the address

A device can resolve a resolvable address by sending a GAPM_RESOLV_ADDR_CMD command to TASK_GAPM with the address to be resolved in *addr field* and the stored IRKs of bonded devices.

If the address is resolved by any of the provided IRKs, the TASK_GAPM will respond with a GAPM_ADDR_SOLVED_IND message containing the resolved address and the IRK resolving it. Otherwise, a GAPM_CMP_EVT event or GAPM_RESOLV_ADDR operation with status value GAP_ERR_NOT_FOUND will be sent.

7 Example

This section describes an example of a DA14580 device with a central role and a public address connecting to a DA14580 device with peripheral role and resolvable private random address. For better understanding of resolvable private address usage, the message flow of host applications running on both devices will be described for the following procedures:

1. Configure and start advertising with resolvable private address on peripheral device.
2. Bonding procedure.
3. Resolve peripheral device random address during scanning procedure.

7.1 Advertising with resolvable private address

The host application of the DA14580 in a peripheral role must generate the IRK before device configuration. This can be done in the *app_init()* function. The following member is added in *app_sec_env* to store the IRK data. The IRK size always has the maximum value: KEY_LEN = 16 (bytes).

```
struct gap_sec_key irk;
```

A sample function of pseudo random key generation is as follows:

```
void app_sec_gen_irk(void)
{
    // Counter
    uint8_t i;
    // Randomly generate the end of the LTK
    for (i = 0; i < KEY_LEN; i++)
    {
        app_sec_env.irk.key[i] = rand()%256;
    }
}
```

In a device configuration command, the application provides the stored local IRK and IRK size. It must also enable the peripheral privacy flag. Below follows a sample code with the required actions for this operation.

```
// set device configuration
struct gapm_set_dev_config_cmd* cmd = KE_MSG_ALLOC(GAPM_SET_DEV_CONFIG_CMD,
TASK_GAPM, TASK_APP, gapm_set_dev_config_cmd);

cmd->role = GAP_PERIPHERAL_SLV;
memcpy (cmd->irk.key, app_sec_env.irk.key, KEY_LEN);
cmd->flags = GAPM_CFG_PRIVACY_EN;
```

In a start advertising command, the application must set *addr_src* to GAPM_GEN_RSLV_ADDR. The *renew_dur* field must be set to the required lifetime of the random address before it gets regenerated.

```
struct gapm_start_advertise_cmd *cmd = KE_MSG_ALLOC(GAPM_START_ADVERTISE_CMD,
                                       TASK_GAPM, TASK_APP,gapm_start_advertise_cmd);

cmd->op.addr_src = GAPM_GEN_RSLV_ADDR;
cmd->op.renew_dur = <Random address lilftime>; // In units of 10ms. Min = 15000,  Max
= 65535
```

7.2 Bonding procedure

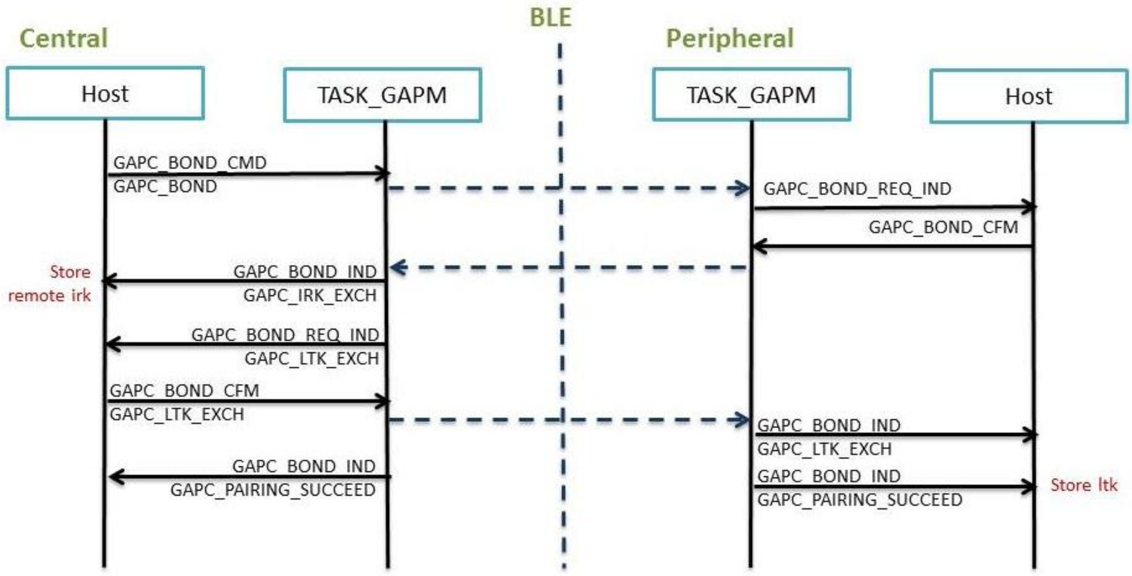

Figure 1: Message flow during bonding procedure

A central device can discover and connect to peripherals with a resolvable private address.

The central device can determine whether a discovered address is a resolvable random address by examining the combination of *adv_addr_type* in GAPM_ADV_REPORT_IND or *peer_addr_type* in GAPC_CONNECTION_REQ_IND with the value of the two most significant bits in the address. The address type of resolvable addresses is '0' for public addresses and '1' for random addresses. The two most significant bits of resolvable private addresses are equal to '01'.

For more details about resolvable private address format refer to [1], *Volume 3, 10.8.2.2 Resolvable Private Address Generation Procedure*.

After establishing a connection, the host application in a central device must initiate the bonding procedure with a peripheral to obtain the remote IRK. The bonding procedure is initiated with a GAPM_BOND_CMD command.

Other security parameters (such as OOB data presence, MITM, LTK distributor) can have various values depending on the application requirements. For this example we will assume that MITM is disabled, OOB data are not present and the LTK distributor will be the security initiator (central).

Below is a sample code enabling this setup:

```
struct gapc_bond_cmd * req = KE_MSG_ALLOC (GAPC_BOND_CMD, KE_BUILD_ID(TASK_GAPC,
app_env.conidx), TASK_APP, gapc_bond_cmd);
req->operation = GAPC_BOND;
req->pairing.sec_req = GAP_NO_SEC;
// OOB information
req->pairing.oob = GAP_OOB_AUTH_DATA_NOT_PRESENT;  // No OOB data present
// IO capabilities
req->pairing.iocap          = GAP_IO_CAP_NO_INPUT_NO_OUTPUT; // No I/O capabilities
// Authentication requirements
req->pairing.auth           = GAP_AUTH_REQ_NO_MITM_BOND; //Bonding, No MITM
// Encryption key size
req->pairing.key_size       = 16;
```

```
//Initiator key distribution
req->pairing.ikey_dist      = SMP_KDIST_ENCKEY; //Initiator distributes LTK
//Responder key distribution
req->pairing.rkey_dist      = SMP_KDIST_IDKEY; //Request receiver to distribute IRK
```

The host application in the peripheral device will receive a GAPC_BOND_REQ_IND message with *request* = GAPC_PAIRING_REQ, indicating that the central device has started the pairing (bonding) procedure. It must respond with a GAPC_BOND_CFM. Below is the sample code for the response:

```
struct gapc_bond_cfm* cfm = KE_MSG_ALLOC(GAPC_BOND_CFM, TASK_GAPC, TASK_APP,
gapc_bond_cfm);

cfm->request = GAPC_PAIRING_RSP; //pairing response
cfm->accept = true;
// OOB information
cfm->data.pairing_feat.oob            = GAP_OOB_AUTH_DATA_NOT_PRESENT; // No OOB data
present
// Encryption key size
cfm->data.pairing_feat.key_size       = KEY_LEN;
// IO capabilities
cfm->data.pairing_feat.iocap          = GAP_IO_CAP_NO_INPUT_NO_OUTPUT; // No I/O
capabilities
// Authentication requirements
cfm->data.pairing_feat.auth           = GAP_AUTH_REQ_NO_MITM_BOND; //Bonding, No MITM
//Security requirements
cfm->data.pairing_feat.sec_req        = GAP_NO_SEC;
//Initiator key distribution
cfm->data.pairing_feat.ikey_dist      = SMP_KDIST_ENCKEY; //Initiator distributes LTK
//Responder key distribution
cfm->data.pairing_feat.rkey_dist      = SMP_KDIST_IDKEY; //Request receiver to
distribute IRK
```

The host application in the central device will receive an indication that the remote IRK is distributed. The host must extract the IRK of the remote device (from the *irk* structure in the data union) and store it for use in resolving the peripheral device’s random address for reconnection. Below is the sample code:

```
int gapc_bond_ind_handler(ke_msg_id_t msgid,
                          struct gapc_bond_ind *param,
                          ke_task_id_t dest_id,
                          ke_task_id_t src_id)
{
    switch (param->info)
    {
           //... other cases ...
           case GAPC_IRK_EXCH:
               // store param->data.irk in the application environment
               // in a variable of type struct gapc_irk
            break;
           //... other cases ...
    }
    return 0;
}
```

On the peripheral side, no IRK distribution is requested at the host application layer. The local IRK is stored in BLE stack layers and IRK distribution is handled in the BLE stack.

Since the central device is the distributor of LTK in this example, its host application will receive a GAPC_BOND_REQ_IND message with *request* = GAPC LTK_EXCH and must distribute the LTK with the corresponding GAPC_BOND_CFM message.

The host application on the peripheral device will receive a GAPC_BOND_IND message with *request* = GAPC LTK_EXCH, containing the distributed LTK. Both devices must store the LTK information for establishing secure links in subsequent connections.

The bonding procedure will be completed by the host applications on both devices by the reception of a GAPC_BOND_IND message with field *info* = GAPC_PAIRING_SUCCEED.

Message flow during bonding procedure on both devices is outlined Figure [1].

8 Resolving peripheral device random address during scanning

The central host application starts scanning as usual. Upon receiving an advertising report (GAPM_ADV_REPORT_IND) the application checks whether it is from a device with a resolvable random address:

```
int gapm_adv_report_ind_handler(ke_msg_id_t msgid,
                                struct gapm_adv_report_ind *param,
                                ke_task_id_t dest_id,
                                ke_task_id_t src_id)
{
        if(param->report.adv_addr_type == 1
                && ( (param->report.adv_addr.addr[5] & 0xC0) = 0x40) )
        {
                // it is a resolvable random address
        }
        return (KE_MSG_CONSUMED);
}
```

Next, resolve the address by issuing the command GAPM_RESOLVE_ADDR_CMD:

```
struct gapm_resolv_addr_cmd *cmd =
(struct gapm_resolv_addr_cmd *)KE_MSG_ALLOC_DYN(GAPM_RESOLVE_ADDR_CMD, TASK_GAPM,
                     TASK_APP, gapm_resolv_addr_cmd, N * sizeof(struct gap_sec_key) );
cmd->operation = GAPM_RESOLV_ADDR; // GAPM requested operation
cmd->nb_key = N;                   // Number of provided IRK
cmd->addr = <<addr>> ; /// Resolvable random address to solve
cmd->irk = <<array of stored IRKs>>; /// Array of IRK used for address resolution (MSB
-> LSB)

/// Resolve Address command
struct gapm_resolv_addr_cmd
{
    /// GAPM requested operation:
    ///  - GAPM_RESOLV_ADDR: Resolve device address
    uint8_t operation;
    /// Number of provided IRK (sahlle be > 0)
    uint8_t nb_key;
    /// Resolvable random address to solve
    struct bd_addr addr;
    /// Array of IRK used for address resolution (MSB -> LSB)
    struct gap_sec_key irk[__ARRAY_EMPTY];
};
```

The host application indicates that the address is resolved with a GAPM_ADDR_SOLVED_IND message, containing the resolved address and the IRK resolving it.

```
/// Indicate that resolvable random address has been solved
struct gapm_addr_solved_ind
{
    /// Resolvable random address solved
    struct bd_addr addr;
    /// IRK that correctly solved the random address
    struct gap_sec_key irk;
};
```

When no provided IRK resolves the address, a GAPM_CMP_EVT event or GAPM_RESOLV_ADDR operation with *status* = GAP_ERR_NOT_FOUND will be sent.

9 Revision history

Revision	Date	Description
1.0	10-Apr-2014	Initial version.

Status definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

Disclaimer

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including without limitation the specification and the design of the related semiconductor products, software and applications.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Customer notes that nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document are subject to Dialog Semiconductor's Standard Terms and Conditions of Sale, unless otherwise stated.

RoHS Compliance

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2).
Dialog Semiconductor's statement on RoHS can be found on the customer portal https://support.diasemi.com/. RoHS certificates from our suppliers are available on request.

Contacting Dialog Semiconductor

Germany Headquarters
Dialog Semiconductor GmbH
Phone: +49 7021 805-0

United Kingdom
Dialog Semiconductor (UK) Ltd
Phone: +44 1793 757700

The Netherlands
Dialog Semiconductor B.V.
Phone: +31 73 640 8822

Email:
enquiry@diasemi.com

North America
Dialog Semiconductor Inc.
Phone: +1 408 845 8500

Japan
Dialog Semiconductor K. K.
Phone: +81 3 5425 4567

Taiwan
Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

Singapore
Dialog Semiconductor Singapore
Phone: +65 64 849929

China
Dialog Semiconductor China
Phone: +86 21 5178 2561

Korea
Dialog Semiconductor Korea
Phone: +82 2 3469 8291