wicentric

Device Manager API

Document 2009-0008
Revision 1.5
September 25, 2015

Table of Contents

1 Introduction

This document describes the API of the Device Manager (DM) subsystem. The device manager is responsible for many important operations of the protocol stack such as:

- Advertising and device visibility.
- Scanning and device discovery.
- Connection management.
- Security management.
- Local device management.

2 Main Interface

2.1 Constants and Data Types

2.1.1 Device Role

This parameter identifies the device role.

Name	Value	Description
DM_ROLE_MASTER	0	Role is master.
DM_ROLE_SLAVE	1	Role is slave.

2.1.2 Discoverability Mode

This parameter sets the GAP discoverability mode.

Name	Value	Description
DM_DISC_MODE_NONE	0	GAP non-discoverable. Peer devices performing GAP discovery cannot discover this device.
DM_DISC_MODE_LIMITED	1	GAP limited discoverable mode. Peer devices performing GAP limited discovery can discover this device.
DM_DISC_MODE_GENERAL	2	GAP general discoverable mode. Peer devices performing GAP limited or general discovery can discover this device.

2.1.3 Advertising Type

The advertising type indicates the connectable and discoverable nature of the advertising packets transmitted by a device.

Name	Value	Description
DM_ADV_CONN_UNDIRECT	0	Connectable undirected advertising. Peer devices can scan and connect to this device.
DM_ADV_CONN_DIRECT	1	Connectable directed advertising. Only a specified peer device can connect to this device.

DM_ADV_DISC_UNDIRECT	2	Discoverable undirected advertising. Peer devices can scan this device but cannot connect.
DM_ADV_NONCONN_UNDIRECT	3	Non-connectable undirected advertising. Peer devices cannot scan or connect to this device.
DM_ADV_SCAN_RESPONSE	4	Scan response. Transmitted in response to an active scan.

2.1.4 Address Type

The address type indicates whether an address is public or random.

Name	Value	Description
DM_ADDR_PUBLIC	0	Public address.
DM_ADDR_RANDOM	1	Random address.

2.1.5 Advertising and Scan Intervals

Advertising and scan intervals in this API are specified in 0.625 msec units, as defined in [3].

2.2 Functions

2.2.1 void DmRegister(dmCback_t cback)

Register a callback with DM for scan and advertising events.

- **cback**: Client callback function. See 2.3.1.

2.2.2 uint8_t *DmFindAdType(uint8_t adType, uint8_t dataLen, uint8_t *pData)

Find an advertising data element in the given advertising or scan response data.

- **adType**: Advertising data element type to find. See 3.2.
- **dataLen**: Data length.
- **pData**: Pointer to advertising or scan response data.

This function returns a pointer to the advertising data element byte array or NULL if not found.

2.3 Callback Interface

2.3.1 void (*dmCback_t)(dmEvt_t *pDmEvt)

This callback function sends DM events to the client.

- **pDmEvt**: Pointer to DM event structure.

2.3.2 Callback Events

The following callback event values are passed in the DM event structure.

Name	Description
DM_RESET_CMPL_IND	Reset complete.
DM_ADV_START_IND	Advertising started.
DM_ADV_STOP_IND	Advertising stopped.

DM_ADV_NEW_ADDR_IND	New resolvable address has been generated.
DM_SCAN_START_IND	Scanning started.
DM_SCAN_STOP_IND	Scanning stopped.
DM_SCAN_REPORT_IND	Scan data received from peer device.
DM_CONN_OPEN_IND	Connection opened.
DM_CONN_CLOSE_IND	Connection closed.
DM_CONN_UPDATE_IND	Connection update complete.
DM_SEC_PAIR_CMPL_IND	Pairing completed successfully.
DM_SEC_PAIR_FAIL_IND	Pairing failed or other security failure.
DM_SEC_ENCRYPT_IND	Connection encrypted.
DM_SEC_ENCRYPT_FAIL_IND	Encryption failed.
DM_SEC_AUTH_REQ_IND	PIN or OOB data requested for pairing.
DM_SEC_KEY_IND	Security key indication.
DM_SEC_LTK_REQ_IND	LTK requested for encyption.
DM_SEC_PAIR_IND	Incoming pairing request from master.
DM_SEC_SLAVE_REQ_IND	Incoming security request from slave.
DM_PRIV_RESOLVED_ADDR_IND	Private address resolved.
DM_VENDOR_SPEC_IND	Vendor specific event.

3 Advertising and Device Visibility

The DM interface for advertising and device visibility configures, enables, and disables the advertising procedure. A device advertises when it wishes to connect to or be discovered by other devices. Devices may also advertise to simply broadcast data.

This interface can only be used when operating as a slave.

3.1 Constants and Data Types

3.1.1 Data Location

This parameter indicates whether data is located in the advertising data or the scan response data.

Name	Value	Description
DM_DATA_LOC_ADV	0	Locate data in the advertising data.
DM_DATA_ LOC_SCAN	1	Locate data in the scan response data.

3.2 Advertising Data Element Types

This parameter indicates the type of advertising data element.

Name	Description
DM_ADV_TYPE_FLAGS	Flag bits.
DM_ADV_TYPE_16_UUID_PART	Partial list of 16 bit UUIDs.
DM_ADV_TYPE_16_UUID	Complete list of 16 bit UUIDs.
DM_ADV_TYPE_128_UUID_PART	Partial list of 128 bit UUIDs.
DM_ADV_TYPE_128_UUID	Complete list of 128 bit UUIDs.

DM_ADV_TYPE_SHORT_NAME	Shortened local name.
DM_ADV_TYPE_LOCAL_NAME	Complete local name.
DM_ADV_TYPE_TX_POWER	TX power level.
DM_ADV_TYPE_CONN_INTERVAL	Slave preferred connection interval.
DM_ADV_TYPE_SIGNED_DATA	Signed data.
DM_ADV_TYPE_16_SOLICIT	Service solitiexpress list of 16 bit UUIDs.
DM_ADV_TYPE_128_SOLICIT	Service solitiexpress list of 128 bit UUIDs.
DM_ADV_TYPE_SERVICE_DATA	Service data.
DM_ADV_TYPE_PUBLIC_TARGET	Public target address.
DM_ADV_TYPE_RANDOM_TARGET	Random target address.
DM_ADV_TYPE_APPEARANCE	Device appearance.
DM_ADV_TYPE_MANUFACTURER	Manufacturer specific data.

3.3 Advertising Channel Map

This parameter indicates the advertising channel map.

Name	Description
DM_ADV_CHAN_37	Advertising channel 37.
DM_ADV_CHAN_38	Advertising channel 38.
DM_ADV_CHAN_39	Advertising channel 39.
DM_ADV_CHAN_ALL	All advertising channels.

3.4 Functions

3.4.1 void DmAdvInit(void)

Initialize DM advertising. This function is typically called once at system startup.

3.4.2 void DmAdvStart(uint8_t advType, uint16_t duration)

This function is called to start advertising using the given advertising type and duration.

- **advType**: Advertising type. See 2.1.3.
- **duration**: The advertising duration, in milliseconds. If set to zero, advertising will continue until DmAdvStop() is called or a connection is established.

If advertising is started successfully the client's callback function is called with a DM_ADV_START_IND event. If advertising fails to start for any reason the client's callback function is called with a DM_ADV_STOP_IND event. The client's callback function is also called with a DM_ADV_STOP_IND event if the advertising duration expires or DmAdvStop() is called.

Example for GAP general discoverable mode:

```
    DmAdvStart(DM_ADV_CONN_UNDIRECT, 30720);
```

Example for GAP broadcast mode:

```
    DmAdvStart(DM_ADV_NONCONN_UNDIRECT, 0);
```

3.4.3 void DmAdvStop(void)

This function is called to stop advertising. When advertising is stopped the client's callback function is called with a DM_ADV_STOP_IND event.

3.4.4 void DmAdvSetInterval(uint16_t intervalMin, uint16_t intervalMax)

This function sets the minimum and maximum advertising intervals. This function should only be called when advertising is stopped.

- **intervalMin**: Minimum advertising interval. See 2.1.4.
- **intervalMax**: Maximum advertising interval. See 2.1.4.

3.4.5 void DmAdvSetChannelMap(uint8_t channelMap)

This function is used to include or exclude certain channels from the advertising channel map. This function should only be called when advertising is stopped.

- **channelMap**: Advertising channel map. See 3.3.

3.4.6 void DmAdvSetData(uint8_t location, uint8_t len, uint8_t *pData)

This function sets the advertising or scan response data to the given data. The data will replace any existing data already present with the same advertising data type.

- **location**: Data location. See 3.1.1.
- **len**: Length of the data. Maximum length is 31 bytes.
- **pData**: Pointer to the data.

3.4.7 void DmAdvSetAddrType (uint8_t addrType)

Set the local address type used while advertising. This function can be used to configure advertising to use a random or private address.

- **addrType**: Address type. See 2.1.4.

3.4.8 DmAdvSetAdValue(uint8_t adType, uint8_t len, uint8_t *pValue, uint8_t *pAdvDataLen, uint8_t *pAdvData)

Set the value of an advertising data element in the given advertising or scan response data. If the element already exists in the data then it is replaced with the new value. If the element does not exist in the data it is appended to it, space permitting.

- **adType**: Advertising data element type.
- **len**: Length of the value. Maximum length is 29 bytes.
- **pValue**: Pointer to the value.
- **pAdvDataLen**: Advertising or scan response data length. The new length is returned in this parameter.
- **pAdvData**: Pointer to advertising or scan response data.

Returns TRUE if the element was successfully added to the data, FALSE otherwise.

3.4.9 DmAdvSetName(uint8_t len, uint8_t *pValue, uint8_t *pAdvDataLen, uint8_t *pAdvData)

Set the device name in the given advertising or scan response data. If the name can only fit in the data if it is shortened, the name is shortened and the AD type is changed to DM_ADV_TYPE_SHORT_NAME.

- **len**: Length of the name. Maximum length is 29 bytes.
- **pValue**: Pointer to the name in UTF-8 format.
- **pAdvDataLen**: Advertising or scan response data length. The new length is returned in this parameter.
- **pAdvData**: Pointer to advertising or scan response data.

Returns TRUE if the element was successfully added to the data, FALSE otherwise.

3.4.10 void DmAdvPrivInit(void)

Initialize private advertising. This function is typically called once at system startup to enable the use of advertising with a private resolvable address.

3.4.11 void DmAdvPrivStart(uint16_t changeInterval)

Start using a private resolvable address and start periodic generation of a new address.

When a new address is generated the client's callback function is called with a DM_ADV_NEW_ADDR_IND event. The application must wait to receive this event once before starting advertising.

To stop using a private resolvable address call function DmAdvPrivStop().

This function should not be used when the device is operating as a master, as master devices are forbidden from using a private resolvable address.

- **changeInterval**: Interval between automatic address changes, in seconds.

3.4.12 void DmAdvPrivStop(void)

Stop using a private resolvable address.

3.5 Callback Interface

3.5.1 DM_ADV_START_IND: Advertising Started

Callback event for advertising started.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.

3.5.2 DM_ADV_STOP_IND: Advertising Stopped

Callback event for advertising stopped.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.

3.5.3 DM_ADV_NEW_ADDR_IND: New Resolvable Address Has Been Generated

Callback event for new resolvable address has been generated.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
bdAddr_t	addr	New resolvable private address.
bool_t	firstTime	TRUE when address is generated for the first time.

4 Scanning and Device Discovery

The DM scanning and device discovery interface configures, enables, and disables the scanning procedure. A device scans when it wishes to discover or connect to other devices. A device may also scan simply to receive broadcast advertisements.

This interface can only be used when operating as a master.

4.1 Constants and Data Types

4.1.1 Scan Type

This parameter indicates the scan type. A passive scan only receives advertising packets. An active scan receives advertising packets and scan response packets.

Name	Value	Description
DM_SCAN_TYPE_PASSIVE	0	Passive scan.
DM_SCAN_TYPE_ACTIVE	1	Active scan.

4.2 Functions

4.2.1 void DmScanInit(void)

Initialize DM scanning. This function is typically called once at system startup.

4.2.2 void DmScanStart(uint8_t mode, uint8_t scanType, bool_t filterDup, uint16_t duration)

This function is called to start scanning. A scan is performed using the given discoverability mode, scan type, and duration.

- **mode**: Discoverability mode. See 2.1.1.
- **scanType**: Scan type. See 4.1.1.
- **filterDup**: Filter duplicates. Set to TRUE to filter duplicate responses received from the same device. Set to FALSE to receive all responses.
- **duration**: The scan duration, in milliseconds. If set to zero, scanning will continue until DmScanStop() is called.

If scanning is started successfully the client's callback function is called with a DM_SCAN_START_IND event. If scanning fails to start for any reason the client's callback function is called with a DM_SCAN_STOP_IND event. The client's callback function is also called with a DM_SCAN_STOP_IND event if the scan duration expires or DmScanStop() is called.

Example for GAP limited discovery:

```
    DmScanStart(DM_DISC_MODE_LIMITED, DM_SCAN_TYPE_ACTIVE, TRUE,
    10240);
```

Example for GAP general discovery:

```
    DmScanStart(DM_DISC_MODE_GENERAL, DM_SCAN_TYPE_ACTIVE, TRUE,
    10240);
```

Example for GAP observe procedure:

```
    DmScanStart(DM_DISC_MODE_NONE, DM_SCAN_TYPE_PASSIVE, FALSE, 0);
```

4.2.3 void DmScanStop(void)

This function is called to stop scanning. When scanning is stopped the client's callback function is called with a DM_SCAN_STOP_IND event.

4.2.4 void DmScanSetInterval(uint16_t scanInterval, uint16_t scanWindow)

This function sets the scan interval and window. This function should only be called when scanning is stopped.

- **scanInterval**: The scan interval. See 2.1.4.
- **scanWindow**: The scan window. See 2.1.4.

4.2.5 void DmScanSetAddrType (uint8_t addrType)

Set the local address type used while scanning. This function can be used to configure scanning to use a random or private address.

- **addrType**: Address type. See 2.1.4.

4.3 Callback Interface

4.3.1 DM_SCAN_START_IND: Scanning Started

Callback event for scanning started.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.

4.3.2 DM_SCAN_STOP_IND: Scanning Stopped

Callback event for scanning stopped.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.

4.3.3 DM_SCAN_REPORT_IND: Scan Report

Callback event for scan report. This event uses type hciLeAdvReportEvt_t defined in [1].

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
uint8_t *	pData	Pointer to received data.
uint8_t	len	Data length.
int8_t	rssi	RSSI of received packet.
uint8_t	eventType	Scan report event type. See 2.1.3.
uint8_t	addrType	Peer address type.
bdAddr_t	addr	Peer address.

5 Connection Management

The DM connection management interface is used to open, accept, configure, and close connections. It is also used to read connection-related information such as the RSSI, channel map, and remote device information.

5.1 Constants and Data Types

5.1.1 Client ID

The client ID parameter to function DmConnRegister() identifies the client to the DM connection manager. The possible values are shown below.

Name	Description
DM_CONN_ATT_ID	Identifier for attribute protocol. For internal use only.
DM_CONN_SMP_ID	Identifier for security manager protocol. For internal use only.
DM_CONN_DM_ID	Identifier for device manager. For internal use only.
DM_CONN_APP_ID	Identifier for the application.

5.1.2 dmConnId_t

This data type is used for the connection identifier. The connection identifier uniquely identifies the connection.

5.1.3 Connection Busy/Idle State

The connection busy/idle state indicates when the connection is busy with a stack protocol procedure, such as pairing or service discovery. The application can use this state to decide whether or not to perform certain connection operations such as a connection parameter update.

Name	Description
DM_CONN_IDLE	Connection is idle.
DM_CONN_BUSY	Connection is busy.

5.1.4 Busy/Idle State Bitmask

The connection busy/idle bitmask indicates which stack protocol procedure or application procedure is busy.

Name	Description
DM_IDLE_SMP_PAIR	SMP pairing in progress.
DM_IDLE_DM_ENC	DM Encryption setup in progress.
DM_IDLE_ATTS_DISC	ATTS service discovery in progress.
DM_IDLE_APP_DISC	App framework service discovery in progress.
DM_IDLE_USER_1	For use by user application.
DM_IDLE_USER_2	For use by user application.
DM_IDLE_USER_3	For use by user application.
DM_IDLE_USER_4	For use by user application.

5.2 Functions

5.2.1 DmConnInit(void)

Initialize DM connection manager. This function is typically called once at system startup.

5.2.2 DmConnMasterInit(void)

Initialize DM connection manager for operation as master. This function is typically called once at system startup.

5.2.3 DmConnSlaveInit(void)

Initialize DM connection manager for operation as slave. This function is typically called once at system startup.

5.2.4 void DmConnRegister(uint8_t clientId, dmCback_t cback)

This function is called by a client to register with the DM connection manager. After registering the client can call other functions in the API to open, close, update or accept a connection. The client will also receive DM connection events via its callback for all connections, whether or not initiated by the client.

- **clientId**: The client identifier. See 5.1.1.
- **cback**: Client callback function. See 2.3.1.

5.2.5 dmConnId_t DmConnOpen(uint8_t clientId, uint8_t addrType, uint8_t *pAddr)

This function opens a connection to a peer device with the given address. This function can only be called when operating as a master.

- **clientId**: The client identifier. See 5.1.1.
- **addrType**: Address type. See 2.1.4.

- **pAddr**: Peer device address.

This function returns a connection identifier. When the connection is opened the client's callback function is called with a DM_CONN_OPEN_IND event. If the connection fails for any reason the client's callback function is called with a DM_CONN_CLOSE_IND event.

5.2.6 void DmConnClose(uint8_t clientId, dmConnId_t connId, uint8_t reason)

This function closes the connection with the give connection identifier. This function can be called when operating as a master or slave.

- **clientId**: The client identifier. See 5.1.1.
- **connId**: Connection identifier. See 5.1.2.
- **reason**: Reason connection is being closed.

When the connection is closed the client's callback function is called with a DM_CONN_CLOSE_IND event.

5.2.7 dmConnId_t DmConnAccept(uint8_t clientId, uint8_t addrType, uint8_t *pAddr)

This function accepts a connection from the given peer device by initiating directed advertising. This function can only be called when operating as a slave.

- **clientId**: The client identifier. See 5.1.1.
- **addrType**: Address type. See 2.1.4.
- **pAddr**: Peer device address.

This function returns a connection identifier. When the connection is opened the client's callback function is called with a DM_CONN_OPEN_IND event. If the connection fails for any reason or if the connection is not opened within 1.28 seconds the client's callback function is called with a DM_CONN_CLOSE_IND event.

5.2.8 void DmConnUpdate(dmConnId_t connId, hciConnSpec_t *pConnSpec)

This function updates the connection parameters of an open connection. This function can be called when operating as a master or a slave.

- **connId**: Connection identifier. See 5.1.2.
- **pConnSpec**: Connection specification. See [1].

5.2.9 void DmConnSetScanInterval(uint16_t scanInterval, uint16_t scanWindow)

This function sets the scan interval and window for created connections created with DmConnOpen(). This function must be called before calling DmConnOpen() for the parameters to be in effect.

- **scanInterval**: The scan interval. See 2.1.4.
- **scanWindow**: The scan window. See 2.1.4.

5.2.10 void DmConnSetConnSpec(hciConnSpec_t *pConnSpec)

This function sets the connection specification parameters for connections created with DmConnOpen(). This function must be called before calling DmConnOpen() for the parameters to be in effect.

- **pConnSpec**: Connection specification. See [1].

5.2.11 void DmConnSetAddrType (uint8_t addrType)

Set the local address type used for connections created with DmConnOpen(). This function can be used to create connections using a random or private address.

- **addrType**: Address type. See 2.1.4.

5.2.12 void DmConnSetIdle(dmConnId_t connId, uint16_t idleMask, uint8_t idle)

Configure a bit in the connection idle state mask as busy or idle.

- **connId**: Connection identifier. See 5.1.2.
- **idleMask**: Bit in the idle state mask to configure. See 5.1.4.
- **idle**: DM_CONN_BUSY or DM_CONN_IDLE. See 5.1.3.

5.2.13 uint16_t DmConnCheckIdle(dmConnId_t connId)

Check if a connection is idle.

- **connId**: Connection identifier. See 5.1.2.

This function returns zero if the connection is idle or nonzero if busy.

5.3 Callback Interface

5.3.1 DM_CONN_OPEN_IND: Connection Opened

Callback event for connection opened. This event uses type hciLeConnCmplEvt_t defined in [1].

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.
uint16_t	handle	Connection handle.
uint8_t	role	Connection role.
uint8_t	addrType	Address type.
bdAddr_t	peerAddr	Peer address.
uint16_t	connInterval	Connection interval.
uint16_t	connLatency	Connection latency.
uint16_t	supTimeout	Connection supervision timeout.
uint8_t	clockAccuracy	Peer clock accuracy.

5.3.2 DM_CONN_CLOSE_IND: Connection Closed

Callback event for connection closed. This event uses type hciDisconnectCmplEvt_t defined in [1].

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.
uint16_t	handle	Connection handle.
uint8_t	reason	Disconnect reason.

5.3.3 DM_CONN_UPDATE_IND: Connection Update

Callback event for connection update complete. This event uses type hciLeConnUpdateCmplEvt_t defined in [1].

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.
uint8_t	status	Status of connection update procedure.
uint16_t	handle	Connection handle.
uint16_t	connInterval	Connection interval.
uint16_t	connLatency	Connection latency.
uint16_t	supTimeout	Supervision timeout.

6 Local Device Management

The DM local device management interface is used for initialization and reset, setting local parameters, sending vendor-specific commands, and LE GAP attribute management.

6.1 Functions

6.1.1 void DmDevReset(void)

This function initiates the HCI reset sequence. When the reset sequence is complete the client's callback function is called with a DM_RESET_CMPL_IND event.

6.1.2 uint8_t DmDevRole(void)

This function returns the device role indicating master or slave. See 2.1.1

6.1.3 void DmDevSetRandAddr(uint8_t *pAddr)

Set the random address to be used by the local device.

- **pAddr**: Random address.

6.1.4 void DmDevWhiteListAdd(uint8_t addrType, uint8_t *pAddr)

Add a peer device to the white list. Note that this function cannot be called while advertising, scanning, or connecting with white list filtering active.

- **addrType**: Address type. See 2.1.4.
- **pAddr**: Peer device address.

6.1.5 void DmDevWhiteListRemove(uint8_t addrType, uint8_t *pAddr)

Remove a peer device from the white list. Note that this function cannot be called while advertising, scanning, or connecting with white list filtering active.

- **addrType**: Address type. See 2.1.4.
- **pAddr**: Peer device address.

6.1.6 void DmDevWhiteListClear(void)

Clear the white list. Note that this function cannot be called while advertising, scanning, or connecting with white list filtering active.

6.2 Callback Interface

6.2.1 DM_RESET_CMPL_IND: Reset Complete

Callback event for reset complete.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.

7 Security Management

The DM security management interface is used for pairing, authentication, and encryption.

7.1 Constants and Data Types

7.1.1 Authentication Flags

This parameter contains the authentication flags of a procedure or its associated data.

Name	Value	Description
DM_AUTH_BOND_FLAG	0x01	Bonding requested.
DM_AUTH_MITM_FLAG	0x04	MITM (authenticated pairing) requested.

7.1.2 Key Distribution

This parameter contains a bit mask of the keys distributed during the pairing procedure.

Name	Value	Description
DM_KEY_DIST_LTK	0x01	Distribute LTK used for encryption.
DM_KEY_DIST_IRK	0x02	Distribute IRK used for privacy.
DM_KEY_DIST_CSRK	0x04	Distribute CSRK used for signed data.

7.1.3 Key Type

This parameter indicates the key type used in DM_SEC_KEY_IND.

Name	Description

DM_KEY_LOCAL_LTK	LTK generated locally for this device.
DM_KEY_PEER_LTK	LTK received from peer device.
DM_KEY_IRK	IRK and identity info of peer device.
DM_KEY_CSRK	CSRK of peer device.

7.1.4 Security Level

This parameter indicates the security level of a connection.

Name	Description
DM_SEC_LEVEL_NONE	Connection has no security.
DM_SEC_LEVEL_ENC	Connection is encrypted with unauthenticated key.
DM_SEC_LEVEL_ENC_AUTH	Connection is encrypted with authenticated key.
DM_SEC_LEVEL_ENC_LESC	Connection is encrypted with LE Secure Connections.

7.1.5 Security Error Codes

These error codes can be used in the status parameter of security functions and callback event structures.

Name	Value	Description
SMP_ERR_PASSKEY_ENTRY	0x01	User input of passkey failed.
SMP_ERR_OOB	0x02	OOB data is not available.
SMP_ERR_AUTH_REQ	0x03	Authentication requirements cannot be met.
SMP_ERR_CONFIRM_VALUE	0x04	Confirm value does not match.
SMP_ERR_PAIRING_NOT_SUP	0x05	Pairing is not supported by the device.
SMP_ERR_ENC_KEY_SIZE	0x06	Insufficient encryption key size.
SMP_ERR_COMMAND_NOT_SUP	0x07	Command not supported.
SMP_ERR_UNSPECIFIED	0x08	Unspecified reason.
SMP_ERR_ATTEMPTS	0x09	Repeated attempts.
SMP_ERR_INVALID_PARAM	0x0A	Invalid parameter or command length.
SMP_ERR_DH_KEY_CHECK	0x0B	DH Key check did not match
SMP_ERR_NUMERIC_COMPARISON	0x0C	Numeric comparison did not match
SMP_ERR_BR_EDR_IN_PROGRESS	0x0D	BR/EDR in progress
SMP_ERR_CROSS_TRANSPORT	0x0E	BR/EDR Cross transport key generation not allowed
SMP_ERR_MEMORY	0xE0	Out of memory.
SMP_ERR_TIMEOUT	0xE1	Transaction timeout.

7.1.6 Keypress Types

These values are used in to notify the peer of a keypress event types.

Name	Value	Description
SMP_PASSKEY_ENTRY_STARTED	0x00	Passkey entry started keypress type.
SMP_PASSKEY_DIGIT_ENTERED	0x01	Passkey digit entered keypress type

SMP_PASSKEY_DIGIT_ERASED	0x02	Passkey digit erased keypress type
SMP_PASSKEY_CLEARED	0x03	Passkey cleared keypress type
SMP_PASSKEY_ENTRY_COMPLETED	0x04	Passkey entry complete keypress type

7.1.7 dmSecLtk_t

This data structure is the LTK data type.

Type	Name	Description
uint8_t	key[SMP_KEY_LEN]	Key.
uint8_t *	rand[SMP_RAND8_LEN]	Random identifier for key.
uint16_t	ediv	Diversifier for key.

7.1.8 dmSecIrk_t

This data structure is the IRK data type.

Type	Name	Description
uint8_t	key[SMP_KEY_LEN]	Key.
bdAddr_t	bdAddr	Peer device address.
uint8_t	addrType	Peer device address type.

7.1.9 dmSecCsrk_t

This data structure is the CSRK data type.

Type	Name	Description
uint8_t	key[SMP_KEY_LEN]	Key.

7.1.10 dmSecKey_t

This data structure is a union of key types.

Type	Name	Description
dmSecLtk_t	ltk	LTK.
dmSecIrk_t	irk	IRK.
dmSecCsrk_t	csrk	CSRK.

7.2 Function Interface

7.2.1 void DmSecInit(void)

Initialize DM security manager. This function is typically called once at system startup.

7.2.2 void DmSecPairReq(dmConnId_t connId, bool_t oob, uint8_t auth, uint8_t iKeyDist, uint8_t rKeyDist)

This function is called by a master device to initiate pairing.

- **connId**: Connection identifier. See 5.1.2.
- **oob**: Out-of-band pairing data present or not present.
- **auth**: Authentication and bonding flags. See 7.1.1.
- **iKeyDist**: Initiator key distribution flags. See 7.1.2.
- **rKeyDist**: Responder key distribution flags. See 7.1.2.

When the pairing procedure is complete the client's callback function is called with a DM_SEC_PAIR_CMPL_IND event if successful or a DM_SEC_PAIR_FAIL_IND if failure.

7.2.3 void DmSecPairRsp(dmConnId_t connId, bool_t oob, uint8_t auth, uint8_t iKeyDist, uint8_t rKeyDist)

This function is called by a slave device to proceed with pairing after a DM_SEC_PAIR_IND event is received. This function must be called within 30 seconds of receiving the event otherwise the procedure will time out.

- **connId**: Connection identifier. See 5.1.2.
- **oob**: Out-of-band pairing data present or not present.
- **auth**: Authentication and bonding flags. See 7.1.1.
- **iKeyDist**: Initiator key distribution flags. See 7.1.2.
- **rKeyDist**: Responder key distribution flags. See 7.1.2.

When the pairing procedure is complete the client's callback function is called with a DM_SEC_PAIR_CMPL_IND event if successful or a DM_SEC_PAIR_FAIL_IND if failure.

7.2.4 void DmSecCancelReq(dmConnId_t connId, uint8_t reason)

This function is called to cancel the pairing process.

- **connId**: Connection identifier. See 5.1.2.
- **reason**: Failure reason. See 7.1.5.

7.2.5 void DmSecAuthRsp(dmConnId_t connId, uint8_t authDataLen, uint8_t *pAuthData)

This function is called in response to a DM_SEC_AUTH_REQ_IND event to provide PIN or OOB data during pairing.

- **connId**: Connection identifier. See 5.1.2.
- **authDataLen**: Length of PIN or OOB data. Set to 3 if PIN is used or 16 if OOB data is used.
- **pAuthData**: Pointer to PIN or OOB data. If PIN is used, this points to a byte array containing a 24-bit integer in little endian format.

7.2.6 void DmSecSlaveReq(dmConnId_t connId, uint8_t auth)

This function is called by a slave device to request that the master initiates pairing or link encryption.

- **connId**: Connection identifier. See 5.1.2.
- **auth**: Authentication and bonding flags. See 7.1.1.

7.2.7 void DmSecEncryptReq(dmConnId_t connId, uint8_t secLevel, dmSecLtk_t *pLtk)

This function is called by a master device to initiate link encryption.

- **connId**: Connection identifier. See 5.1.2.
- **secLevel**: Security level of pairing when LTK was exchanged. See 7.1.4.
- **pLtk**: Pointer to LTK parameter structure.

When the encryption procedure is complete the client's callback function is called with a DM_ENCRYPT_IND event if successful or a DM_ENCRYPT_FAIL_IND if failure.

7.2.8 void DmSecLtkRsp(dmConnId_t connId, bool_t keyFound, uint8_t secLevel, uint8_t *pKey)

This function is called by a slave in response to a DM_SEC_LTK_REQ_IND event to provide the long term key used for encryption.

- **connId**: Connection identifier. See 5.1.2.
- **keyFound**: TRUE if key found.
- **secLevel**: Security level of pairing when LTK was exchanged. See 7.1.4.
- **pKey**: Pointer to the key, if found.

7.2.9 void DmSecSetLocalCsrk(uint8_t *pCsrk)

This function sets the local CSRK used by the device.

- **pCsrk**: Pointer to CSRK.

7.2.10 void DmSecSetLocalIrk(uint8_t *pIrk)

This function sets the local IRK used by the device.

- **pIrk**: Pointer to IRK.

7.2.11 void DmSecLescInit(void)

This function is called to initialize the LE Secure Connections subsystem.

7.2.12 void DmSecKeypressReq(dmConnId_t connId, uint8_t keypressType)

This function can be used to send a keypress request command to the peer device during LE Secure Connections Passkey Security.

- **ConnId**: Connection identifier. See 5.1.2.
- **KeypressType**: Type of keypress reported to peer. See 7.1.6.

7.2.13 void DmSecGenerateEccKeyReq(void);

This function is called to generate an ECC Key for use in LE Secure Connections. The application is notified of the result of the generate ECC key operation via the DM_SEC_ECC_KEY_IND event.

7.2.14 void DmSecSetEccKey(wsfSecEccKey_t *pKey);

This function is called to set the ECC key used in LE Secure Connections.

- **pKey**: Pointer to the ECC key.

7.2.15 void DmSecSetDebugEccKey(void);

This function is called to set the ECC key used in LE Secure Connections.

7.2.16 void DmSecSetOob(dmConnId_t connId, dmSecLescOobCfg_t *pConfig);

This function is called to set the Out of Band configuration containing the local and remote confirm and random values for LE Secure Connections Security.

- **ConnId**: Connection identifier. See 5.1.2.
- **pConfig**: The OOB configuration.

7.2.17 void DmSecCalcOobReq(uint8_t *pRand, uint8_t *pPubKeyX);

This function is used to calculate the local confirm value used in Out of Band LE Secure Connections Security.

- **pRand**: A 128-bit random value.
- **pPubKeyX**: The X component of the ECC public key.

7.2.18 void DmSecCompareRsp(dmConnId_t connId, bool_t valid);

This function is used to indicate the LE Secure Connections Numeric Comparison value is valid or invalid. It is typically called in response to a DM_SEC_COMPARE_IND event.

- **ConnId**: Connection identifier. See 5.1.2.
- **valid**: TRUE if the compare value is correct, else FALSE.

7.3 Callback Interface

7.3.1 DM_SEC_PAIR_CMPL_IND: Pairing Complete

Callback event for pairing complete. This event uses type dmSecPairCmplIndEvt_t.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.
uint8_t	auth	Authentication and bonding flags. See 7.1.1.

7.3.2 DM_SEC_PAIR_FAIL_IND: Pairing Failed

Callback event for pairing failed. This event uses type wsfMsgHdr_t.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.
wsfMsgHdr_t	hdr.status	Pairing failure status. See 7.1.5.

7.3.3 DM_SEC_ENCRYPT_IND: Connection Encrypted

Callback event for connection encrypted. This event uses type dmSecEncryptIndEvt_t.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.
bool_t	usingLtk	TRUE if connection encrypted with LTK.

7.3.4 DM_SEC_ENCRYPT_FAIL_IND: Encryption Failed

Callback event for encryption failed. This event uses type wsfMsgHdr_t.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.
wsfMsgHdr_t	hdr.status	Encryption failure status. See 7.1.5.

7.3.5 DM_SEC_AUTH_REQ_IND: Authentication Requested

Callback event for PIN or OOB data requested for pairing. This event uses type dmSecAuthReqIndEvt_t.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.
bool_t	oob	Out-of-band data requested.
bool_t	display	TRUE if PIN is to be displayed.

If OOB is TRUE, the client should call DmSecAuthRsp() with OOB data, if available. If display is TRUE, the client will typically generate and display a random PIN and call DmSecAuthRsp() with this PIN. If display is FALSE, the client will typically prompt the user to enter a PIN and call DmSecAuthRsp() with this PIN.

7.3.6 DM_SEC_KEY_IND: Key Data

Callback event for key data indication. This event uses data type dmSecKeyIndEvt_t.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.

wsfMsgHdr_t	hdr.param	Connection identifier.
dmSecKey_t	keyData	Key data.
uint8_t	type	Key type. See 7.1.3.
uint8_t	secLevel	Security level of pairing when key was exchanged. See 7.1.4.
uint8_t	encKeyLen	Length of encryption key used when data was transferred.

7.3.7 DM_SEC_LTK_REQ_IND: LTK Requested

Callback event for LTK requested. This event uses type wsfMsgHdr_t.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.

7.3.8 DM_SEC_PAIR_IND: Incoming Pairing Request

Callback event for incoming pairing request. This event uses type dmSecPairIndEvt_t.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.
uint8_t	auth	Authentication and bonding flags. See 7.1.1.
bool_t	oob	Out-of-band pairing data present or not present.
uint8_t	iKeyDist	Initiator key distribution flags. See 7.1.2.
uint8_t	rKeyDist	Responder key distribution flags. See 7.1.2.

7.3.9 DM_SEC_SLAVE_REQ_IND: Incoming Slave Security Request

Callback event for incoming slave security request. This event uses type dmSecPairIndEvt_t.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.param	Connection identifier.
uint8_t	auth	Authentication and bonding flags. See 7.1.1.

7.3.10 DM_SEC_CALC_OOB_IND

Callback with the result of an Out Of Band confirm calculation. This event uses type dmSecOobCalcIndEvt_t.

Type	Name	Description
uint8_t	confirm[SMP_CONFIRM_LEN]	Local confirm value.
uint8_t	random[SMP_RANDOM_LEN]	Local random value.

7.3.11 DM_SEC_ECC_KEY_IND

Callback with the result of an ECC Key generation. This event uses type wsfSecEccMsg_t.

Type	Name	Description
uint8_t	pubKey_x [WSF_ECC_KEY_LEN]	X component of the public key.
uint8_t	pubKey_y [WSF_ECC_KEY_LEN]	Y component of the public key.
uint8_t	privKey[WSF_ECC_KEY_LEN]	Private key.

7.3.12 DM_SEC_COMPARE_IND

Callback with the confirm value during Numeric Comparison LE Secure Connections pairing. This event uses type dmSecCnfIndEvt_t.

Type	Name	Description
uint8_t	confirm[SMP_CONFIRM_LEN]	Local confirm value.

7.3.13 DM_SEC_KEYPRESS_IND

Callback when peer receives a keypress command from the peer during LE Secure Connections passkey pairing. This event uses type dmSecKeypressIndEvt_t.

Type	Name	Description
uint8_t	notificationType	Type of keypress.

8 Privacy

The DM Privacy interface is used by a master or slave device for private address resolution.

8.1 Function Interface

8.1.1 DmPrivInit(void)

Initialize DM privacy module. This function is typically called once at system startup.

8.1.2 DmPrivResolveAddr(uint8_t *pAddr, uint8_t *pIrk, uint16_t param)

Resolve a private resolvable address. When complete the client's callback function is called with a DM_PRIV_RESOLVED_ADDR_IND event. The client must wait to receive this event before executing this function again.

- **pAddr**: Peer device address.
- **pIrk**: The peer's identity resolving key.
- **Param**: Client-defined parameter returned with callback event.

8.2 Callback Interface

8.2.1 DM_PRIV_RESOLVED_ADDR_IND: Private Address Resolved

Callback event for private address resolved. This event uses type wsfMsgHdr_t. If address resolution is successful hdr.status is set to HCI_SUCCESS, otherwise it is set to HCI_ERR_AUTH_FAILURE.

Type	Name	Description
wsfMsgHdr_t	hdr.event	Callback event.
wsfMsgHdr_t	hdr.status	Status.
wsfMsgHdr_t	hdr.param	Client-defined parameter passed to DmPrivResolveAddr().

9 Scenarios

9.1 Advertising and Scanning

Figure 1 shows a master device performing a scan and a slave device advertising. The slave application first configures the advertising parameters by calling DmAdvSetInterval() to set the advertising interval and then DmAdvSetData() twice to set the advertising data and the scan response data. Then it calls DmAdvStart() to start advertising.

The master application configures the scan interval and then calls DmScanStart() to begin scanning. When advertisements are received the stack sends DM_SCAN_REPORT_IND events to the application. The master application stops scanning by calling DmScanStop(). The slave application stops advertising by calling DmAdvStop().

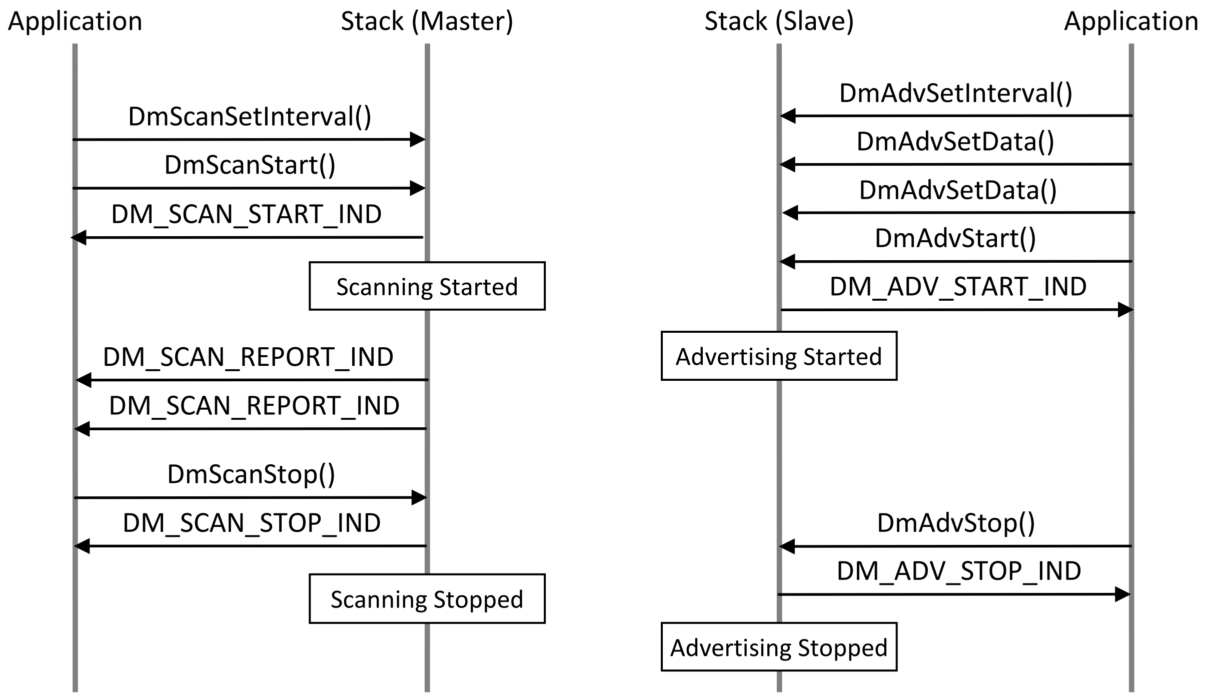

Figure 1. Advertising and scanning.

9.2 Connection Open and Close

Figure 2 shows connection procedures between two devices. The scenario starts with the slave device advertising and the master device already having the address of the slave. The master application calls DmConnOpen() to initiate a connection. A connection is established and a DM_CONN_OPEN_IND is sent to the application from the stack on each device.

Next, the master performs a connection update by calling DmConnUpdate(). When the connection update is complete a DM_CONN_UPDATE_IND is sent to the application from the stack on each device.

Next, the slave closes the connection by calling DmConnClose(). A DM_CONN_CLOSE_IND event is sent from the stack on each device when the connection is closed.

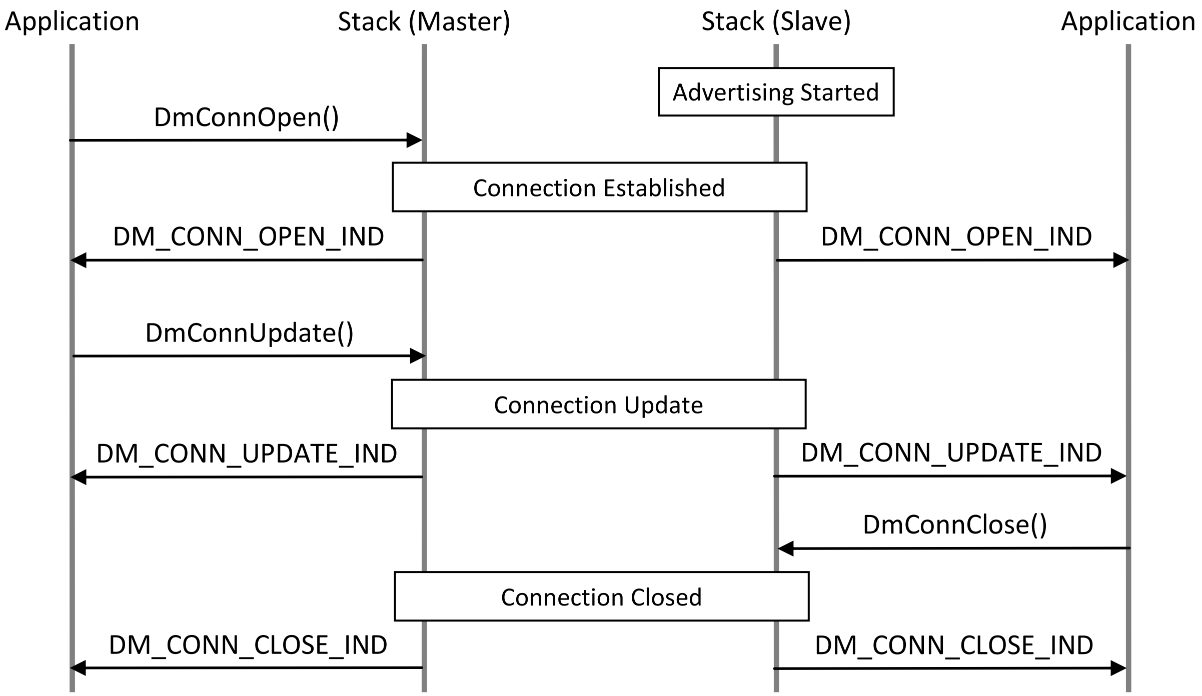

Figure 2. Connection open and close.

9.3 Pairing

Figure 3 shows a pairing procedure between two devices. A connection is established between the two devices and the master application initiates pairing by calling DmSecPairReq(). The slave application receives a DM_SEC_PAIR_IND and calls DmSecPairRsp() to proceed with pairing. In this example a PIN is used and a DM_SEC_AUTH_REQ_IND is sent to the application on each device to request a PIN. Each application responds with the PIN by calling DmSecAuthRsp().

In the next phase of pairing the connection is encrypted and a DM_SEC_ENCRYPT_IND event is sent to the application on each device. Then key exchange begins. According to the Bluetooth specification, the slave device always distributes keys first. In this example, the slave distributes two keys and the master device distributes one. The slave sends its key data to the master. Note that when the slave sends its LTK, the slave application receives a DM_SEC_KEY_IND containing its own LTK. Then the master sends its key data to the slave. When the key exchange is completed successfully, a DM_SEC_PAIR_CMPL_IND event is sent to the application on each device.

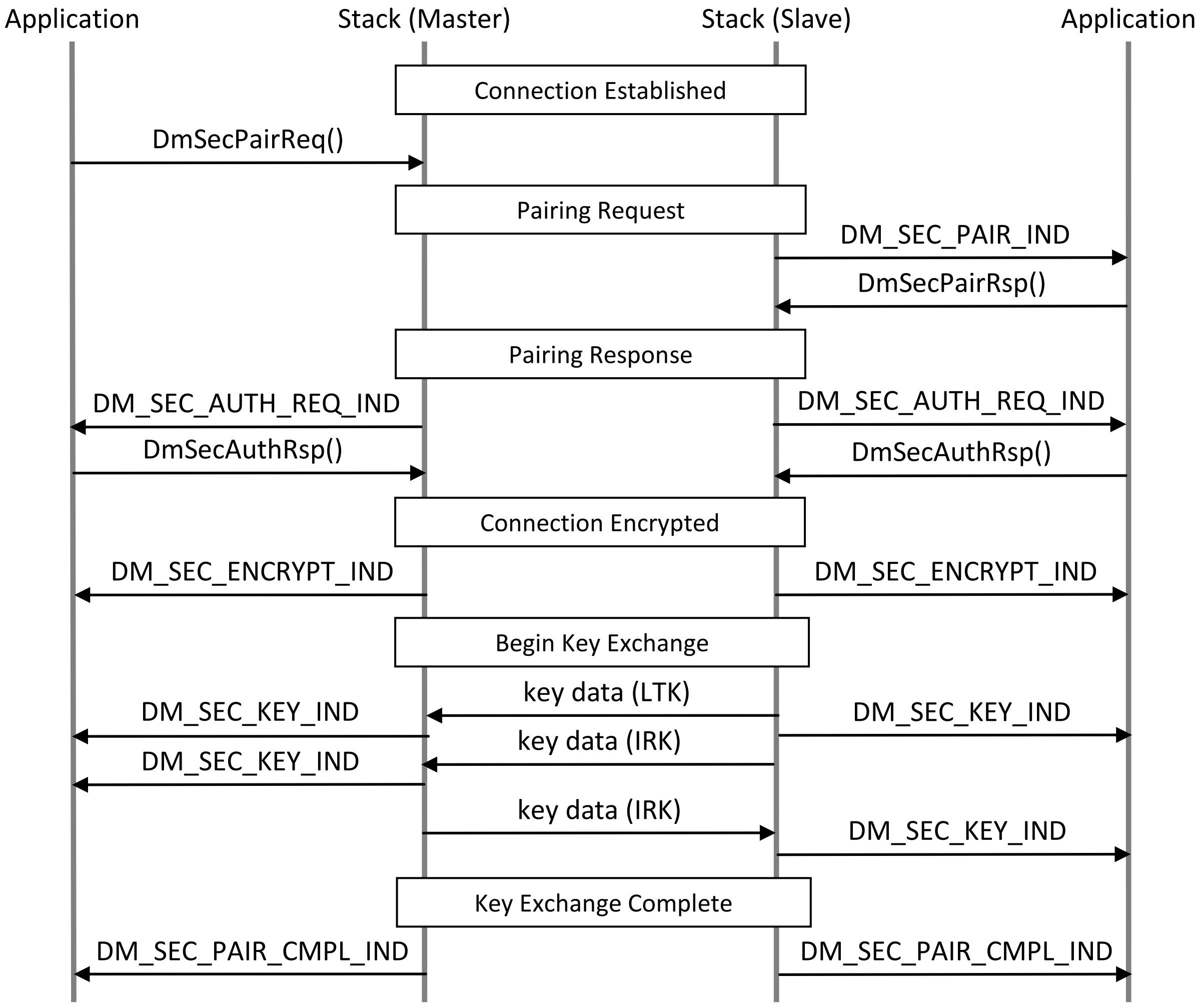

Figure 3. Pairing

9.4 Encryption

Figure 4 shows an encryption procedure. In this example the slave device requests security by calling DmSecSlaveReq() to sends a slave security request message to the master. The stack on the master sends a DM_SEC_SLAVE_REQ_IND to the application. Upon receiving the event the master application determines that this is a bonded device and its LTK is available, so it calls DmSecEncryptReq() to enable encryption.

After the encryption procedure is initiated the slave application receives a DM_SEC_LTK_REQ_IND, requesting the LTK used with this master device. The application finds the key and calls DmSecLtkRsp(). The encryption procedure completes and a DM_SEC_ENCRYPT_IND event is sent to the application on each device.

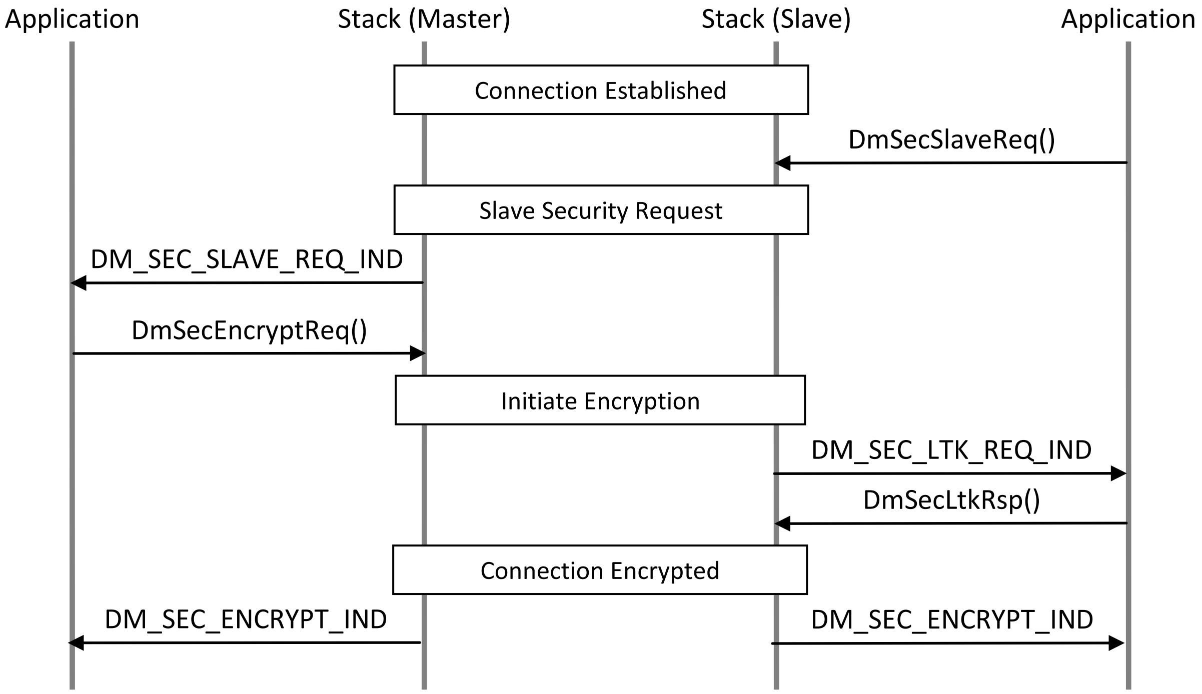

Figure 4. Encryption

9.5 Privacy

Figure 5 shows a master device performing a scan and a slave device advertising with a private resolvable address. Before a master device can resolve a slave's address the devices must have paired and the master must have received the slave's IRK during pairing.

The slave application first enables use of a private resolvable address by calling DmAdvPrivStart(). If this is the first time since device reset that DmAdvPrivStart() has been called, the application must wait for a DM_ADV_NEW_ADDR_IND before it starts advertising. Then it calls DmAdvStart() to start advertising.

The master application calls DmScanStart() to begin scanning. When advertisements are received the stack sends DM_SCAN_REPORT_IND events to the application. The master application calls DmPrivResolveAddr() with the address and address type from the scan report to resolve the address with the IRK it had received previously.

After the slave application stops advertising it may call DmAdvPrivStop() to stop using a private resolvable address.

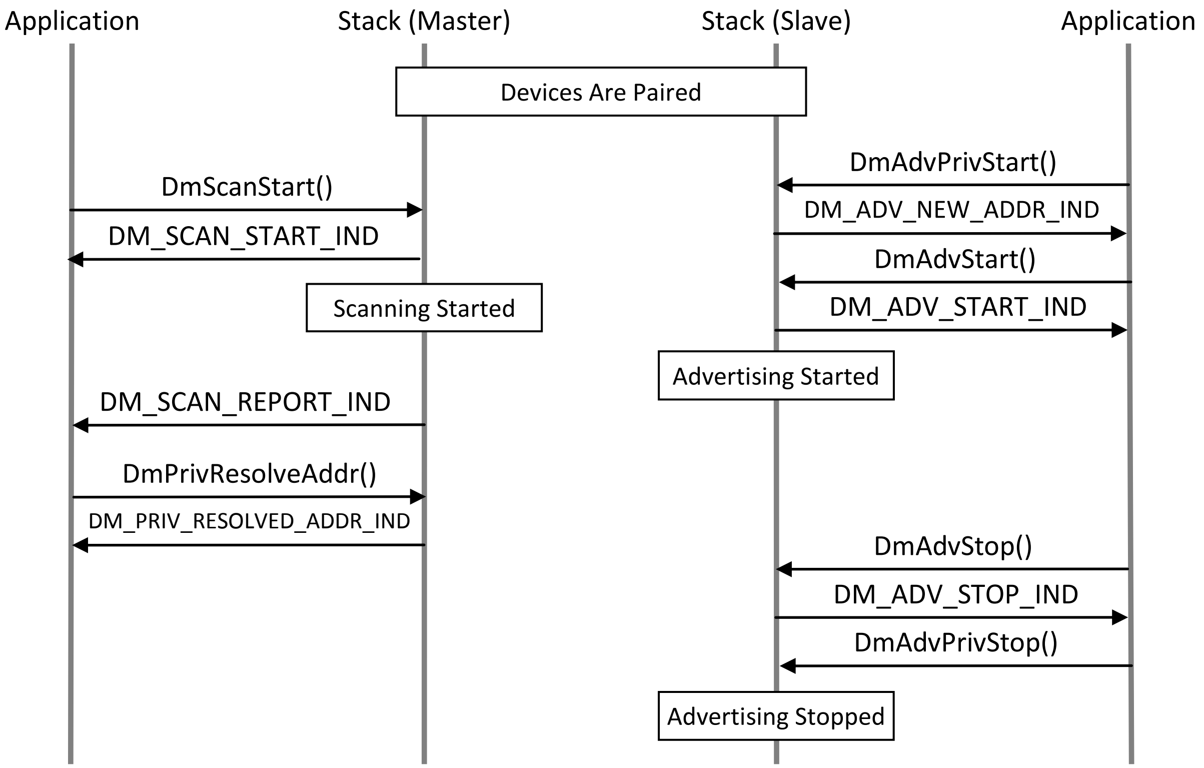

Figure 5. Privacy

9.6 ECC Key Generation

An ECC Key must be stored in the Device Manager prior to use of LE Secure Connections pairing. The Device Manager can generate an ECC, Elliptic Curve Cryptography, key, or the application can store an ECC Key in Non-Volatile storage. An ECC key cannot be generated until after the Device Manager reset is complete.

To generate an ECC Key, call the DmSecGenerateEccKeyReq function after receiving the DM_RESET_CMPL_IND event. The DM_SEC_ECC_KEY_IND event will be called after the ECC Key generation is complete. The ECC Key can then be stored into the DM using the DmSecSetEccKey function.

Note: For some applications, it may be desirable to skip ECC Key Generation and store an ECC key in Non Volatile storage. In these situations, the ECC key can be written to the Device Manager with DmSecSetEccKey any time after the DM is reset, and before pairing begins.

Note: The Device Manager makes use of the WSF ECC subsystem to generate and validate ECC keys. The WSF ECC subsystem may need to be ported to an application's target hardware or software framework for LE Secure Connections to operate properly.

The following figure shows the ECC Key generation scenario:

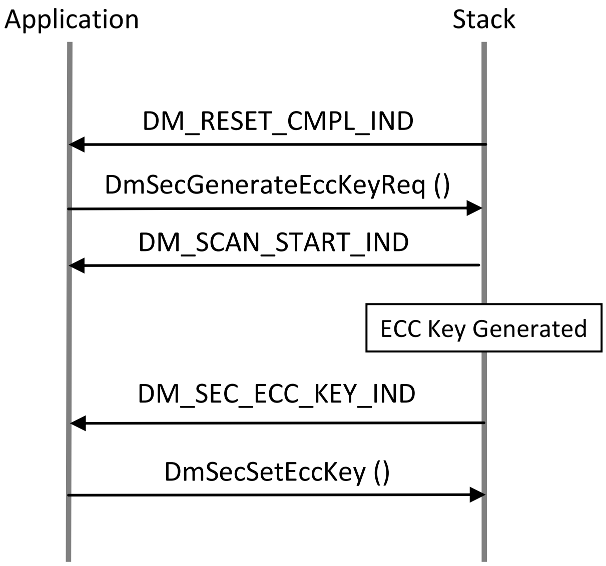

Figure 6: ECC Key Generation

9.7 Out of Band Confirm Calculation

When using Out-of-Band (OOB) LE Secure Connections pairing, devices must generate random and confirm values. Furthermore, the devices must exchange random and confirm values through an out-of-band mechanism. At which point, the local and peer random and confirm values must be stored in the Device Manager prior to OOB pairing.

The OOB confirm calculation can be performed with DmSecCalcOobReq, and requires an ECC, Elliptic Curve Cryptography, key. Therefore, on receipt of the ECC key indication event, DM_SEC_ECC_KEY_IND, an application may call the DmSecCalcOobReq function to calculate an OOB confirm value. The result of the confirm calculation will be returned via the DM_SEC_CALC_OOB_IND event.

After an application exchanges random and confirm values via an out-of-band mechanism with a peer, the application must store the local random and confirm values in the device manager. This can be performed with the DmSecSetOob function. This must happen prior to initiating LE Secure Connections OOB Pairing.

The following figure shows the OOB confirm calculation scenario:

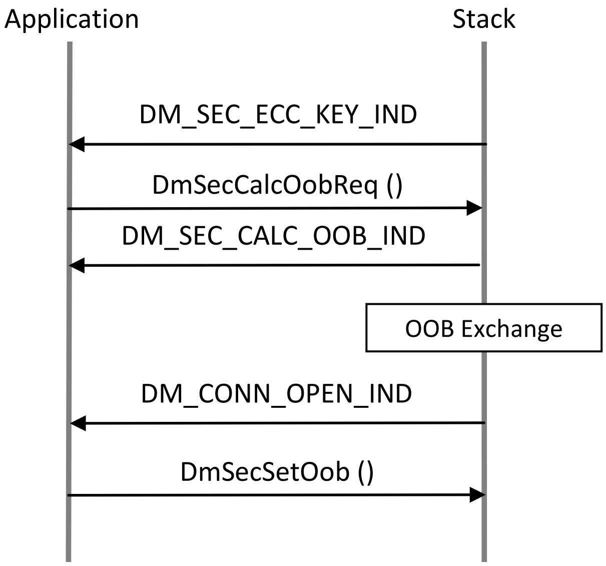

Figure 7: OOB Confirm Generation

10 References

1. Wicentric, “HCI API”, 2009-0006.
2. Wicentric, “Software Foundation API”, 2009-0003.
3. Bluetooth SIG, “Specification of the Bluetooth System”, Version 4.2, December 2, 2015.

11 Definitions

ACL	Asynchronous Connectionless data packet
CID	Connection Identifier
CSRK	Connection Signature Resolving Key
DM	Device Manager software subsystem
HCI	Host Controller Interface
IRK	Identity Resolving Key
L2C	L2CAP software subsystem
L2CAP	Logical Link Control Adaptation Protocol
LE	(Bluetooth) Low Energy
LTK	Long Term Key
MITM	Man In The Middle pairing (authenticated pairing)
OOB	Out Of Band data
SMP	Security Manager Protocol
STK	Short Term Key
WSF	Wicentric Software Foundation software service and porting layer