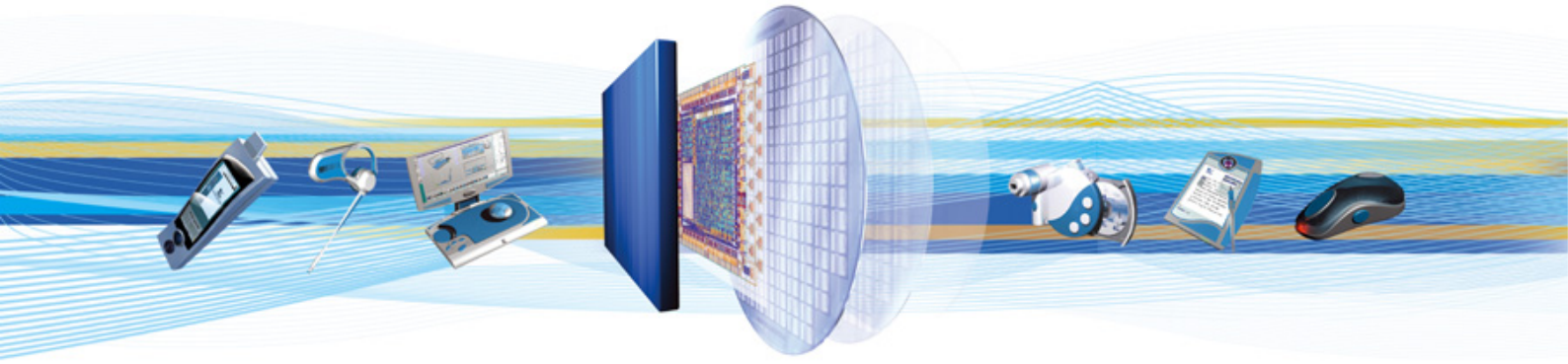
CSR Synergy Bluetooth 18.2.2

SC – Security Controller

API Description

January 2012



Cambridge Silicon Radio Limited

Churchill House
Cambridge Business Park
Cowley Road
Cambridge CB4 0WZ
United Kingdom

Registered in England and Wales 3665875

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001
www.csr.com

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction and Scope

This document describes the functionality and message interface provided by the Security Controller (SC) in CSR Synergy Bluetooth. The Security Controller is responsible for all security related functions between the user application and the CSR Synergy Bluetooth.

1.2 Assumptions

The following assumptions and preconditions are made in the following:

- There is a secure and reliable transport between the profile part, i.e. the Security Controller and the application, for communication between the components
- The application must implement Non-Volatile Storage possibilities according to the interface described in [PORT]

2 Description

2.1 Introduction

In many applications integrity and authenticity is of paramount importance. Bluetooth® provides for security and authenticity through a set of security procedures defined in the Bluetooth® specification [BT]. The specification deals with procedures for device to device security on a link level and defines how the authentication procedure for device authenticity and the encryption procedure for link encryption must be implemented. The security features are based on a secret link key that is shared between the pair of devices.

The link manager security features can be used in different security modes. These are defined in the Bluetooth® Generic Access Profile [GAP]. Four security modes are defined:

- Security mode 1 (non-secure), i.e. a device will not initiate any security procedure
- Security mode 2 (service-level enforced security) where a device does not initiate security procedures before channel establishment at L2CAP level. This mode allows different and flexible access policies for applications, especially running applications with different security requirements in parallel
- Security mode 3 (link level enforced security). In mode 3 a device initiates security procedures before the link set-up at the link manager level is completed
- Security mode 4 (service-level enforced security) which is similar to mode 2 but adds more strength to security by using another set of authentication procedures, Secure Simple Pairing (SSP).

Security mode 1, 2 and 3 are known as legacy pairing.

Especially it is possible to let the profile managers apply the needed level of security when in security mode 2/4. The Bluetooth® SIG has defined a white paper [SEC] for how to apply these basic procedures to achieve security on a Bluetooth® service level. Through the Security Controller it is possible to utilize all the different security modes.

The Security Controller provides functionality for:

- Creating a bond between a set of devices
- Storing and retrieval of the bond (the link key) between the devices for subsequent authentication and encryption
- Changing encryption mode of a connect
- Defining the device security level
- Handling of service security registration and subsequent activation on request from the profile managers and protocol stack

It must be noted that it is possible to complement the Bluetooth® security with application specific security procedures. These may depend on the actual application being applied on top of the Bluetooth® connection between the devices. Examples of application level security are PPP and IPSEC.

2.2 Reference Model

The security layer is designed to handle as much of the security functionality autonomously as possible. This implies that the interface between the application layer and the Security Controller is very simple. The application layer must interact with the SC for functions that normally require user interaction, e.g. inserting a passkey.

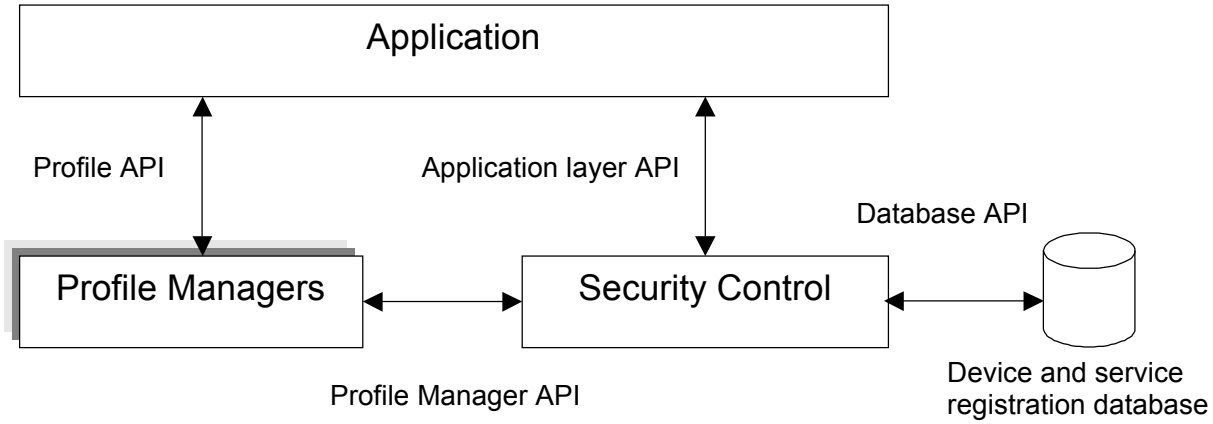


Figure 1: Security Controller model

The security layer defines three interfaces. One is an interface that can be applied by the application layer. The application layer interface is mainly for functions that require user interaction. In addition to the application layer interface also an internal interface intended for use by the profile managers is defined. The internal interface is utilized by the profile managers for registration of their specific service; the internal interface is not available for application layers.

The security components will handle all subsequent access control according to the registration. The last interface is between the Security Controller and a database. The database provides the possibility for storing and retrieving link keys in a non-volatile storage (NVS).

2.3 Sequence Overview

The security layer in CSR Synergy Bluetooth implements a security model, where devices are assigned a trust level and where services are defined to require a certain trust level. The trust level of a device can be changed when bonding/de-bonding is performed. The trust level required by a service is defined in the respective profiles by the SIG.

The normal scenario for use of the Security Controller starts when the application decides that bonding with a remote device should be started. Using the defined API, the application will initiate the bonding procedure and respond to the subsequent passkey request from the Security Controller. The Security Controller will handle all inter-working with the remote device and storing of the bond in the registration database.

Upon registering a profile service, the corresponding security settings are defined for this particular profile service. The individual profile layer handles registering for the service toward the Security Controller.

3 Interface Description

3.1 Application Layer API

3.1.1 Registration of application layer handle

The application layer handle registration process is initiated by sending a CSR_BT_SC_ACTIVATE_REQ message from the application layer to the SC layer. The message is used for registering a default task handle for the application layer in the SC layer. If a default application handle is already defined in the csr_bt_usr_config.h file, the value received in the CSR_BT_SC_ACTIVATE_REQ overwrites the default value from the csr_bt_usr_config.h file.

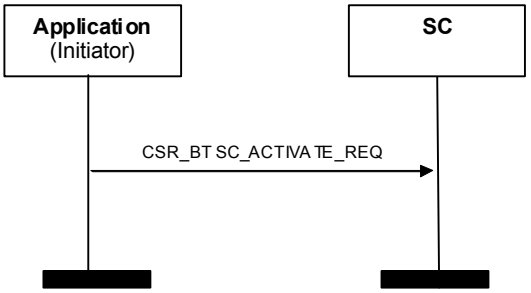


Figure 2: Default application layer handle registration sequence

Please note that the default application handle is only used for unsolicited messages/sequences. I.e. if the application layer has initiated a sequence for which a response is expected, an application layer handle is included in the request signal and the response is returned to this address.

3.1.2 Secure Simple Pairing

With Bluetooth 2.1 pairing/bonding has become easier, more secure and flexible with regards to a device's IO capabilities. For more information about SPP, please refer to [UIF].

To use Secure Simple Pairing (SSP) the device must enter security mode 4. This can be done at runtime using SC_SET_SEC_MODE_REQ or at compile time in csr_bt_usr_config.h by setting the DEFAULT_SEC_MODE define to SEC_MODE4_SSP which is the default setting. Security mode 4 is backwards compatible with devices supporting only Bluetooth 2.0 or older and it is highly recommended that mode 4 is used. If the local device is set in security mode 4 and the peer device does not support this mode the security controller will automatically fall back and use legacy bonding. In section 2.1 the security modes available by legacy bonding are defined. After entering security mode 4, all attempts to change the security mode are rejected. This is done in order to avoid interoperability problems.

In the Bluetooth v.2.1 Core specification the term 'Bonding' is defined as the connection process where the link key and connection information are stored in non-volatile memory. Bonding implies that the device's link key information is available after the connection is released. If the link key is not stored, then the connection process uses 'pairing'. In CSR Synergy BT this corresponds to the fact that the security is being started implicit, when an incoming or outgoing connection is being setup. In the MCS shown in Figure 3, Figure 4, Figure 5, Figure 6, Figure 7 and Figure 9 this is shown as the Application Responder. In the Bluetooth v.2.1 Core specification this is also known as General Bonding.

The Bluetooth v.2.1 Core specification also uses the term 'Dedicated Bonding'. Where Dedicated Bonding in general is associated with the 'Add device' user action where no particular service is being selected prior to initiating the connection. In CSR Synergy BT this corresponds to sending the SC_BOND_REQ message. In the MCS illustrated in Figure 3, Figure 4, Figure 5, Figure 6, Figure 7 and Figure 9 this is shown as the Application Initiator.

The parameters used for SSP must be set according to a device's input/output capability. Table 1 shows how to map input/output capability to the required format. The default IO capability can be set in csr_bt_usr_config.h by

setting the DEFAULT_IO_CAPABILITY define to one of the values defined in Table 1. The default setting is HCI_IO_CAP_DISPLAY_YES_NO as this setting represents the standard capabilities of a mobile handset.

Output capability / **Input capability**	No output	Numeric Output
No input	HCI_IO_CAP_NO_INPUT_NO_OUTPUT	HCI_IO_CAP_DISPLAY_ONLY
Yes / No	HCI_IO_CAP_NO_INPUT_NO_OUTPUT	HCI_IO_CAP_DISPLAY_YES_NO
Keyboard	HCI_IO_CAP_KEYBOARD_ONLY	HCI_IO_CAP_DISPLAY_YES_NO

Table 1: IO Capability Mapping

Compared to bonding/pairing in Bluetooth 2.0, SSP uses different authentication models to ensure that a high level of security is obtained by considering the involved parties IO capabilities. The exact SSP authentication model used depends on both the local and remote device's IO capability. Table 2 reflects which model will be used if and only if one of the devices requires MITM (Man-In-The-Middle) protection. If no 'A' or 'B' prefix is presented, both sides will use the authentication model illustrated in Table 2. Please note that requiring MITM protection should be a rare thing. MITM is provided for cases where explicit user authentication is truly needed, for example by profiles such as the Human Interface Device Profile (HID) or the SIM Access Profile (SAP) that allow access to personal and/or mission critical data. Please also note that each profile in CSR Synergy BT has an API for setting its security requirements, and it is possible to demand MITM on HID and no MITM on OBEX FTP. In cases like this, where one profile demands MITM and another one don't, the Security Manager will always try to use MITM. If MITM fails and the pairing attempt were made to a profile, which did not required MITM, the Security Manager will automatic fallback and use JustWorks instead. Opposite, if MITM fails and the pairing attempt were made to a profile which demand MITM the Security Manager will NOT fallback to JustWorks.

The default security setting for the profiles is defined in csr_bt_usr_config.h. When performing a Dedicated Bonding, by issuing a SC_BOND_REQ message, the MITM protection requirements can be set by the DEFAULT_AUTH_REQUIREMENTS, which is defined in csr_bt_usr_config.h or at runtime using the SC_SET_AUTH_REQUIREMENTS_REQ message, see section 4.17.

The valid values are listed in Table 3. It is strongly recommended that MITM is not required doing a Dedicated Bonding, because if a profile like SAP demands MITM and a Dedicated Bonding has been performed with success, the security will be raised automatically, by starting a new bonding/pairing procedure, whenever a connection is established to this profile. Also note that if the local device demands MITM and the peer device does not, then a connection cannot be established, until both either use MITM or not. Therefore DEFAULT_AUTH_REQUIREMENTS is also per default set to HCI_MITM_NOT_REQUIRED_DEDICATED_BONDING. Please note that the security settings only indicate the local device's intent to the remote device. If the local device wishes to do Dedicated Bonding, the remote device may delete the link key after authentication/pairing has completed.

Device A / **Device B**	DisplayOnly	DisplayYesNo	KeyboardOnly	NoInputNoOutput
DisplayOnly	Just Works	Just Works	A: Passkey B: Notification	Just Works
DisplayYesNo	Just Works	Numeric Comparison	A: Passkey B: Notification	Just Works
KeyboardOnly	A: Notification	A: Notification	Passkey	Just Works

	B: Passkey	B: Passkey		
NoInputNoOutput	Just Works	Just Works	Just Works	Just Works

Table 2: Authentication Model

In the case where none of the parties involved wishes to have MITM (Man-In-The-Middle) protection a less secure authentication procedure is used. This model is referred to as Just Works, which does not require any input from the user. The application may simply accept the bonding/pairing request on behalf of the user without passing it on to the MMI or the application can present a dialog box asking the user to add the device to the trusted device list or not. If both the local and the peer device is using security mode 4 with no MITM at least the Just Works model will be used, where as for legacy bonding it is possible to switch all security off depending on the actual needs. This means that even for transferring e.g. a vCard, bonding/pairing will still have to take place between two BT2.1 devices. This is done using the Just Works mechanism. In BT2.0 or older it was possible to disable security for such a use case, but in BT2.1 the specification mandates that at least the Just Works mechanism must take place in this scenario as well.

HCI_MITM_NOT_REQUIRED_NO_BONDING	Deprecated shall not be used. It is handled by the security controller itself.
HCI_MITM_REQUIRED_NO_BONDING	Deprecated shall not be used. It is handled by the security controller itself.
HCI_MITM_NOT_REQUIRED_DEDICATED_BONDING	MITM not required when the application wishes to do a Dedicated Bonding, by sending a SC_BOND_REQ message. (Recommended setting)
HCI_MITM_REQUIRED_DEDICATED_BONDING	MITM required when the application wishes to do a Dedicated Bonding, by sending a SC_BOND_REQ message.
HCI_MITM_NOT_REQUIRED_GENERAL_BONDING	Deprecated shall not be used. It is handled by the security controller itself.
HCI_MITM_REQUIRED_GENERAL_BONDING	Deprecated shall not be used. It is handled by the security controller itself.

Table 3: Authentication Requirements

3.1.3 Link Keys

After pairing/bonding has successfully completed a link key is generated and can be used for subsequent authentication requests and the application will not receive any CSR_BT_SC_PASSKEY_IND, CSR_BT_SC_SSP_JUSTWORKS_IND, CSR_BT_SC_SSP_COMPARE_IND, CSR_BT_SC_SSP_PASSKEY_IND or CSR_BT_SC_SSP_NOTIFICATION_ID for the remote device as long as the remote device also retains the ink key. The resulting link key from the bonding procedure can be categorised according to the following:

- Combination link key (occurs when application responds to a CSR_BT_SC_PASSKEY)
- Unauthenticated combination link key (occurs when application responds to a CSR_BT_SC_SSP_JUSTWORKS)
- Authenticated combination link key (occurs when application responds to a CSR_BT_SC_SSP_COMPARE/ CSR_BT_SC_SSP_PASSKEY/ CSR_BT_SC_SSP_NOTIFICATION)

The authenticated and unauthenticated combination link key is only generated when the local and remote device are in security mode 4. The procedure used for generating these link keys are considered more secure than legacy pairing (combination link key). Note that the unauthenticated combination link key does not provide MITM protection.

3.1.4 Bonding

The bonding procedure is used when a link is established specifically for carrying out the pairing procedure. Before any security features, e.g. encryption, can be utilised, a bond between the two devices must have been created and link keys exchanged; the dedicated bonding procedure can be used for this. During bonding the application must handle one of the following primitives and act accordingly:

CSR_BT_SC_PASSKEY_IND (see Section 3.1.5)

CSR_BT_SC_SSP_PASSKEY_IND (see Section 3.1.6)

CSR_BT_SC_SSP_NOTIFICATION_IND (see Section 3.1.7)

CSR_BT_SC_SSP_COMPARE_IND (see Section 3.1.8)

CSR_BT_SC_SSP_JUSTWORKS_IND (see Section 3.1.9)

After bonding has completed the initiator will receive a CSR_BT_SC_BOND_CFM and the responder a CSR_BT_SC_BOND_IND. It should be noted that a CSR_BT_SC_BOND_CFM can be received at any time after bonding has been initiated – even while a user is entering the passkey. Both primitives will contain a result parameter indicating whether bonding was successful, failed or was cancelled.

If bonding succeeds, the ACL connection will be kept open for an amount of time defined by CSR_BT_DM_CONFIG_ACL_IDLE_TIMEOUT_BONDING in csr_bt_usr_config.h. Note that CSR_BT_DM_CONFIG_ACL_IDLE_TIMEOUT_BONDING specifies the time in quarters (1/4) of a second.

If bonding fails the CSR_BT_SC_BOND_IND is received with an error code; in the failure case the CSR_BT_SC_BOND_IND is generated based upon a supervision timer, CSR_BT_SC_INTERNAL_SUPERVISION_TIMER, defined in csr_bt_sc_main.h. This is the reason that the responding side might be slower to react/detect a failed bonding request.

Note: The application may issue one bond request per device (in other words, it can have multiple outstanding requests) but it shall not assume that the requests will be processed in that order.

3.1.5 Legacy Passkey Bonding

The procedure is started when the application layer sends a CSR_BT_SC_BOND_REQ to the SC. The bonding sequence is outlined in Figure 3 for the CSR_BT_SC_PASSKEY case.

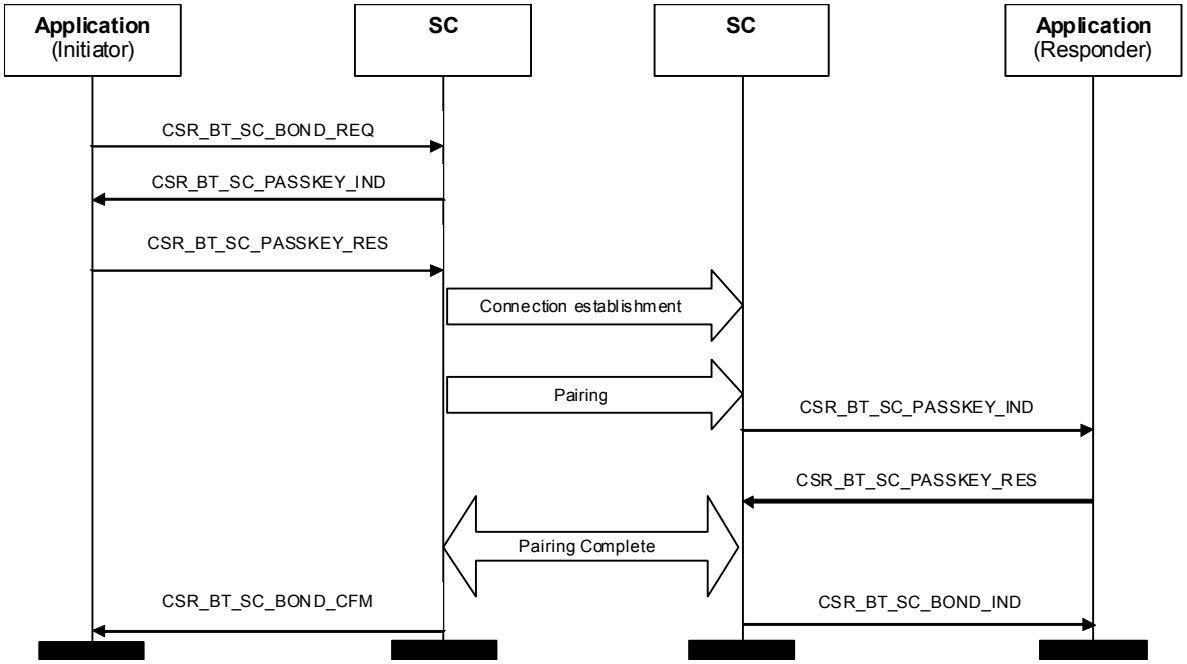


Figure 3: Legacy Passkey bonding sequence

The SC will remove any exiting bonds between the two devices before commencing the bonding procedure. The application layer must specify the Bluetooth® device address of the device to bond with.

The SC will request the Bluetooth® passkey from the application layer. The application may insert the passkey directly, e.g. in a headset application, or may request the user through the MMI to insert the right passkey.

A CSR_BT_SC_BOND_CFM is returned to the application on the initiating side as a confirmation that the bonding is completed. The bonding may fail if e.g. a wrong passkey is entered or if it is not possible to establish a connection toward the remote device. If it is requested, in the CSR_BT_SC_PASSKEY_RES message, the SC will store a copy of the just generated link key in NVS. On the responding side no response is sent to the application but in case of successful pairing the security database is invoked for a write request of a new link key.

On the responding side a CSR_BT_SC_PASSKEY_IND is received. This signal must be responded with a CSR_BT_SC_PASSKEY_RES including the passkey or optionally a reject if bonding is not permitted.

If a CSR_BT_SC_PASSKEY_IND is not responded within a window defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT in csr_bt_usr_config.h the SC will interpret it as if the a CSR_BT_SC_PASSKEY_RES where the bonding procedure was denied and it will then handle accordingly to this by rejecting the pin request and notify the application that the bonding procedure has failed. Please be aware that the timeout value defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT.

3.1.6 SSP Passkey Bonding

The bonding procedure is started when the application layer sends a CSR_BT_SC_BOND_REQ to the SC. The bonding sequence is outlined in Figure 4 for the CSR_BT_SC_SSP_PASSKEY case.

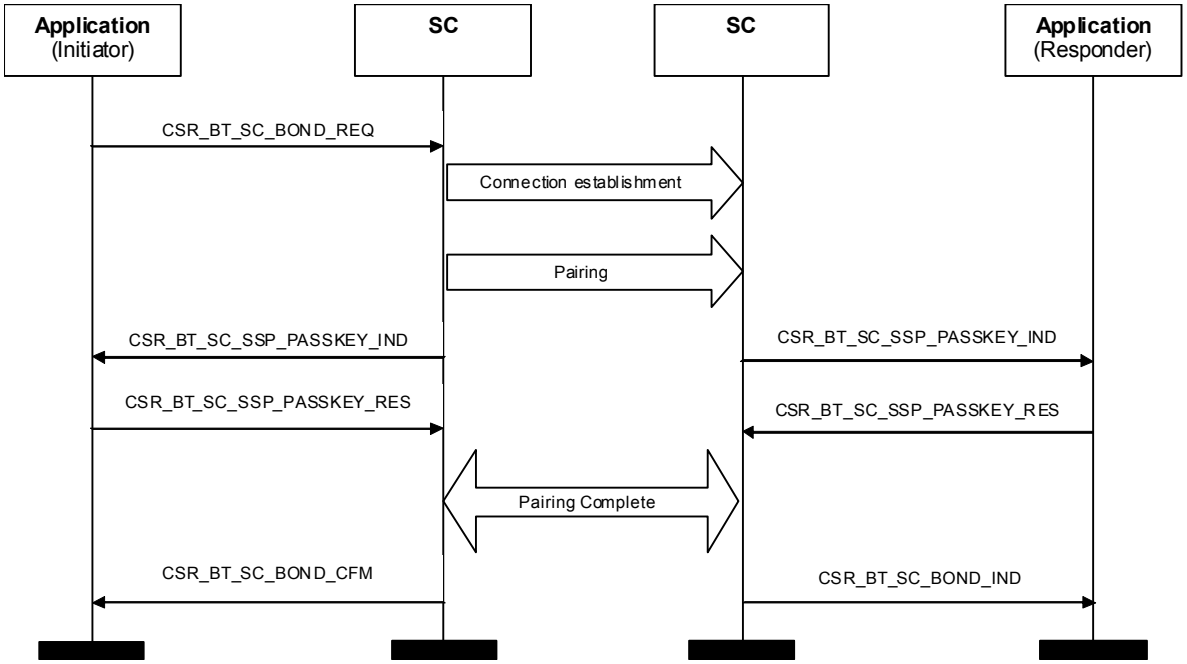


Figure 4: SSP Passkey bonding sequence

The SC will remove any exiting bonds between the two devices before commencing the bonding procedure. The application layer must specify the Bluetooth® device address of the device to bond with.

The SC will request the SSP passkey from the application layer. The application may insert the passkey directly, e.g. in a headset application, or may request the user through the MMI to insert the right passkey.

A CSR_BT_SC_BOND_CFM is returned to the application on the initiating side as a confirmation that the bonding is completed. The bonding may fail if e.g. a wrong passkey is entered or if it is not possible to establish a connection toward the remote device. If it is requested, in the CSR_BT_SC_SSP_PASSKEY_RES message, the SC will store a copy of the just generated link key in NVS. On the responding side no response is sent to the application but in case of successful pairing the security database is invoked for a write request of a new link key.

On the responding side a CSR_BT_SC_SSP_PASSKEY_IND is received. This signal must be responded with a CSR_BT_SC_SSP_PASSKEY_RES including the passkey or optionally a reject if bonding is not permitted.

If a CSR_BT_SC_SSP_PASSKEY_IND is not responded within a window defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT in csr_bt_usr_config.h. the SC will interpret it as if the a CSR_BT_SC_SSP_PASSKEY_RES where the bonding procedure was denied and it will then handle accordingly to this by rejecting the pin request and notify the application that the bonding procedure has failed. Please be aware that the timeout value is defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT.

3.1.7 SSP Notification Bonding

The bonding procedure is started when the application layer sends a CSR_BT_SC_BOND_REQ to the SC. The bonding sequence is outlined in Figure 5 for the CSR_BT_SC_SSP_PASSKEY/ CSR_BT_SC_SSP_NOTIFICATION case.

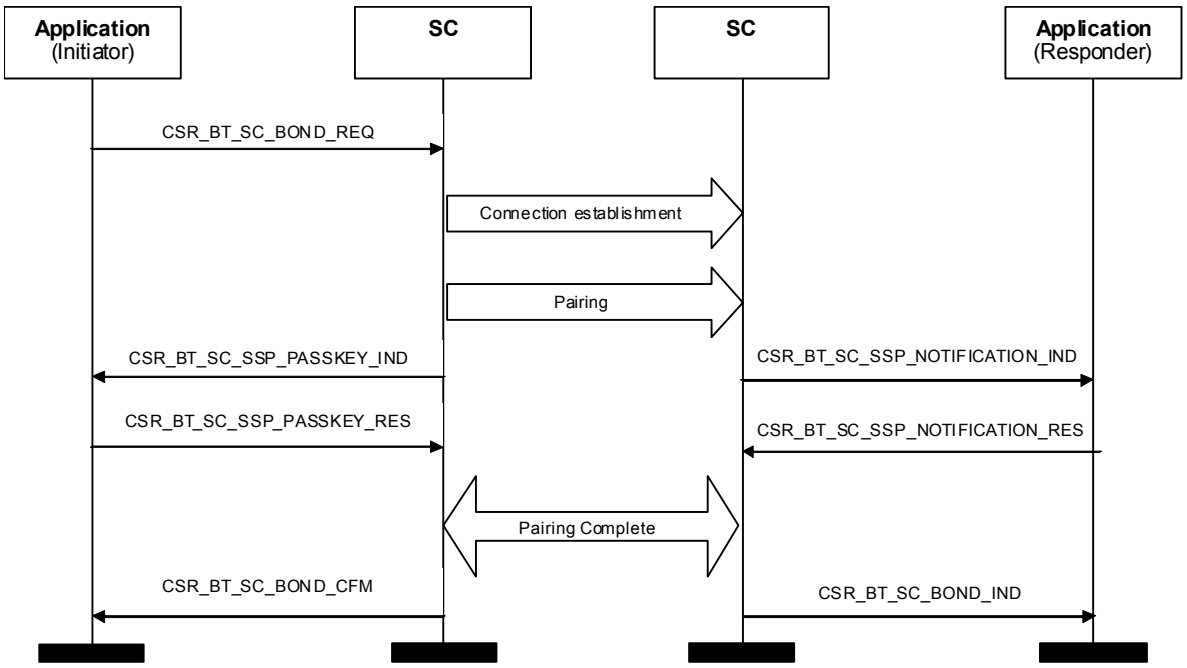


Figure 5: SSP Notification bonding sequence

The SC will remove any exiting bonds between the two devices before commencing the bonding procedure. The application layer must specify the Bluetooth® device address of the device to bond with.

The SSP Notification authentication model is asymmetric; one side will receive a CSR_BT_SC_SSP_PASSKEY_IND and the other side a CSR_BT_SC_SSP_NOTIFICATION_IND. If the application receives a CSR_BT_SC_SSP_PASSKEY_IND it must request the user through the MMI to insert the right passkey. The passkey to be entered will be shown on the other side.

A CSR_BT_SC_BOND_CFM is returned to the application on the initiating side as a confirmation that the bonding is completed. The bonding may fail if e.g. a wrong passkey is entered or if it is not possible to establish a connection toward the remote device. If it is requested, in the CSR_BT_SC_SSP_PASSKEY_RES message, the SC will store a copy of the just generated link key in NVS. On the responding side no response is sent to the application but in case of successful pairing the security database is invoked for a write request of a new link key.

On the responding side a CSR_BT_SC_SSP_NOTIFICATION_IND is received. This signal must be responded with a CSR_BT_SC_SSP_NOTIFICATION_RES or optionally a reject if bonding is not permitted. The indication contains a passkey that must be shown on the display so that the remote side will be able to enter the same passkey in the CSR_BT_SC_SSP_PASSKEY_RES.

If a CSR_BT_SC_SSP_NOTIFICATION_IND is not responded within a window defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT in csr_bt_usr_config.h, the SC will interpret it as if the a CSR_BT_SC_SSP_NOTIFICATION_RES where the bonding procedure was denied and it will then handle accordingly to this by rejecting the pin request and notify the application that the bonding procedure has failed. Please be aware that the timeout value is defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT.

3.1.8 SSP Compare Bonding

The bonding procedure is started when the application layer sends a CSR_BT_SC_BOND_REQ to the SC. The bonding sequence is outlined in Figure 6 for the CSR_BT_SC_SSP_COMPARE case.

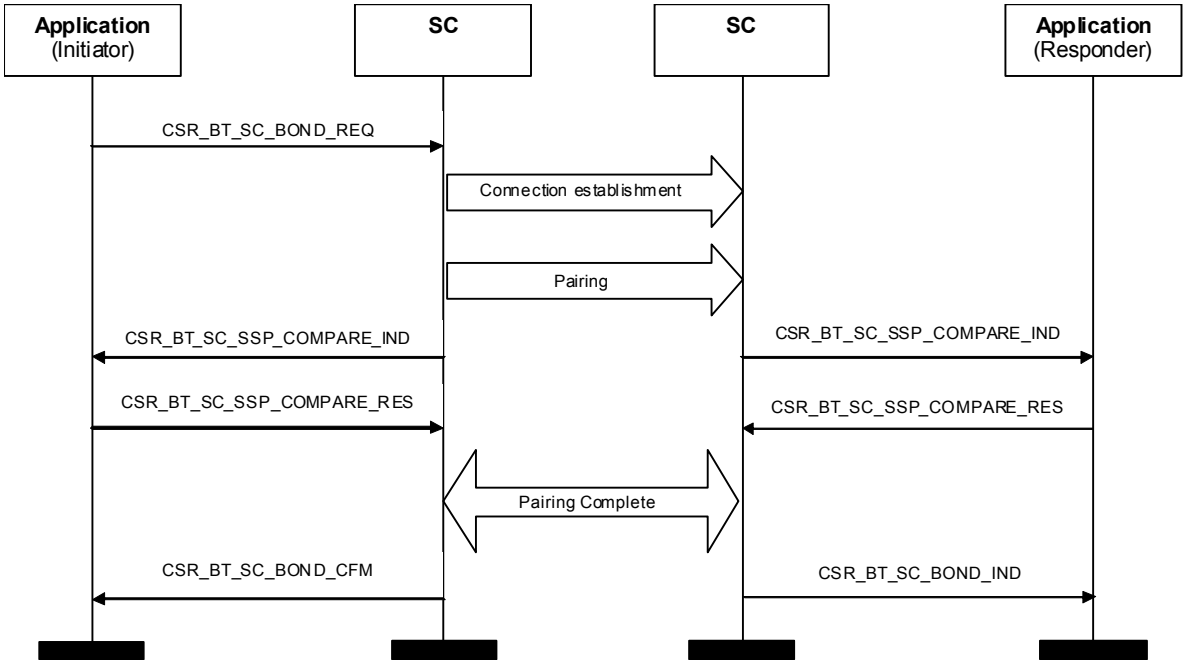


Figure 6: SSP Compare bonding sequence

The SC will remove any exiting bonds between the two devices before commencing the bonding procedure. The application layer must specify the Bluetooth® device address of the device to bond with.

The SC will request the application layer to compare the numeric value in the CSR_BT_SC_SSP_COMPARE_IND with the value shown on the remote device's display. The application may reject or accept this comparison according to the value shown on the remote device.

A CSR_BT_SC_BOND_CFM is returned to the application on the initiating side as a confirmation that the bonding is completed. The bonding may fail if e.g. the numeric values did not match or if it is not possible to establish a connection toward the remote device. If it is requested, in the CSR_BT_SC_SSP_COMPARE_RES message, the SC will store a copy of the just generated link key in NVS. On the responding side no response is sent to the application but in case of successful pairing the security database is invoked for a write request of a new link key.

On the responding side a CSR_BT_SC_SSP_COMPARE_IND is received. This signal must be responded with a CSR_BT_SC_SSP_COMPARE_RES or optionally a reject if bonding is not permitted. The numeric in the CSR_BT_SC_SSP_COMPARE_IND must be compared with the value shown on the initiating side display.

If a CSR_BT_SC_SSP_COMPARE_IND is not responded within a window defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT in csr_bt_usr_config.h, the SC will interpret it as if the a CSR_BT_SC_SSP_COMPARE_RES where the bonding procedure was denied and it will then handle accordingly to this by rejecting the pin request and notify the application that the bonding procedure has failed. Please be aware that the timeout value is defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT.

3.1.9 SSP Just Works Bonding

The bonding procedure is started when the application layer sends a CSR_BT_SC_BOND_REQ to the SC. The bonding sequence is outlined in Figure 7 for the CSR_BT_SC_SSP_JUSTWORKS case.

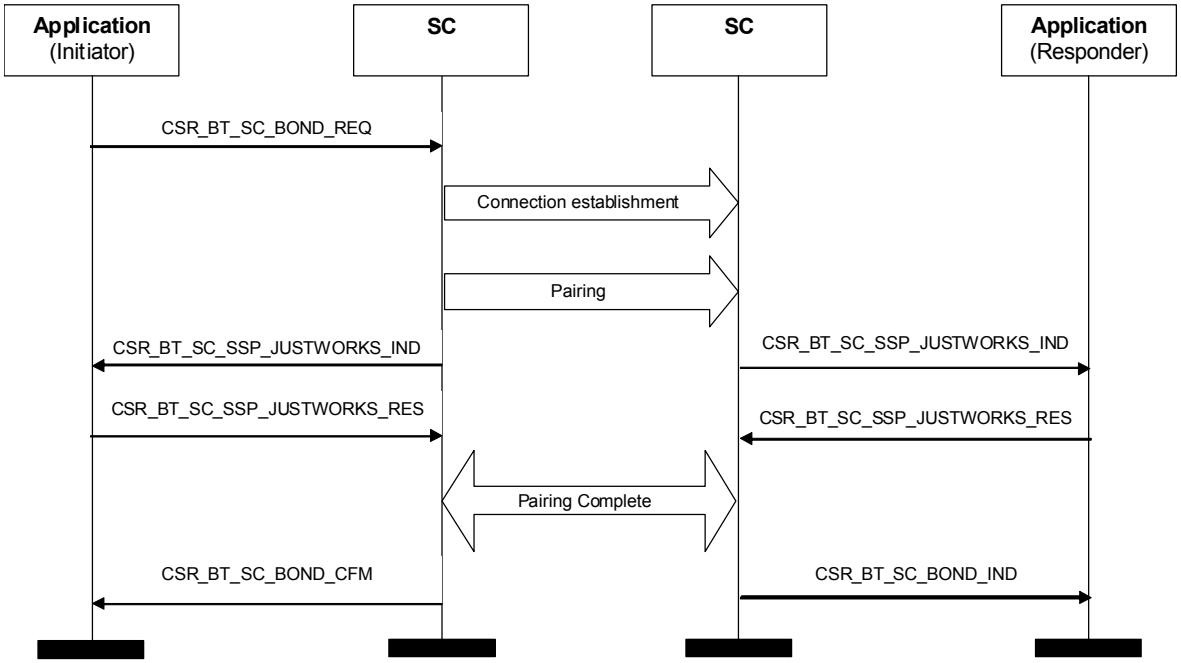


Figure 7: SSP Just Works bonding sequence

The SC will remove any exiting bonds between the two devices before commencing the bonding procedure. The application layer must specify the Bluetooth® device address of the device to bond with.

The SC will request the application layer to accept the bonding by sending a CSR_BT_SC_SSP_JUSTWORKS_IND. The application may reject or accept bonding according to some policy or forward it to the MMI to allow the user to determine the verdict. Note, that there is no number to compare or passkey to enter for this primitive.

A CSR_BT_SC_BOND_CFM is returned to the application on the initiating side as a confirmation that the bonding is completed. The bonding may fail if if it is not possible to establish a connection toward the remote device. If it is requested, in the CSR_BT_SC_SSP_JUSTWORKS_RES message, the SC will store a copy of the just generated link key in NVS. On the responding side no response is sent to the application but in case of successful pairing the security database is invoked for a write request of a new link key.

On the responding side a CSR_BT_SC_SSP_JUSTWORKS_IND is received. This signal must be responded with a CSR_BT_SC_SSP_JUSTWORKS_RES or optionally a reject if bonding is not permitted.

If a CSR_BT_SC_SSP_JUSTWORKS_IND is not responded within a window defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT in csr_bt_usr_config.h, the SC will interpret it as if the a CSR_BT_SC_SSP_JUSTWORKS_RES where the bonding procedure was denied and it will then handle accordingly to this by rejecting the request and notify the application that the bonding procedure has failed. Please be aware that the timeout value is defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT.

3.1.10 Rebond

The rebond procedure is started by the Security Controller when it detects that a connection establishment has failed due to the remote device having discarded/lost its link key. When this occurs the application will receive a CSR_BT_SC_REBOND_IND. This application must determine whether to go forward with the rebond or reject it. The rebond sequence is outlined in Figure 8.

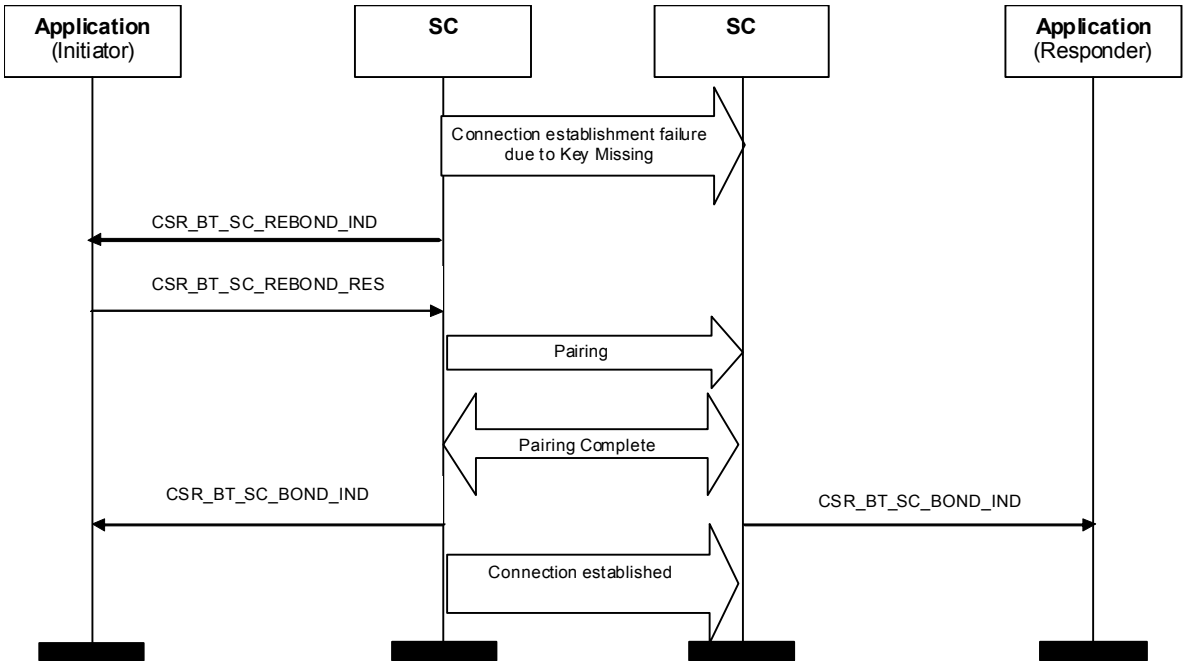


Figure 8: Rebond sequence

If the application rejects the rebond indication, pairing will not be attempted by the Security Controller and subsequently the connection establishment request will fail. However, if the application chooses to rebond, the Security Controller will attempt to initiate pairing and if successful, notify the Connection Manager to retry connection establishment. To indicate that the rebond procedure has completed, the application will receive a CSR_BT_SC_BOND_IND. The application must be able to respond appropriately as described in Section 3.1.5 – 3.1.9 to successfully complete pairing. This will be followed by a connection establishment confirm. Note, in this case the first failed connection establishment is hidden from the application.

Note, this retry procedure is only attempted once.

If a CSR_BT_SC_REBOND_IND is not responded within a window defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT in csr_bt_usr_config.h, the SC will interpret it as a CSR_BT_SC_REBOND_RES where the rebond procedure was denied and it will then handle accordingly by not attempting pairing and notify the application that the rebond procedure has failed. Please be aware that the timeout value is defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT.

3.1.11 SSP OOB Bonding

The application can opt to use OOB for bonding. This is done by enabling pairing indications (see Section 4.19) prior to initiating bonding. The bonding procedure is started when the application layer sends a CSR_BT_SC_BOND_REQ to the SC. The bonding sequence is outlined in Figure 9 for the OOB case.

Note, that the OOB will be used if either side indicates that they have OOB data.

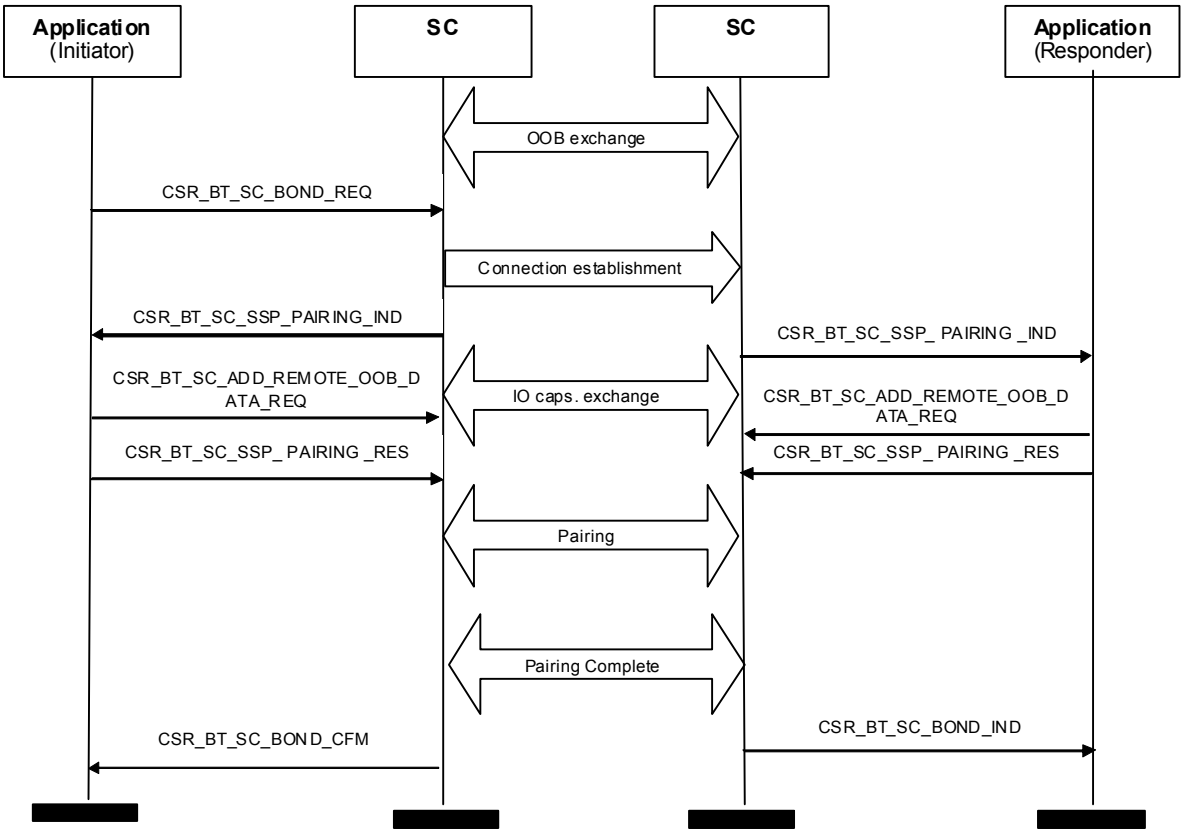


Figure 9: SSP OOB Bonding sequence

The SC will remove any exiting bonds between the two devices before commencing the bonding procedure. The application layer must specify the Bluetooth® device address of the device to bond with.

The SC will request the application layer to accept the bonding by sending a CSR_BT_SC_SSP_PAIRING_IND. The application should request the user through the MMI to accept or reject the request. Note, that there is no number to compare or passkey to enter for this primitive.

A CSR_BT_SC_BOND_CFM is returned to the application on the initiating side as a confirmation that the bonding is completed. The bonding may fail if it is not possible to establish a connection toward the remote device. If it is requested, in the CSR_BT_SC_SSP_PAIRING_RES message, the SC will store a copy of the just generated link key in NVS. On the responding side no response is sent to the application but in case of successful pairing the security database is invoked for a write request of a new link key.

On the responding side a CSR_BT_SC_SSP_PAIRING_IND is received. This signal must be responded with a CSR_BT_SC_SSP_PAIRING_RES or optionally a reject if bonding is not permitted.

If a CSR_BT_SC_SSP_PAIRING_IND is not responded within a window defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT in csr_bt_usr_config.h, the SC will interpret it as if the a CSR_BT_SC_SSP_PAIRING_RES where the bonding procedure was denied and it will then handle accordingly to this by rejecting the request and notify the application that the bonding procedure has failed. Please be aware that the timeout value is defined by CSR_BT_SC_PASSKEY_RESPONSE_TIMEOUT.

3.1.12 Cancel Bonding

The application can cancel a CSR_BT_SC_BOND_REQ by sending a CSR_BT_SC_CANCEL_BOND_REQ. If the CSR_BT_SC_BOND_REQ is cancelled the responses will be a CSR_BT_SC_BOND_CFM with a result code different from CSR_BT_SUCCESS, please refer to csr_bt_profiles.h.

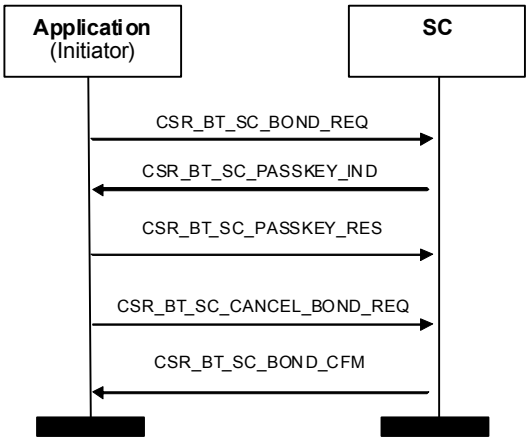


Figure 10: Cancel Bonding

3.1.13 Link Level Initiated Pairing (Security mode 3)

If either side has enabled link level security, i.e. the Bluetooth® link must be authenticated before it is fully established, either side may receive a request for a new passkey.

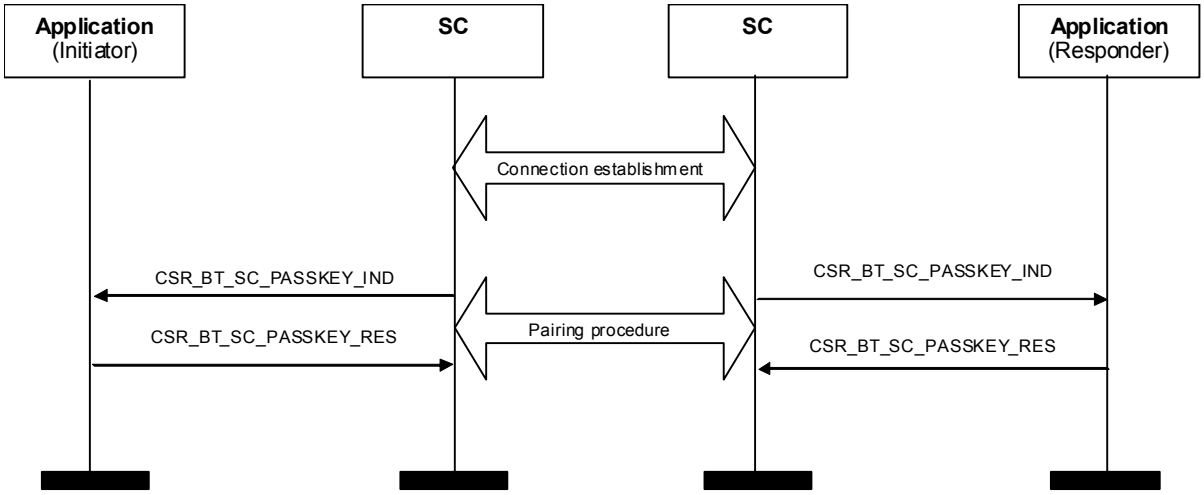


Figure 11: Link level bonding sequence

In case of successful pairing the security database is invoked for a write request of a new link key if it has been requested in the CSR_BT_SC_PASSKEY_RES message. In the CSR_BT_SC_PASSKEY_RES messages the application can also decide if the peer device shall be marked as trusted or untrusted. Where trusted devices are automatically granted access by the Security Manager and untrusted devices result in a CSR_BT_SC_AUTHORISE_IND, see section 3.1.15. Please note that a CSR_BT_SC_AUTHORISE_IND is only sent to the application when an untrusted device is attempting to access a service that requires authorisation in security mode 2/4. The mandatory security setting for each service can be found in csr_bt_usr_config.h and the default security setting can be set in csr_bt_profiles.h.

3.1.14 Set Trust Level

If a peer device is stored in NVS and set to require authorisation in csr_bt_usr_config.h, the application can use this message to change the trust level of a device, e.g. trusted or untrusted.

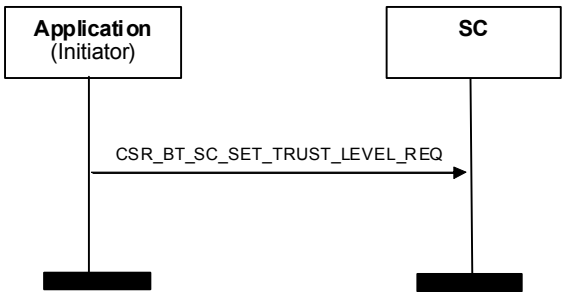


Figure 12: Changing trust level

3.1.15 Authorisation

If authorisation is enabled the application layer is requested to grant access when a connection is established from a mistrusted device. When devices are already paired and trusted, authorisation will not take place.

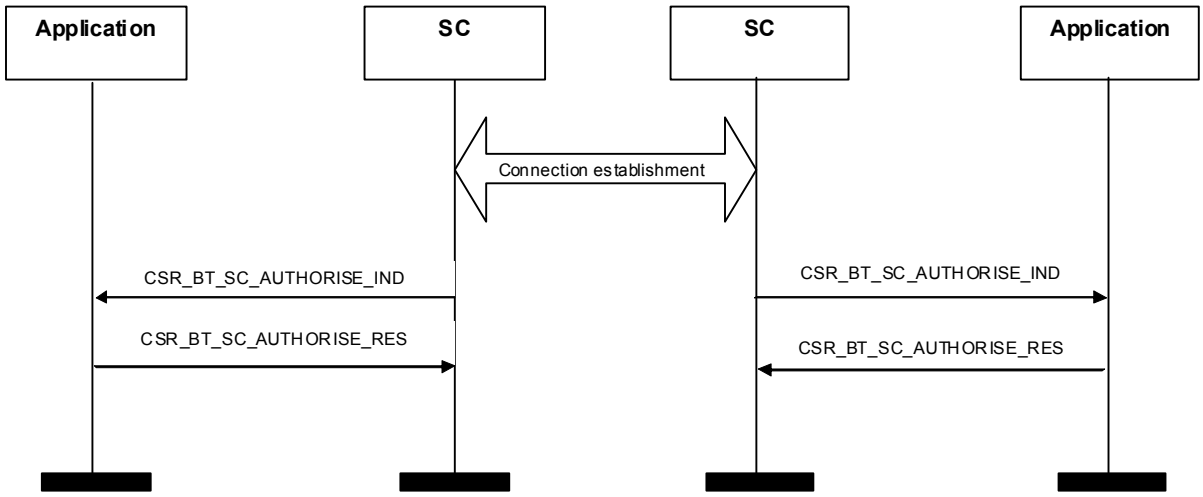


Figure 13: Authorisation sequence

Please note that the authorisation procedure is a local procedure, which does not involve any over the air procedure and where a user of a Bluetooth device grants a specific (remote) Bluetooth device access to a specific service/profile. This means that if authorisation is enabled for the HFG, the AV and the AVRCP profile, the application will receive a SC_AUTHORISE_IND when a service level connection is being established to the HFG and once again when a service level connection is being established to the AV profile and again when a service level connection is made to AVRCP. Please also note that the application **must** respond to a SC_AUTHORISE_IND, because the SC will not accept other signals until this is done.

3.1.16 De-Bonding

A bond between two devices may be removed using the CSR_BT_SC_DEBOND_REQ/CFM signal pair. The de-bonding procedure is a fully local procedure, i.e. no information is exchanged between the devices for which the bond exists.

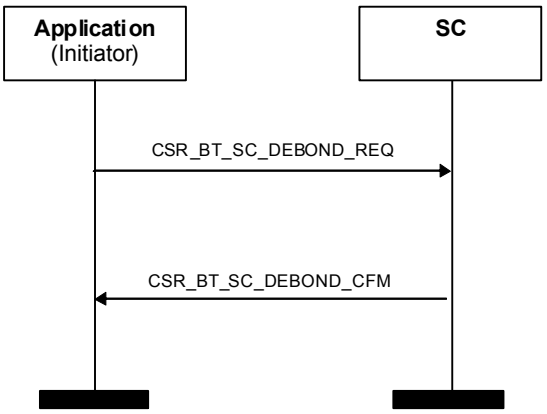


Figure 14: De-bonding sequence

As the procedure is a local procedure the remote device may still be in the possession of a link key for the device. This means that the next time a secure connection must be established, a pairing procedure needs to be carried out to create a new link key.

Please note that if the link key is currently in use for a connection, then the link key will not be deleted from the HCI layer until all the connections are disconnected.

3.1.17 Encryption Control

With the CSR_BT_SC_ENCRYPTION_REQ/CFM signal the encryption mode of a specified active connection can be switched on/off. This allows the application to control the protection of the data being exchanged based on the application's knowledge of the sensitive nature of the data.

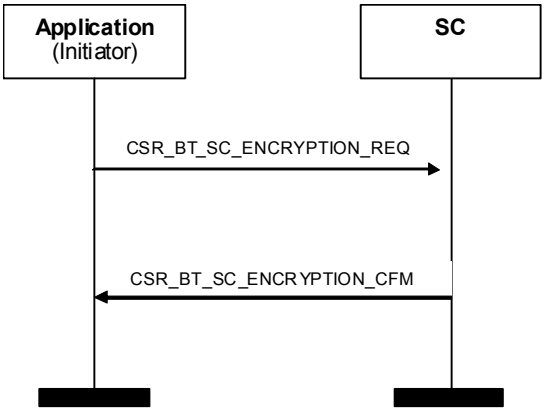


Figure 15: Encryption control sequence

If the devices have not been paired at the time the encryption procedure is initiated (switching encryption on), pairing will be initiated at link-level as part of the encryption procedure. As a result of this, a request for a passkey will be sent to the application prior to switching the encryption on.

3.1.18 Security Mode Setting

The security mode to be applied by the device can be determined by the application through the CSR_BT_SC_SET_SECURITY_MODE_REQ/CFM. The different security modes (1-4) are described in section 2.1; the default security mode is set to 2 at start up by the SC (can be set in csr_bt_usr_config.h file).

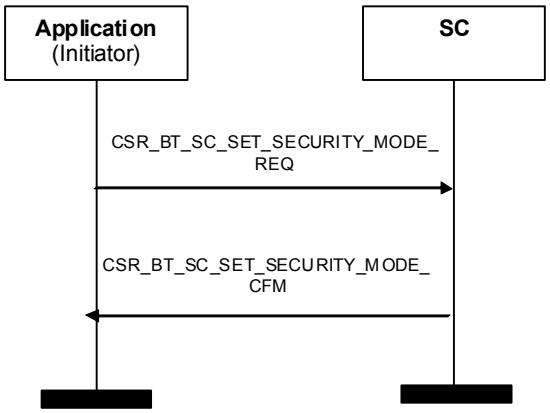


Figure 16: Sequence for setting the security mode of the device

Mode 2 or 4 must be used in order to let the different profile managers decide which security procedures should be applied. In security mode 2/4 the profile managers assure that the security procedures as defined in the Bluetooth® profile specification are applied. Mode 4 is recommended for interoperability reasons.

Please note that the security mode setting is applied for the device and not just for single connection between two devices.

3.1.19 SSP Debug Mode

In order to enable SSP debug mode, the application must use CSR_BT_SC_DEBUG_MODE_REQ.

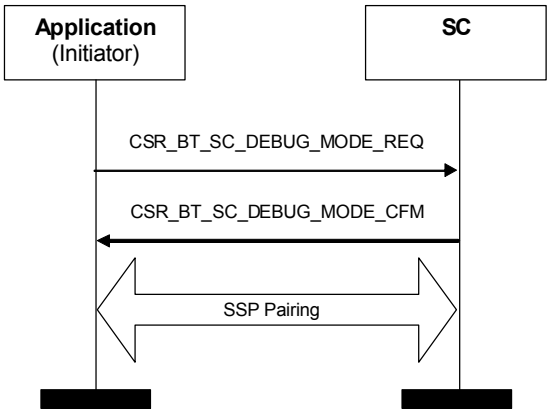


Figure 17: Sequence for changing SSP debug mode

A CSR_BT_SC_DEBUG_MODE_CFM containing a success will indicate that the local device has successfully entered SSP debug mode. A Bluetooth sniffer (e.g., FTS) will now be able to track the air traffic using a set of predefined Diffie Hellman public/ private keys.

3.1.20 Bondable without service records registered

In order to make the local device bondable to other devices, the application must use CSR_BT_SC_ACCEPT_BOND_REQ.

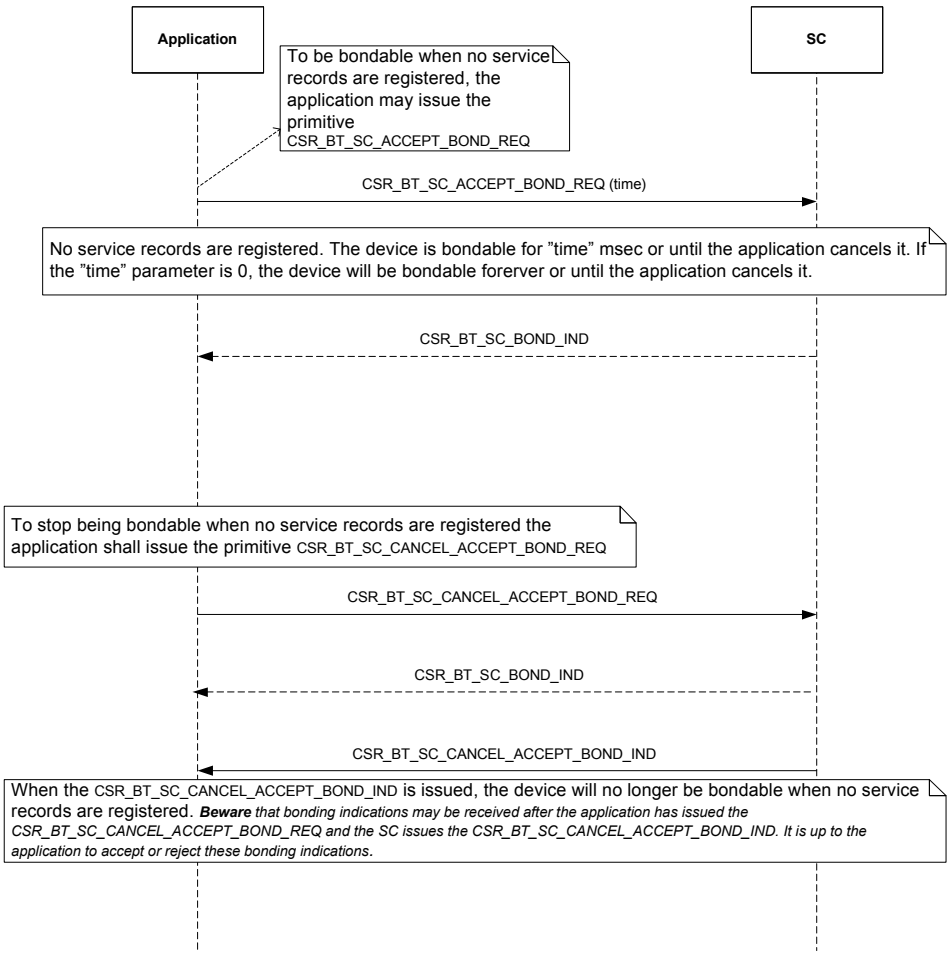


Figure 18: Sequence for making the device bondable

The application may want the device to be bondable for a determined period. In that case, it shall indicate the time in milliseconds in the CSR_BT_SC_ACCEPT_BOND_REQ. When that time passes, the device will cease to accept bonding from remote devices and the application will receive a CSR_BT_SC_CANCEL_ACCEPT_BOND_IND. If the time parameter is zero the device will be bondable until the application issues the CSR_BT_SC_CANCEL_ACCEPT_BOND_REQ. The answer to that primitive is a CSR_BT_SC_CANCEL_ACCEPT_BOND_IND. Beware that devices may be able to bond to the local device after the application has sent the CSR_BT_SC_CANCEL_ACCEPT_BOND_REQ. It is only after the SC sends the CSR_BT_SC_CANCEL_ACCEPT_BOND_IND that the device does not accept bonding from remote devices.

3.1.21 Bluetooth Low Energy Security

Bluetooth 4.0 “Low Energy” reuses the Secure Simple Pairing used by Bluetooth 2.1, and as the application developer does not need to take any special precautions. Still, there are a few differences between classic Bluetooth Secure Simple Pairing and Low Energy Secure Simple Pairing:

- Low Energy uses only Just Works or Passkey. Numerical Comparison is not used on Low Energy
- Security on Low Energy is always performed Just in Time. This means that the application should never actively ask for security on a Low Energy link. The Generic Attribute Profile (GATT) will automatically start security when needed

- The final security indication signal for Low Energy (CsrBtScLeSecurityInd) is different from the BR/EDR case.

- A few new utility functions have been added to the SC API to facilitate the security parameters the application may need to tweak, e.g. encryption key size and key distribution.

The security requirements for Low Energy connections are controlled using CSR_BT_SC_SET_AUTH_REQUIREMENTS (see 4.17), while the bonding parameters are controlled via CSR_BT_SC_LE_KEY_DISTRIBUTION_REQ (see 4.28).

The security requirements can either be encryption and/or authentication (man-in-the-middle protection), while the key distribution sets what keys are exchanged during bonding: encryption key, identity resolving key and/or the data signing key.

4 Security Controller Primitives

This section gives an overview of the primitives and parameters in the interface. Detailed information can be found in the corresponding csr_bt_sc_prim.h file.

4.1 List of All Primitives

Primitives	Reference
CSR_BT_SC_ACTIVATE_REQ	See section 4.2
CSR_BT_SC_BOND_REQ	See section 4.3
CSR_BT_SC_BOND_CFM	See section 4.3
CSR_BT_SC_BOND_IND	See section 4.3
CSR_BT_SC_DEBOND_REQ	See section 4.4
CSR_BT_SC_DEBOND_CFM	See section 4.4
CSR_BT_SC_PASSKEY_IND	See section 4.5
CSR_BT_SC_PASSKEY_RES	See section 4.5
CSR_BT_SC_ENCRYPTION_REQ	See section 4.6
CSR_BT_SC_ENCRYPTION_CFM	See section 4.6
CSR_BT_SC_SET_SECURITY_MODE_REQ	See section 4.7
CSR_BT_SC_SET_SECURITY_MODE_CFM	See section 4.7
CSR_BT_SC_DEBUG_MODE_REQ	See section 4.8
CSR_BT_SC_DEBUG_MODE_CFM	See section 4.8
CSR_BT_SC_AUTHORISE_IND	See section 4.9
CSR_BT_SC_AUTHORISE_RES	See section 4.9
CSR_BT_SC_SET_TRUST_LEVEL_REQ	See section 4.10
CSR_BT_SC_CANCEL_BOND_REQ	See section 4.11
CSR_BT_SC_SSP_PASSKEY_IND	See section 4.12
CSR_BT_SC_SSP_PASSKEY_RES	See section 4.12
CSR_BT_SC_SSP_COMPARE_IND	See section 4.13
CSR_BT_SC_SSP_COMPARE_RES	See section 4.13
CSR_BT_SC_SSP_NOTIFICATION_IND	See section 4.14
CSR_BT_SC_SSP_NOTIFICATION_RES	See section 4.14
CSR_BT_SC_SSP_JUSTWORKS_IND	See section 4.15
CSR_BT_SC_SSP_JUSTWORKS_RES	See section 4.15
CSR_BT_SC_SET_IO_CAPABILITY_REQ	See section 4.16
CSR_BT_SC_SET_AUTH_REQUIREMENTS_REQ	See section 4.17
CSR_BT_SC_MODE_REQ	See section 4.18
CSR_BT_SC_MODE_IND	See section 4.18
CSR_BT_SC_SET_EVENT_MASK_REQ	See section 4.19
CSR_BT_SC_REBOND_IND	See section 4.20
CSR_BT_SC_REBOND_RES	See section 4.20
CSR_BT_SC_SSP_PAIRING_IND	See section 4.21
CSR_BT_SC_SSP_PAIRING_RES	See section 4.21
CSR_BT_SC_ADD_REMOTE_OOB_DATA_REQ	See section 4.22

Primitives	Reference
CSR_BT_SC_READ_LOCAL_OOB_DATA_REQ	See section 4.23
CSR_BT_SC_READ_LOCAL_OOB_DATA_CFM	See section 4.23
CSR_BT_SC_SSP_KEYPRESS_NOTIFICATION_REQ	See section 4.24
CSR_BT_SC_SSP_KEYPRESS_NOTIFICATION_IND	See section 4.24
CSR_BT_SC_CONFIG_REQ	See section 4.25
CSR_BT_SC_ACCEPT_BOND_REQ	See section 4.26
CSR_BT_SC_CANCEL_ACCEPT_BOND_REQ	See section 4.26
CSR_BT_SC_CANCEL_ACCEPT_BOND_IND	See section 4.26
CSR_BT_SC_ENCRYPTION_KEY_SIZE_REQ	See section 4.27
CSR_BT_SC_ENCRYPTION_KEY_SIZE_CFM	See section 4.27
CSR_BT_SC_LE_KEY_DISTRIBUTION_REQ	See section 4.28
CSR_BT_SC_SET_ENCRYPTION_KEY_SIZE_REQ	See section 4.29
CSR_BT_SC_LE_SECURITY_IND	See section 4.30

Table 4: List of all primitives

4.2 CSR_BT_SC_ACTIVATE

Primitives \ Parameters	type	phandle
CSR_BT_SC_ACTIVATE_REQ	✓	✓

Table 5: CSR_BT_SC_ACTIVATE Primitives

Description

Register a default application layer handle in the Security Controller. The default application layer handle is used as return address for unsolicited signals received from the lower layers in the stack and for which application layer response is required.

If the default handle can be determined at compile time it is possible to use the default handle define in the configuration file. If the compile time configuration is used and a CSR_BT_SC_ACTIVATE_REQ is sent, the CSR_BT_SC_ACTIVATE_REQ overwrites the compiler define.

Parameters

type	The signal identity, CSR_BT_SC_ACTIVATE_REQ.
phandle	The identity of the calling process. It is possible to initiate the bonding procedure by any higher layer process as the response is returned to phandle.

4.3 CSR_BT_SC_BOND

Parameters / Primitives	type	phandle	deviceAddr	cod	addedToScDbList	resultCode	resultSupplier	addressType
CSR_BT_SC_BOND_REQ	✓	✓	✓					
CSR_BT_SC_BOND_IND	✓		✓	✓	✓	✓		✓
CSR_BT_SC_BOND_CFM	✓		✓	✓	✓	✓	✓	✓

Table 6: CSR_BT_SC_BOND Primitives

Description

Start a Dedicated Bonding process for creation of a common bond (the link key) between a pair of Bluetooth® devices.

Parameters

type	The signal identity, CSR_BT_SC_BOND_REQ/IND/CFM.
phandle	The identity of the calling process. It is possible to initiate the bonding procedure by any higher layer process as the response is returned to phandle.
deviceAddr	The Bluetooth® device address of the device to which a bond is being created. If the procedure is initiated locally the address must have been obtained using another procedure, e.g. using the discovery procedure.
cod	Class of device of peer device with which a bond is created. The *cod* is defined in the Bluetooth specification assigned numbers. Please note that it cannot be guaranteed that the cod can be assign doing the pairing process.
addedToScDbList	If TRUE the device is added to NVS else not
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
addressType	Address type, always CSR_BT_ADDR_PUBLIC.

Notes

This signal pair is used in combination with either:

- CSR_BT_SC_PASSKEY_IND/RES
- CSR_BT_SC_SSP_PASSKEY_IND/RES
- CSR_BT_SC_SSP_COMPARE_IND/RES
- CSR_BT_SC_SSP_NOTIFICATION_IND/RES
- CSR_BT_SC_SSP_COMPARE_IND/RES
- CSR_BT_SC_SSP_JUSTWORKS_IND/RES

4.4 CSR_BT_SC_DEBOND

Primitives \ Parameters	type	resultCode	resultSupplier	phandle	deviceAddr	addressType
CSR_BT_SC_DEBOND_REQ	✓			✓	✓	✓
CSR_BT_SC_DEBOND_CFM	✓	✓	✓		✓	✓

Table 7: CSR_BT_SC_DEBOND Primitives

Description

Remove an existing bond between a pair of devices. Invoking this procedure a new pairing procedure must be completed before any security procedure can be completed. Please note that if the link key is currently in use for a connection, then the link key will not be deleted from the HCI layer until all the connections are disconnected. The application can use the zero device address (0000:00:000000) to debond all devices currently in NVS.

Parameters

type	The signal identity, CSR_BT_SC_DEBOND_REQ/CFM.
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
phandle	The identity of the calling process. It is possible to initiate the bonding procedure by any higher layer process as the response is returned to phandle.
deviceAddr	The Bluetooth® device address for which a bond must be removed.
addressType	Address type. For classic Bluetooth this is always CSR_BT_ADDR_PUBLIC. For Low Energy devices, this may an integral part of the device address and may be either CSR_BT_ADDR_PUBLIC or CSR_BT_ADDR_PRIVATE.

4.5 CSR_BT_SC_PASSKEY

Parameters / Primitives	type	cod	deviceAddr	deviceName	accept	passKeyLength	passKey	addDevice	paired	authorised	initiator	addressType
CSR_BT_SC_PASSKEY_IND	✓	✓	✓	✓					✓		✓	✓
CSR_BT_SC_PASSKEY_RES	✓		✓		✓	✓	✓	✓		✓		✓

Table 8: CSR_BT_SC_PASSKEY Primitives

Description

Request for entering a Bluetooth device passkey. The application may respond to the request or may forward the request to the user, e.g. through the MMI.

Parameters

type	The signal identity, CSR_BT_SC_PASSKEY_IND/RES
cod	Class of device of peer device with which a bond is created. The CoD is defined in the Bluetooth specification assigned numbers. Please note that it cannot be guaranteed that the cod can be assign doing the pairing process.
deviceAddr	The Bluetooth® device address of the device requesting a passkey to be entered.
deviceName	The name of the remote device.
accept	TRUE to accept the pairing attempt, FALSE to reject.
passKeyLength	The length of the passKey. The maximum number of digit is 16
passkey	The passkey entered in ASCII.
addDevice	If TRUE the device is added to NVS else not.
paired	Indicates whether there is a bond between the local and remote device. The following values are applicable (from csr_bt_sc_prim.h): CSR_BT_SC_PAIRING_NONE - no bond exists CSR_BT_SC_PAIRING_LEGACY - paired with legacy method CSR_BT_SC_PAIRING_NO_MITM - paired with SSP and no MITM CSR_BT_SC_PAIRING_MITM - paired with SSP and MITM
authorised	TRUE if authorisation is automatic granted, e.g. the device is marked as trusted. FALSE if not
Initiator	TRUE if this indication was triggered by the local host
addressType	Address type. For classic Bluetooth this is always CSR_BT_ADDR_PUBLIC. For Low Energy devices, this may an integral part of the device address and may be either CSR_BT_ADDR_PUBLIC or CSR_BT_ADDR_PRIVATE.

4.6 CSR_BT_SC_ENCRYPTION

Parameters / Primitives	type	phandle	resultCode	resultSupplier	encEnable	deviceAddr
CSR_BT_SC_ENCRYPTION_REQ	✓	✓			✓	✓
CSR_BT_SC_ENCRYPTION_CFM	✓		✓	✓	✓	✓

Table 9: CSR_BT_SC_ENCRYPTION Primitives

Description

Request for changing the encryption mode of an active connection.

Parameters

type	The signal identity, CSR_BT_SC_ENCRYPTION_REQ/CFM
phandle	The identity of the calling process. It is possible to initiate the encryption control procedure by any higher layer process as the response is returned to phandle.
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
encEnable	In a request the wanted encryption mode and in a confirm the current encryption mode. (TRUE: on, FALSE: off)
deviceAddr	The Bluetooth® device address of the connected device.

4.7 CSR_BT_SC_SET_SECURITY_MODE

Primitives \ Parameters	type	resultCode	resultSupplier	phandle	securityMode
CSR_BT_SC_SET_SECURITY_MODE_REQ	✓			✓	✓
CSR_BT_SC_SET_SECURITY_MODE_CFM	✓	✓	✓		

Table 10: CSR_BT_SC_SET_SECURITY_MODE Primitives

Description

Set the security mode to be used by the device. The setting is applied to all connections and services invoked. The possible security modes are defined in Section 2.1.

Parameters

type	The signal identity, CSR_BT_SC_SET_SECURITY_MODE_REQ/CFM.
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
phandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to phandle.
securityMode	The security mode, 1 to 4, to be applied.

4.8 CSR_BT_SC_DEBUG_MODE

Primitives \ Parameters	type	resultCode	resultSupplier	phandle	enable	enabled
CSR_BT_SC_DEBUG_MODE_REQ	✓			✓	✓	
CSR_BT_SC_DEBUG_MODE_CFM	✓	✓	✓			✓

Table 11: CSR_BT_SC_DEBUG_MODE Primitives

Description

Request to enable/disable SSP debug mode. This will only affect new connections.

Parameters

type	The signal identity, CSR_BT_SC_DEBUG_MODE_REQ/CFM.
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
phandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to phandle.
enable	If TRUE the SSP debug mode will be enabled.
enabled	If TRUE the SSP debug mode has been enabled.

4.9 CSR_BT_SC_AUTHORISE

Primitives \ Parameters	type	deviceAddr	deviceName	serviceId	authorised	protocolId	channel	incoming	addressType
CSR_BT_SC_AUTHORISE_IND	✓	✓	✓	✓					✓
CSR_BT_SC_AUTHORISE_EXT_IND	✓	✓	✓	✓		✓	✓	✓	✓
CSR_BT_SC_AUTHORISE_RES	✓	✓			✓				

Table 12: CSR_BT_SC_AUTHORISE Primitives

Description

Request for authorisation of a remote device and the service requested by the remote device. Response must indicate if the application layer grants access to this service or not.

Note: In case the application has set the Security Controller's event mask to CSR_BT_SC_EVENT_MASK_AUTHORISE_EXTENDED then the CSR_BT_SC_AUTHORISE_EXT_IND primitive replaces the CSR_BT_SC_AUTHORISE_IND primitive. How the Security Controller's event mask can be set in explained in section 4.19.

Parameters

type	The signal identity, CSR_BT_SC_AUTHORISE_IND/EXT_IND/RES
deviceAddr	The Bluetooth® device address for which authorisation is requested.
deviceName	The name of the remote device.
serviceId	The UUID of the service for which authorisation is requested.
authorized	TRUE if authorisation granted, FALSE to reject
protocolId	A security protocol identifier, can either be CSR_BT_SC_PROTOCOL_RFCOMM or CSR_BT_SC_PROTOCOL_L2CAP. These values are defined in csr_bt_profiles.h
channel	The local Channel, which has been authorized. Note if the 'protocolId' is CSR_BT_SC_PROTOCOL_RFCOMM the channel refers to a local server channel number, and if it is CSR_BT_SC_PROTOCOL_L2CAP the channel refers to a local PSM.
Incoming	Flag indicating peer-initiated (TRUE) or locally-initiated (FALSE)
addressType	Address type. Always CSR_BT_ADDR_PUBLIC.

4.10 CSR_BT_SC_SET_TRUST_LEVEL

Parameters / Primitives	type	deviceAddr	authorised	addressType
CSR_BT_SC_SET_TRUST_LEVEL_REQ	✓	✓	✓	✓

Table 13: CSR_BT_SC_SET_TRUST_LEVEL Primitives

Description

If a peer device is stored in NVS and set to require authorisation in csr_bt_usr_config.h, the application can use this message to change the trust level of a device, e.g. trusted or untrusted. Trusted devices are automatically granted access by the Security Manager and untrusted devices result in a CSR_BT_SC_ AUTHORISE_IND.

Parameters

type	The signal identity, CSR_BT_SC_SET_TRUST_LEVEL_REQ.
deviceAddr	The Bluetooth® device address of the device, where the trust level must be changed
authorised	TRUE to mark the device a trusted, e.g. authorisation is automatic granted. FALSE to mark device as untrusted.
addressType	Address type. Always CSR_BT_ADDR_PUBLIC.

4.11 CSR_BT_SC_CANCEL_BOND

Primitives \ Parameters	type	phandle	deviceAddr	addressType
CSR_BT_SC_CANCEL_BOND_REQ	✓	✓	✓	✓

Table 14: CSR_BT_SC_CANCEL_BOND Primitives

Description

The CSR_BT_SC_CANCEL_BOND_REQ is used for cancelling a CSR_BT_SC_BOND_REQ procedure. If the SC succeeds to cancel the CSR_BT_SC_BOND_REQ the application will receive a CSR_BT_SC_BOND_CFM with a result code different from CSR_BT_SUCCESS, please see csr_bt_profiles.h.

Parameters

type	The signal identity, CSR_BT_SC_CANCEL_BOND_REQ.
phandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to phandle.
deviceAddr	The Bluetooth® device address of the device, given previously in the CSR_BT_SC_BOND_REQ
addressType	Address type. For classic Bluetooth this is always CSR_BT_ADDR_PUBLIC. For Low Energy devices, this may an integral part of the device address and may be either CSR_BT_ADDR_PUBLIC or CSR_BT_ADDR_PRIVATE.

4.12 CSR_BT_SC_SSP_PASSKEY

Primitives \ Parameters	type	cod	deviceAddr	deviceName	accept	numericValue	paired	authRequirements	addDevice	authorised	initiator	addressType	localAuthRequirements
CSR_BT_SC_SSP_PASSKEY_IND	✓	✓	✓	✓			✓	✓			✓	✓	✓
CSR_BT_SC_SSP_PASSKEY_RES	✓		✓		✓	✓			✓	✓		✓	

Table 15: CSR_BT_SC_SSP_PASSKEY Primitives

Description

Request for entering a Bluetooth® device numeric value. The application may respond to the request or may forward the request to the user, e.g. through the MMI.

Note: This primitive is only applicable when both the local and remote device supports SSP.

Parameters

type	The signal identity, CSR_BT_SC_SSP_PASSKEY_IND/RES
cod	Class of device of peer device with which a bond is created. The CoD is defined in the Bluetooth specification assigned numbers. Please note that it cannot be guaranteed that the cod can be assign doing the pairing process.
deviceAddr	The Bluetooth® device address of the device requesting a passkey to be entered.
deviceName	The name of the remote device.
accept	TRUE to accept the pairing attempt, FALSE to reject.
numericValue	The numeric value.
paired	Indicates whether there is a bond between the local and remote device. The following values are applicable (from csr_bt_sc_prim.h): CSR_BT_SC_PAIRING_NONE - no bond exists CSR_BT_SC_PAIRING_LEGACY - paired with legacy method CSR_BT_SC_PAIRING_NO_MITM - paired with SSP and no MITM CSR_BT_SC_PAIRING_MITM - paired with SSP and MITM
authRequirements	The authentication requirements sent by the remote device. See HCI_MITM_XXX defines in hci.h for possible values.
localAuthRequirements	The authentication requirements of the local device
addDevice	If TRUE the device is added to NVS else not.
authorised	TRUE if authorisation is automatic granted, e.g. the device is marked as trusted. FALSE if not
initiator	TRUE if this indication was triggered by the local host FALSE if not
addressType	Address type. For classic Bluetooth this is always CSR_BT_ADDR_PUBLIC. For Low

Energy devices, this may an integral part of the device address and may be either CSR_BT_ADDR_PUBLIC or CSR_BT_ADDR_PRIVATE.

4.13 CSR_BT_SC_SSP_COMPARE

Parameters / Primitives	type	cod	deviceAddr	deviceName	accept	numericValue	paired	authRequirements	addDevice	authorised	initiator	addressType	localAuthRequirements
CSR_BT_SC_SSP_COMPARE_IND	✓	✓	✓	✓		✓	✓	✓			✓	✓	✓
CSR_BT_SC_SSP_COMPARE_RES	✓		✓		✓				✓	✓			

Table 16: CSR_BT_SC_SSP_COMPARE Primitives

Description

Request for comparing a Bluetooth® device numeric value. The application must forward the request to the user, e.g. through the MMI to allow the user to make the comparison with the value shown on the remote device's display.

Note: This primitive is only applicable when both the local and remote device supports SSP.

Parameters

type	The signal identity, CSR_BT_SC_SSP_COMPARE_IND/RES
cod	Class of device of peer device with which a bond is created. The CoD is defined in the Bluetooth specification assigned numbers. Please note that it cannot be guaranteed that the cod can be assign doing the pairing process.
deviceAddr	The Bluetooth® device address of the device requesting a passkey to be entered.
deviceName	The name of the remote device.
accept	TRUE to accept the pairing attempt, FALSE to reject.
numericValue	The numeric value that should be verified by the user.
paired	Indicates whether there is a bond between the local and remote device. The following values are applicable (from csr_bt_sc_prim.h): CSR_BT_SC_PAIRING_NONE - no bond exists CSR_BT_SC_PAIRING_LEGACY - paired with legacy method CSR_BT_SC_PAIRING_NO_MITM - paired with SSP and no MITM CSR_BT_SC_PAIRING_MITM - paired with SSP and MITM
authRequirements	The authentication requirements sent by the remote device. See HCI_MITM_XXX defines in hci.h for possible values.
localAuthRequirements	The authentication requirements of the local device
addDevice	If TRUE the device is added to NVS else not.
authorised	TRUE if authorisation is automatic granted, e.g. the device is marked as trusted. FALSE if not

Initiator	TRUE if this indication was triggered by the local host
addressType	Address type. Not used. Always CSR_BT_ADDR_PUBLIC.

4.14 CSR_BT_SC_SSP_NOTIFICATION

Primitives \ Parameters	type	cod	deviceAddr	deviceName	accept	numericValue	paired	authRequirements	addDevice	authorised	initiator	addressType	localAuthRequirements
CSR_BT_SC_SSP_NOTIFICATION_IND	✓	✓	✓	✓		✓	✓	✓			✓	✓	✓
CSR_BT_SC_SSP_NOTIFICATION_RES	✓		✓		✓				✓	✓			

Table 17: CSR_BT_SC_SSP_NOTIFICATION Primitives

Description

Request for displaying a Bluetooth® device numeric value. The application may respond to the request or may forward the request to the user, e.g. through the MMI. The application must display this value even after it has responded to the request allowing the remote user to enter the shown numeric value.

Note: This primitive is only applicable when both the local and remote device supports SSP.

Parameters

type	The signal identity, CSR_BT_SC_SSP_NOTIFICATION_IND/RES
cod	Class of device of peer device with which a bond is created. The CoD is defined in the Bluetooth specification assigned numbers. Please note that it cannot be guaranteed that the cod can be assign doing the pairing process.
deviceAddr	The Bluetooth® device address of the device requesting a passkey to be entered.
deviceName	The name of the remote device.
accept	TRUE to accept the pairing attempt, FALSE to reject.
numericValue	The numeric value that should be entered by the remote user.
paired	Indicates whether there is a bond between the local and remote device. The following values are applicable (from csr_bt_sc_prim.h): CSR_BT_SC_PAIRING_NONE - no bond exists CSR_BT_SC_PAIRING_LEGACY - paired with legacy method CSR_BT_SC_PAIRING_NO_MITM - paired with SSP and no MITM CSR_BT_SC_PAIRING_MITM - paired with SSP and MITM
authRequirements	The authentication requirements sent by the remote device. See HCI_MITM_XXX defines in hci.h for possible values.
localAuthRequirements	The authentication requirements of the local device.
addDevice	If TRUE the device is added to NVS otherwise not.
authorised	TRUE if authorisation is automatic granted, e.g. the device is marked as trusted. FALSE if not
Initiator	TRUE if this indication was triggered by the local host

addressType	Address type. Not used. Always CSR_BT_ADDR_PUBLIC.

4.15 CSR_BT_SC_SSP_JUSTWORKS

Primitives \ Parameters	type	cod	deviceAddr	deviceName	accept	paired	authRequirements	addDevice	authorised	initiator	addressType	localAuthRequirements
CSR_BT_SC_SSP_JUSTWORKS_IND	✓	✓	✓	✓		✓	✓			✓	✓	✓
CSR_BT_SC_SSP_JUSTWORKS_RES	✓		✓		✓			✓	✓		✓	

Table 18: CSR_BT_SC_SSP_JUSTWORKS Primitives

Description

Request for accepting a pairing attempt. The application may respond to the request or may forward the request to the user, e.g. through the MMI.

Note: This primitive is only applicable when both the local and remote device supports SSP and MITM is not required by both parties.

Parameters

type — The signal identity, CSR_BT_SC_SSP_JUSTWORKS_IND/RES

cod — Class of device of peer device with which a bond is created. The CoD is defined in the Bluetooth specification assigned numbers. Please note that it cannot be guaranteed that the cod can be assign doing the pairing process.

deviceAddr — The Bluetooth® device address of the device requesting a passkey to be entered.

deviceName — The name of the remote device.

accept — TRUE to accept the pairing attempt, FALSE to reject.

paired — Indicates whether there is a bond between the local and remote device. The following values are applicable (from csr_bt_sc_prim.h):

CSR_BT_SC_PAIRING_NONE - no bond exists
CSR_BT_SC_PAIRING_LEGACY - paired with legacy method
CSR_BT_SC_PAIRING_NO_MITM - paired with SSP and no MITM
CSR_BT_SC_PAIRING_MITM - paired with SSP and MITM

authRequirements — The authentication requirements sent by the remote device. See HCI_MITM_XXX defines in hci.h for possible values.

localAuthRequirements — The authentication requirements of the local device.

addDevice — If TRUE the device is added to NVS else not.

authorised — TRUE if authorisation is automatic granted, e.g. the device is marked as trusted. FALSE if not

Initiator — TRUE if this indication was triggered by the local host

addressType — Address type. For classic Bluetooth this is always CSR_BT_ADDR_PUBLIC. For Low Energy devices, this may an integral part of the device address and may be either

CSR_BT_ADDR_PUBLIC or CSR_BT_ADDR_PRIVATE.

4.16 CSR_BT_SC_SET_IO_CAPABILITY

Parameters / Primitives	type	ioCapability
CSR_BT_SC_SET_IO_CAPABILITY_REQ	✓	✓

Table 19: CSR_BT_SC_SET_IO_CAPABILITY Primitive

Description

Set the local device's IO capability. This can also be done by setting the CSR_BT_DEFAULT_IO_CAPABILITY define in csr_bt_usr_config.h

The local and remote device's IO capability will determine which SSP authentication model will be used when either the local or remote device require MITM protection during pairing.

Note: This primitive is only applicable when the local device supports SSP else the primitive is ignored.

Parameters

type	The signal identity, CSR_BT_SC_SET_IO_CAPABILITY_REQ
ioCapability	The IO capability of the local device. The following values are applicable (from hci.h): HCI_IO_CAP_DISPLAY_ONLY – can only display a value HCI_IO_CAP_DISPLAY_YES_NO – can display and enter a value HCI_IO_CAP_KEYBOARD_ONLY – can only enter a value HCI_IO_CAP_NO_INPUT_NO_OUTPUT – no display/input capability

4.17 CSR_BT_SC_SET_AUTH_REQUIREMENTS

Parameters / Primitives	type	authRequirements	leAuthRequirements
CSR_BT_SC_SET_AUTH_REQUIREMENTS_REQ	✓	✓	✓

Table 20: CSR_BT_SC_SET_AUTH_REQUIREMENTS Primitive

Description

Set the local device's authentication requirements. This can also be done by setting the CSR_BT_DEFAULT_AUTH_REQUIREMENTS define in csr_bt_usr_config.h. This parameter is sent to the remote device during SSP and determines which SSP authentication model is used for pairing. If both sides do not require MITM protection, the Just Works procedure is used. Else the IO capability will determine the SSP procedure.
Note: This primitive is only applicable when the local device supports SSP else the primitive is ignored.

Parameters

type — The signal identity, CSR_BT_SC_SET_AUTH_REQUIREMENTS_REQ

authRequirements — The authentication requirements of the local device. The following values are applicable (from hci.h):

HCI_MITM_NOT_REQUIRED_NO_BONDING (Deprecated shall not be used. It is handled by the security controller itself)

HCI_MITM_REQUIRED_NO_BONDING (Deprecated shall not be used. It is handled by the security controller itself)

HCI_MITM_NOT_REQUIRED_DEDICATED_BONDING (MITM not required when the application wishes to do a Dedicated Bonding, by sending a SC_BOND_REQ message)

HCI_MITM_REQUIRED_DEDICATED_BONDING (MITM required when the application wishes to do a Dedicated Bonding, by sending a SC_BOND_REQ message)

HCI_MITM_NOT_REQUIRED_GENERAL_BONDING (Deprecated shall not be used. It is handled by the security controller itself)

HCI_MITM_REQUIRED_GENERAL_BONDING (Deprecated shall not be used. It is handled by the security controller itself)

Some of the options are marked as deprecated as their does not affect the SSP procedure that will be used. It only addresses the intent of the local device.

Note, it is strongly recommended that authRequirements is set to HCI_MITM_NOT_REQUIRED_DEDICATED_BONDING. Because if a profile like SAP demands MITM and a Dedicated Bonding has been performed with success, the security will be raised automatically, by starting a new bonding/pairing procedure, whenever a connection is established to this profile, see also section 3.1.2

leAuthRequirements Security requirements for Low Energy links. The following values are defined (can be OR'ed together)

Value	Parameter description
CSR_BT_SC_LE_SECURITY_ENCRYPTION	Require Low Energy links to be encrypted
CSR_BT_SC_LE_SECURITY_MITM	Require Man-in-the-Middle protection for Low Energy links.

4.18 CSR_BT_SC_MODE

Primitives \ Parameters	type	duration	mode
CSR_BT_SC_MODE_REQ	✓	✓	✓
CSR_BT_SC_MODE_IND	✓		✓

Table 21: CSR_BT_SC_MODE Primitives

Description

Set the local device's idle mode behaviour. This can also be done by setting the CSR_BT_DEFAULT_CSR_BT_SC_MODE define in csr_bt_usr_config.h. This parameter determines how the SC will react towards unsolicited paring attempts. When the device changes idle mode behaviour a CSR_BT_SC_MODE_IND will be sent to the application.

Note: The application must make sure that the device is connectable if the device should allow pairing.

Parameters

type — The signal identity, CSR_BT_SC_MODE_REQ/IND

duration — The duration of this mode. If the mode should be applied forever, the CSR_BT_INFINITE_TIME define should be used.

mode — The idle mode of the local device. The following values are applicable (from csr_bt_profiles.h):

CSR_BT_SEC_MODE_PAIRABLE - allow pairing

CSR_BT_SEC_MODE_NON_PAIRABLE - reject pairing

CSR_BT_SEC_MODE_NON_BONDABLE - allow pairing if remote device does not request dedicated/general bonding else reject.

Note: The define CSR_BT_SEC_MODE_NON_BONDABLE is only applicable when the local device supports SSP.

4.19 CSR_BT_SC_SET_EVENT_MASK

Parameters / Primitives	type	eventMask
CSR_BT_SC_SET_EVENT_MASK_REQ	✓	✓

Table 22: CSR_BT_SC_SET_EVENT_MASK Primitive

Description

Set the Security Controller's event mask. This control which events are forwarded to the application. This can also be done by setting the CSR_BT_SC_CSR_BT_DEFAULT_EVENT_MASK define in csr_bt_usr_config.h

Parameters

type — The signal identity, CSR_BT_SC_SET_EVENT_MASK_REQ

eventMask — The following values are applicable (from csr_bt_profiles.h) and can be OR'ed. Setting one of the following values means that the application will receive the corresponding primitive:

- CSR_BT_SC_EVENT_MASK_NONE – No additional primitives will be forwarded to the application. This means that rebond attempts are automatically accepted by the SC.
- CSR_BT_SC_EVENT_MASK_REBOND – all rebond events are forwarded to the application, which can then reject or accept it. See Section 4.20
- CSR_BT_SC_EVENT_MASK_PAIR – all pairing requests are forwarded to the application, which can then reject or accept it. See Section 4.21.
- CSR_BT_SC_EVENT_MASK_AUTHORISE_EXTENDED – the CSR_BT_SC_AUTHORISE_EXT_IND primitive replaces the CSR_BT_SC_AUTHORISE_IND primitive. See Section 4.9.

Note, the default eventMask is set to CSR_BT_SC_EVENT_MASK_NONE.

4.20 CSR_BT_SC_REBOND

Parameters / Primitives	type	cod	deviceAddr	deviceName	accept	addressType
CSR_BT_SC_REBOND_IND	✓	✓	✓	✓		✓
CSR_BT_SC_REBOND _RES	✓		✓		✓	

Table 23: CSR_BT_SC_REBOND Primitives

Description

Request for accepting/rejecting a rebond procedure. The application should forward the request to the user, e.g. through the MMI and let the user decide on which action to perform.

Parameters

type	The signal identity, CSR_BT_SC_REBOND_IND/RES
cod	Class of device of peer device with which a rebond is attempted. The CoD is defined in the Bluetooth specification assigned numbers. Please note that it cannot be guaranteed that the cod is set.
deviceAddr	The Bluetooth® device address of the device to attempt rebonding with.
deviceName	The name of the remote device.
accept	TRUE to accept the rebond attempt, FALSE to reject.
addressType	Address type. Not used. Always CSR_BT_ADDR_PUBLIC.

4.21 CSR_BT_SC_SSP_PAIRING_IND

Parameters / Primitives	type	cod	deviceAddr	deviceName	authValid	authRequirements	paired	accept	authorised	addDevice	initiator	addressType	localAuthRequirements
CSR_BT_SC_SSP_PAIRING_IND	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓
CSR_BT_SC_SSP_PAIRING_RES	✓		✓					✓	✓	✓		✓	

Table 24: CSR_BT_SC_SSP_PAIRING Primitives

Description

If the application has enabled pairing indications (see Section 4.19), it will receive a CSR_BT_SC_SSP_PAIRING_IND before starting pairing. This message can be used for triggering a change in the device's authentication requirements (see Section 4.17) or collecting OOB data for the remote device (see Section 4.22), If the application wishes to change either the authentication requirements or OOB data for the remote device, it needs to do so *before* responding with a CSR_BT_SC_SSP_PAIRING_RES.

Note, the *authorised* and *addDevice* parameters are used if the OOB authentication model is chosen. In this case the application will not receive any pairing primitives (such as CSR_BT_SC_SSP_COMPARE_IND), since pairing/bonding is able to complete without any application interaction.

Parameters

type	The signal identity, CSR_BT_SC_SSP_PAIRING_IND/RES
cod	Class of device of peer device with which a rebond is attempted. The CoD is defined in the Bluetooth specification assigned numbers. Please note that it cannot be guaranteed that the cod is set.
deviceAddr	The Bluetooth® device address of the device to attempt rebonding with.
deviceName	The name of the remote device.
authValid	Indicate whether the authRequirements field is valid or not
authRequirements	The authentication requirements sent by the remote device. See HCI_MITM_XXX defines in hci.h for possible values.
localAuthRequirements	The authentication requirements of the local device
paired	Indicates whether there is a bond between the local and remote device. The following values are applicable (from csr_bt_sc_prim.h): CSR_BT_SC_PAIRING_NONE - no bond exists CSR_BT_SC_PAIRING_LEGACY - paired with legacy method CSR_BT_SC_PAIRING_NO_MITM - paired with SSP and no MITM CSR_BT_SC_PAIRING_MITM - paired with SSP and MITM
accept	TRUE to accept the pairing attempt, FALSE to reject.
addDevice	If TRUE the device is added to NVS else not.
authorised	TRUE if authorisation is automatic granted, e.g. the device is marked as trusted. FALSE if not

Initiator	TRUE if this indication was triggered by the local host
addressType	Address type. Not used. Always CSR_BT_ADDR_PUBLIC.

4.22 CSR_BT_SC_ADD_REMOTE_OOB_DATA

Primitives \ Parameters	type	deviceAddr	oobHashC	oobRandR
CSR_BT_SC_ADD_REMOTE_OOB_DATA_REQ	✓	✓	✓	✓

Table 25: CSR_BT_SC_ADD_REMOTE_OOB_DATA Primitive

Description

The CSR_BT_SC_ADD_REMOTE_OOB_DATA_REQ is used for adding OOB data for the given remote device. The Security Controller is only able to store one set of OOB data and will overwrite any existing data if multiple CSR_BT_SC_ADD_REMOTE_OOB_DATA_REQ are issued. The OOB data will automatically be discarded after use by the Security Controller.

Note, the CSR_BT_SC_ADD_REMOTE_OOB_DATA_REQ *shall* only be used if pairing indications have previously been enabled (see Section 4.19).

Parameters

type	The signal identity, CSR_BT_SC_ADD_REMOTE_OOB_DATA_REQ.
deviceAddr	The Bluetooth device address of the device, given previously in the CSR_BT_SC_BOND_REQ
oobHashC	The remote device's OOB Hash C value
oobRandR	The remote device's OOB Rand R value.

4.23 CSR_BT_SC_READ_LOCAL_OOB_DATA

Primitives \ Parameters	type	phandle	deviceAddr	oobHashC	oobHashCLength	oobRandR	oobRandRLenth	resultCode	resultSupplier	addressType
CSR_BT_SC_READ_LOCAL_OOB_DATA_REQ	✓	✓								
CSR_BT_SC_READ_LOCAL_OOB_DATA_CFM	✓		✓	✓	✓	✓	✓	✓	✓	✓

Table 26: CSR_BT_SC_READ_LOCAL_OOB_DATA Primitives

Description

The CSR_BT_SC_READ_LOCAL_OOB_DATA_REQ is used for reading the local device's OOB data. This OOB data can then be sent to another device using a different (and secure) transport. If either side has OOB data and pairing indications are enabled, the OOB data authentication model will be selected for the subsequent pairing/bonding procedure.

Parameters

type	The signal identity, CSR_BT_SC_READ_LOCAL_OOB_DATA_REQ/CFM.
deviceAddr	The Bluetooth device address of the device, given previously in the CSR_BT_SC_BOND_REQ
oobHashC	The remote device's OOB Hash C value
oobHashCLength	Length of oobHashC parameter
oobRandR	The remote device's OOB Rand R value.
oobRandRLenth	Length of oobRandR parameter
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
addressType	Address type. For classic Bluetooth this is always CSR_BT_ADDR_PUBLIC. For Low Energy devices, this may an integral part of the device address and may be either CSR_BT_ADDR_PUBLIC or CSR_BT_ADDR_PRIVATE.

4.24 CSR_BT_SC_SSP_KEYPRESS_NOTIFICATION

Primitives \ Parameters	type	deviceAddr	notificationType	addressType
CSR_BT_SC_SSP_KEYPRESS_NOTIFICATION_REQ	✓	✓	✓	✓
CSR_BT_SC_SSP_KEYPRESS_NOTIFICATION_IND	✓	✓	✓	✓

Table 27: CSR_BT_SC_SSP_KEYPRESS_NOTIFICATION Primitives

Description

This primitive may be issued by the application after receiving a CSR_BT_SC_SSP_PASSKEY_IND. This will notify the remote device that the local user is entering the passkey. The remote device can use this to extend any timer it may use to allow more time to complete pairing.

Parameters

type — The signal identity, CSR_BT_SC_SSP_KEYPRESS_NOTIFICATION_REQ/IND

deviceAddr — The Bluetooth® device address of the device that should receive/sending the notification.

notificationType — The notification type. The following values are applicable (from hci.h):

HCI_NOTIFICATION_TYPE_PASSKEY_STARTED
HCI_NOTIFICATION_TYPE_PASSKEY_DIGIT_ENTERED
HCI_NOTIFICATION_TYPE_PASSKEY_DIGIT_ERASED
HCI_NOTIFICATION_TYPE_PASSKEY_CLEARED
HCI_NOTIFICATION_TYPE_PASSKEY_COMPLETED

addressType — Address type. Always CSR_BT_ADDR_PUBLIC.

4.25 CSR_BT_SC_CONFIG

Primitives \ Parameters	type	configMask
CSR_BT_SC_CONFIG_REQ	✓	✓

Table 28: CSR_BT_SC_CONFIG Primitives

Description

Used for configuring the Security controller, the 'configMask' is per default set to CST_BT_SC_CONFIG_MASK_USE_STANDARD.

Parameters

type — The signal identity, CSR_BT_SC_CONFIG_REQ.

configMask — Bitmask containing a set of behaviour defining flags that the Security Controller can be configured to use. Note unused bits are reserved, and must be set to zero

The configMask values are defined in csr_bt_sc_prim.h.

Value	Parameter Description
CSR_BT_SC_CONFIG_MASK_USE_STANDARD	Use default setting.
CSR_BT_SC_CONFIG_MASK_FORCE_NAME_UPDATE	Force the name to be updated doing a pairing procedure
CSR_BT_SC_CONFIG_MASK_ALWAYS_TRY_MITM	Always try to use MITM, but allow non-MITM to be used.

4.26 CSR_BT_SC_ACCEPT_BOND

Primitives \ Parameters	type	phandle	time	resultCode	resultSupplier
CSR_BT_SC_ACCEPT_BOND_REQ	✓	✓	✓		
CSR_BT_SC_CANCEL_ACCEPT_BOND_REQ	✓	✓			
CSR_BT_SC_CANCEL_ACCEPT_BOND_IND	✓			✓	✓

Table 29: CSR_BT_SC_ACCEPT_BOND Primitives

Description

Used for making the local device bondable or for making it non-bondable when no service records are registered.

Note that there is no confirmation primitive for neither the CSR_BT_SC_ACCEPT_BOND_REQ nor the CSR_BT_SC_CANCEL_ACCEPT_BOND_REQ. The SC will send the primitive CSR_BT_SC_CANCEL_ACCEPT_BOND_IND when the device is no longer bondable, either due to timeout, an error, or because the application issues the CSR_BT_SC_CANCEL_ACCEPT_BOND_REQ.

Parameters

type	The signal identity, CSR_BT_SC_ACCEPT_BOND_REQ, CSR_BT_SC_CANCEL_ACCEPT_BOND_REQ or CSR_BT_SC_CANCEL_ACCEPT_BOND_IND.
phandle	The identity of the calling process that shall receive the CSR_BT_SC_CANCEL_ACCEPT_BOND_IND.
time	Time in milliseconds when the CSR_BT_SC_ACCEPT_BOND_REQ is valid. If 0, it means forever.
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

4.27 CSR_BT_SC_ENCRYPTION_KEY_SIZE

Parameters / Primitives	type	appHandle	address	keySize	resultCode	resultSupplier
CSR_BT_SC_ENCRYPTION_KEY_SIZE_REQ	✓	✓	✓			
CSR_BT_SC_ENCRYPTION_KEY_SIZE_CFM	✓	✓	✓	✓	✓	✓

Table 30: CSR_BT_SC_ENCRYPTION_KEY_SIZE Primitives

Description

This API is used to read the encryption key size used towards a given peer address. The key size may be obtained for any active link (including Low Energy connections). If no active link exists towards the given address, the SC will attempt to read the key size from SC_DB.

Parameters

type	The signal identity, CSR_BT_SC_ENCRYPTION_KEY_SIZE_REQ or CSR_BT_SC_ENCRYPTION_KEY_SIZE_CFM.
phandle	The identity of the calling process that shall receive the CSR_BT_SC_ENCRYPTION_KEY_SIZE_CFM.
keySize	The encryption key size in octets.
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

4.28 CSR_BT_SC_LE_KEY_DISTRIBUTION

Primitives \ Parameters	type	keyDistribution
CSR_BT_SC_LE_KEY_DISTRIBUTION_REQ	✓	✓

Table 31: CSR_BT_SC_LE_KEY_DISTRIBUTION Primitives

Description

This API sets the Low Energy key distribution. The key distribution determines what keys are exchanged when two Low Energy devices bond. Setting the key distribution will take effect on all future LE security procedures. There is no confirmation for this request.

Parameters

type — The signal identity, CSR_BT_SC_LE_KEY_DISTRIBUTION_REQ.

phandle — The key distribution used. The following values are defined:

Value	Parameter Description
CSR_BT_SC_LE_KEYDIST_ENCRYPTION	Encryption keys are exchanged.
CSR_BT_SC_LE_KEYDIST_ID	Identity resolving keys are exchanged.
CSR_BT_SC_LE_KEYDIST_SIGN	Signing keys are exchanged.

4.29 CSR_BT_SC_SET_ENCRYPTION_KEY_SIZE

Primitives \ Parameters	type	minKeySize	maxKeySize
CSR_BT_SC_SET_ENCRYPTION_KEY_SIZE_REQ	✓	✓	✓

Table 32: CSR_BT_SC_SET_ENCRYPTION_KEY_SIZE Primitives

Description

This API sets the minimum and maximum encryption key sizes used for future security procedures. There is no confirmation for this request.

Parameters

type	The signal identity, CSR_BT_SC_SET_ENCRYPTION_KEY_SIZE_REQ.
minKeySize	The minimum encryption key size.
maxKeySize	The maximum encryption key size.

4.30 CSR_BT_SC_LE_SECURITY

Parameters / Primitives	type	address	addedToScDbList	resultCode	resultSupplier
CSR_BT_SC_LE_SECURITY_IND	✓	✓	✓	✓	✓

Table 33: CSR_BT_SC_LE_SECURITY Primitives

Description

This indication is sent to the registered application whenever a Low Energy security procedure has completed either successfully or if the procedure failed.

This signal is similar to CSR_BT_SC_BOND_CFM/IND, except it is only sent to the application for Low Energy security procedures.

Parameters

type	The signal identity, CSR_BT_SC_LE_SECURITY_IND.
address	The typed address of the peer with whom security has completed.
addedToScDbList	If TRUE the device is added to NVS else not
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

5 Document References

Document	Reference
Bluetooth Core Specification v.1.1, v1.2 and v2.0, section N/A	[BT
The Bluetooth Specification, the Generic Access Profile, section K:1	[GAP]
Bluetooth Security Architecture, Version 1.0, section N/A	[SEC]
The whitepaper: Bluetooth User Interface Flow Diagrams for Bluetooth Secure Simple Pairing. Revision v1.0.	[UIF]
CSR Synergy Bluetooth, Porting guide, gu-0102-porting.pdf	[PORT]

Terms and Definitions

BlueCore®	Group term for CSR's range of Bluetooth wireless technology chips
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
CSR	Cambridge Silicon Radio
UniFi™	Group term for CSR's range of chips designed to meet IEEE 802.11 standards
SC	Security Controller
SIG	Special Interest Group
NVS	Non-Volatile Storage
PPP	Point to Point Protocol
MITM	Man-in-the-middle
SSP	Secure Simple Pairing
IPSEC	IP Security

Document History

Revision	Date	History
1	26 SEP 11	Ready for release 18.2.0
2	23 JAN 12	Ready for release 18.2.2

TradeMarks, Patents and Licences

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

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.

Life Support Policy and Use in Safety-critical Compliance

CSR's products are not authorised for use in life-support or safety-critical applications. Use in such applications is done at the sole discretion of the customer. CSR will not warrant the use of its devices in such applications.

Performance and Conformance

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